
PREMIER REFERENCE SOURCE
SOFTWARE APPLICATIONS
Concepts, Methodologies, Tools, and Applications
Volume I
SOFTWARE APPLICATIONS
Concepts, Methodologies, Tools, and Applications
Volume I
REFERENCE
Volume II
Volume III
Volume IV
Volume V
Volume VI

Software Applications: Concepts, Methodologies, Tools, and Applications

Pierre F. Tiako
Langston University, USA

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content: Kristin Klinger
Senior Managing Editor: Jamie Snavely
Managing Editor: Jeff Ash
Assistant Managing Editor, MVB: Michael Brehm
Assistant Managing Editor: Carole Coulson
Typesetters: Mandy Appicello, Jeff Ash, Kim Barger, Lindsey Bergman, Michael Brehm, Carole Coulson, Elizabeth Duke, Jennifer Johnson, Christopher Hrobak, Jamie Snavely, Larissa Vinci, Sean Woznicki
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanbookstore.com

Library of Congress Cataloging-in-Publication Data

Software applications : concepts, methodologies, tools, and applications /

Pierre F. Tiako, editor.

v. ; cm.

Includes bibliographical references and index.

Summary: "This set of books contains a comprehensive collection of over 300

authoritative contributions from top influential experts in the field of

software applications"--Provided by publisher.

ISBN 978-1-60566-060-8 (hardcover) -- ISBN 978-1-60566-061-5 (ebook) 1.

Computer software. 2. Software engineering. I. Tiako, Pierre F.

QA76.754S64433 2009

005.1--dc22

2009001521

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Editor-in-Chief

Associate Editors

Editorial Advisory Board

Additional Research Collections found in the "Contemporary Research in Information Science and Technology" Book Series

Data Mining and Warehousing: Concepts, Methodologies, Tools, and Applications
John Wang, Montclair University, USA • 6-volume set • ISBN 978-1-60566-056-1

Electronic Business: Concepts, Methodologies, Tools, and Applications
In Lee, Western Illinois University • 4-volume set • ISBN 978-1-59904-943-4

Electronic Commerce: Concepts, Methodologies, Tools, and Applications
S. Ann Becker, Florida Institute of Technology, USA • 4-volume set • ISBN 978-1-59904-943-4

Electronic Government: Concepts, Methodologies, Tools, and Applications
Ari-Veikko Anttiroiko, University of Tampere, Finland • 6-volume set • ISBN 978-1-59904-947-2

Knowledge Management: Concepts, Methodologies, Tools, and Applications
Murray E. Jennex, San Diego State University, USA • 6-volume set • ISBN 978-1-59904-933-5

Information Communication Technologies: Concepts, Methodologies, Tools, and Applications
Craig Van Slyke, University of Central Florida, USA • 6-volume set • ISBN 978-1-59904-949-6

Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications
Vijayan Sugumaran, Oakland University, USA • 4-volume set • ISBN 978-1-59904-941-0

Information Security and Ethics: Concepts, Methodologies, Tools, and Applications
Hamid Nemati, The University of North Carolina at Greensboro, USA • 6-volume set • ISBN 978-1-59904-937-3

Medical Informatics: Concepts, Methodologies, Tools, and Applications
Joseph Tan, Wayne State University, USA • 4-volume set • ISBN 978-1-60566-050-9

Mobile Computing: Concepts, Methodologies, Tools, and Applications
David Taniar, Monash University, Australia • 6-volume set • ISBN 978-1-60566-054-7

Multimedia Technologies: Concepts, Methodologies, Tools, and Applications
Syed Mahbubur Rahman, Minnesota State University, Mankato, USA • 3-volume set • ISBN 978-1-60566-054-7

Virtual Technologies: Concepts, Methodologies, Tools, and Applications
Jerzy Kisielnicki, Warsaw University, Poland • 3-volume set • ISBN 978-1-59904-955-7

List of Contributors

Contents

Volume I

Section I. Fundamental Concepts and Theories

This section serves as the foundation for this exhaustive reference tool by addressing crucial theories essential to the understanding of software applications computing. Chapters found within these pages provide an excellent framework in which to position software applications within the field of information science and technology. Individual contributions provide overviews of open source software, software testing, social software, and software quality, while also exploring critical stumbling blocks of this field. Within this introductory section, the reader can learn and choose from a compendium of expert research on the elemental theories underscoring the research and application of software applications.

Section II. Development and Design Methodologies

This section provides in-depth coverage of conceptual architectures, frameworks and methodologies related to the design and implementation of software systems and applications. Throughout these contributions, research fundamentals in the discipline are presented and discussed. From broad examinations to specific discussions on particular frameworks and infrastructures, the research found within this section spans the discipline while also offering detailed, specific discussions. Basic designs, as well as abstract developments, are explained within these chapters, and frameworks for designing successful software applications.

Volume II

Section III. Tools and Technologies

This section presents extensive coverage of the technology that both derives from and informs software applications. These chapters provide an in-depth analysis of the use and development of innumerable devices and tools, while also providing insight into new and upcoming technologies, theories, and instruments that will soon be commonplace. Within these rigorously researched chapters, readers are presented with examples of the tools that facilitate and support software design and engineering. In addition, the successful implementation and resulting impact of these various tools and technologies are discussed within this collection of chapters.

Volume III

Section IV. Utilization and Application

This section introduces and discusses the ways in which information technology has been used to shape the realm of software applications and proposes new ways in which IT-related innovations can be implemented within organizations and in society as a whole. These particular selections highlight, among other topics, intelligent software agents in e-commerce and utilizing open source software in organizations. Contributions included in this section provide excellent coverage of today's changing environment and insight into how evolutions in software applications impact the fabric of our present-day global village.

Volume IV

Section V. Organizational and Social Implications

This section includes a wide range of research pertaining to the social and organizational impact of software applications. Chapters introducing this section analyze open source software communities, while later selections discuss organizational modeling and the analysis of user interfaces. The inquiries and methods presented in this section offer insight into the implications of software applications at both a personal and organizational level, while also emphasizing potential areas of study within the discipline.

Volume V

Section VII. Critical Issues

This section addresses conceptual and theoretical issues related to the field of software applications, which include ethics in software engineering, software piracy, and morality in free and open source software. Within these chapters, the reader is presented with analysis of the most current and relevant conceptual inquires within this growing field of study. Overall, contributions within this section ask unique, often theoretical questions related to the study of software applications and, more often than not, conclude that solutions are both numerous and contradictory.

Volume VI

Section VIII. Emerging Trends

This section highlights research potential within the field of software applications while exploring uncharted areas of study for the advancement of the discipline. Chapters within this section highlight evolutions in social software, state-of-the-art agile software methods, and modeling large software systems. These contributions, which conclude this exhaustive, multi-volume set, provide emerging trends and suggestions for future research within this rapidly expanding discipline.

Preface

Since the introduction of von Neumann architecture in the 1940s, which stipulated the critical division between hardware and software, engineers and programmers alike have been striving to build bigger, faster and cheaper software applications. This end goal of creating the most elegant program utilizing the least resources while performing the most work has seen a plethora of developments over the last several decades. Every innovation from the early programming languages of COBOL and FORTRAN to the more recent move toward the agile programming method has informed the creation of software applications designed to meet increasing demands and to answer a greater number of needs in the most efficient way possible.

Now, with an entire generation of computer users who have come to expect the integration of software applications in their everyday lives, there is a call for greater research and development and more efficient programming in an ever widening spectrum of disciplines from business, government, and education to everyday use and recreation. To keep up with the demand for newer and better software, practitioners and researchers must thrive within a fluid environment. They need be cognizant of constant advances that must be considered and responded to in order to ensure the success of developing software applications.

Due to the ever changing landscape of software applications, it is a constant challenge for researchers and experts in the development of software applications to absorb the volume of innovations which will inform and quickly outdate current software applications. Information Science Reference is pleased to offer a six-volume reference collection on this rapidly growing discipline, in order to empower students, researchers, academicians, and practitioners with a comprehensive understanding of the most critical areas within this field of study. This collection provides the most comprehensive, in-depth, and recent coverage of all issues related to the development of cutting-edge software applications, as well as a single reference source on all conceptual, methodological, technical and managerial issues, and the opportunities, future challenges and emerging trends related to the development of software applications,

Entitled **"Software Applications: Concepts, Methodologies, Tools, and Applications,"** this collection is organized in eight distinct sections, providing the most wide-ranging coverage of topics such as: 1) Fundamental Concepts and Theories; 2) Development and Design Methodologies; 3) Tools and Technologies; 4) Utilization and Application; 5) Organizational and Social Implications; 6) Managerial Impact; 7) Critical Issues; and 8) Emerging Trends. The following provides a summary of what is covered in each section of this multi-volume reference collection:

Section I, ***Fundamental Concepts and Theories***, offers an extensive view of the foundational theories and concepts which inform the development of software applications. This section begins with the chapter "A Historical Analysis of the Emergence of Free Cooperative Software Production," by Nicolas Jullien which describes the vacillation of open source programming from its popularity in the 1950s to its near disappearance in the 1980s up through its most recent boom in popularity. Other chapters in the section

such as, "Software Modernization of Legacy Systems for Web Services Interoperability" by Chia-Chu Chiang and "Intelligent Analysis of Software Maintenance Data" by Marek Reformat, Petr Musilek and Efe Igbide, deal with the critical issue of analysis and adaptation for the maintenance of software applications. "Malicious software" by Thomas M. Chen and Gregg W. Tally discusses the problematic topic of malware. In this chapter, Chen and Tally present malware as one of the most costly and widespread types of virtual attacks on organizations. This section also contains chapters such as David Lo and Siau-Cheng Khoo's "Mining Software Specifications" which discusses the emerging area of data mining, focusing on drawing out software specifications based on the behavior of a software application.

In Section II, ***Development and Design Methodologies,*** the creative stages of software application design are explored through developmental strategies and design methodologies. This section opens with "Ontology Based Object-Oriented Domain Modeling: Representing Behavior" by Jeorg Evermann and Yair Wand. Evermann and Wand describe the usefulness of the object-oriented software modeling language to the early conceptual stage of application domain analysis for which no specific language is accepted. More application specific chapters, such as "Building an LMS with Ubiquitous Software" by Michael Rees and Charles Herring, can be found later in the section. In their chapter, Rees and Herring present their experience during the development and trial run of an off-the-shelf learning management system that utilized Microsoft Office Systems at its core. In addition, this section contains selections like "Bug Fixing Practices within Free/Libre Open Source Software Development Teams" by Kevin Crowston and Barbara Scozzi which examines the success of debugging in open source software despite the fact that it is primarily developed by distributed teams.

Section III, ***Tools and Technologies,*** begins with Kerstin Fink and Christian Plodder's chapter "Knowledge Management Toolkit for SMEs" which presents four knowledge processes designed to help small and medium enterprises take advantage of their intellectual capital. Additional tools for free software projects are presented in "Tools for the Study of the Usual Data Sources found in Libre Software Projects" by Gregorio Robles, Jesús M. González-Barahona, Daniel Izquierdo-Cortazar and Israel Herraiz. Several chapters, such as "A Software Tool for Reading DICOM Directory Files" by Ricardo Villegas, Guillermo Montilla and Hyxia Villegas and "LOGIC-Minimiser: A Software Tool to Enhance Teaching and Learning Minimization of Boolean Expressions" by Nurul I. Sarkar and Khaleel I. Petrus discuss individual tools with specific uses in mind. This overview of various tools and technologies for software applications comes to a close with "A Pliant-Based Software Tool for Courseware Development" by Marcus Vinicius dos Santos, Isaac Woungang and Moses Nyongwa. In this chapter, the authors discuss the use of the Pliant software framework for web-based courseware development.

Section IV, ***Utilization and Application***, describes the implementation of technologies, methodologies, and theories related to software applications. This section contains chapters such as "Online Synchronous vs. Asynchronous Software Training Through the Behavioral Modeling Approach: A Longitudinal Field Experiment" by Charlie C. Chen and R. S. Shaw which compares and contrasts the implementation of a particular modeling approach in three different learning environments. "Patchwork Prototyping with Open Source Software" by M. Cameron Jones, Ingbert R. Floyd and Michael B. Twidale proposes the utilization of a prototype which has been patched together from existing open source resources to provide real life feedback for software application development. Lawrence R. LaForge's chapter, entitled "Teaching Operations Management with Enterprise Software" rounds out the section describing the use of enterprise software to provide operations management students with an understanding of how operations management decisions affect various aspects of a manufacturing enterprise.

Section V, ***Organizational and Social Implications***, delves into the vital conversation surrounding the issues that arise from developments in software design. "The Impact of Ideology on the Organizational Adoption of Open Source Software" by Kris Ven and Jan Verelst begins the discussion of how ideology

can factor into the pragmatic decisions of open source software adoption. In their article "Volunteers in Large Libre Software Projects: A Quantitative Analysis Over Time," Martin Michlmayr, Gregorio Robles, and Jesus M. Gonzalez-Barahona investigate the effectiveness of drawing participants to open source software. Jennifer Papin-Ramcharan and Frank Soodeen offer their perspective on the use of open source software in developing countries in "Open Source Software: A Developing Country View." Other important social issues are also presented in "Facilitating E-Learning with Social Software: Attitudes and Usage from the Student's Point of View," by Reinhard Bernsteiner, Herwig Ostermann, and Roland Staudinger, and "Planning and Managing the Human Factors for the Adoption and Diffusion of Object-Oriented Software Development Processes," by Magdy K. Serour.

This section illuminating the interaction of human beings and software is completed by "Mental Contents in Interacting with a Multiobjective Optimization Program" by Pertti Saariluoma, Katja Kaario, Kaisa Miettinen, and Marko M. Mäkelä, which uses a specific psychological theory to analyze interaction with professional software.

Section VI, ***Managerial Impact,*** presents contemporary coverage of the managerial applications and implications in the field of software. "Combining Tailoring and Evolutionary Software Development for Rapidly Changing Business Systems" by Jeanette Eriksson and Yvonne Dittrich begins this section with the analysis of a case study performed to evaluate the usability of software for business systems. The principles of management in software development projects and optimization tools for managerial decision making, especially in the environment of small IT companies, are thoroughly discussed in "Computer-Aided Management of Software Development in Small Companies," by Lukáš Pichl and Takuya Yamano. Also included in this section is the chapter "A Survey of Competency Management Software Information Systems in the Framework of Human Resources Management," by Alfonso Urquiza, which shows the Competency Management Software evolution from a previous fragmented market situation to a much more integrated scenario in which best of breed single-function oriented product preferences are now swiftly moving to the Enterprise Resource Planning (ERP) type of architecture. The many tools available to managers are also explored through selections such as "Channel Optimization for On Field Sales Force by Integration of Business Software on Mobile Platforms" by Rishi Kalra and Amit Nanchahal, "Revenue Models in the Open Source Software Business" by Risto Rajala, Jussi Nissilä and Mika Westerlund, and "Knowledge Management and organizational Performance in the egyptian software Firms," by Ahmed Seleim and Omar Khalil. The chapters in this section provide a comprehensive survey of the many intersections between software and management.

Section VII, ***Critical Issues,*** surveys some of the most important considerations that impact software development and are in turn influenced by software. Dave Yeats offers his research on the subtle philosophical differences represented in two open source software movements in "Morality and Pragmatism in Free Software and Open Source." The more technical issues are also discussed in selections such as "A Graphical User Interface (GUI) Testing Methodology," by Zafar Singhera, Ellis Horowitz and Abad Shah, "A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications," by W. K. Chan, S. C. Cheung and Karl R. P. H. Leung, and "A Formal Verification and Validation Approach for Real-Time Databases," by Pedro Fernandes Ribeiro Neto, Maria Lígia Barbosa Perkusich, Hyggo Oliveira de Almeida and Angelo Perkusich, which raise issues surrounding testing. The issues raised in this section are, in many ways, the crux of software engineering today, giving the reader a full understanding of the research surrounding the most pressing questions found in this discipline.

The concluding section of this authoritative reference tool, ***Emerging Trends,*** highlights research potential within the field of software while exploring uncharted areas of study for the advancement of the discipline. "Bridging the Gap between Agile and Free Software Approaches: The Impact of Sprinting" by J. Paul Adams and Andrea Capiluppi presents an analysis of the effect of sprinting—commonly used

in the agile programming method—on the free software approach to programming. In addition, this section highlights future trends related to social software in chapters such as "Social Software and Language Acquisition" by Sarah Guth and Corrado Petrucco and "Social Software Trends in Business" by Peter Burkhardt. The trends highlighted in this section present a broad range of topics for future research and implementation leading the continued advancement of software application development.

Although the primary organization of the contents in this multi-volume collection is based on its eight sections, offering a progression of coverage of the important concepts, methodologies, technologies, applications, social issues, and emerging trends, the reader can also identify specific contents by utilizing the extensive indexing system listed at the end of each volume. Furthermore to ensure that the scholar, researcher and educator have access to the entire contents of this multi-volume set as well as additional coverage that could not be included in the print version of this publication, the publisher will provide unlimited multi-user electronic access to the online aggregated database of this collection for the life of edition, free of charge when a library purchases a print copy. This aggregated database provides far more contents than what can be included in the print version in addition to continual updates. This unlimited access, coupled with the continuous updates to the database ensures that the most current research is accessible to knowledge seekers.

Advances in software applications have been both immense and pervasive in the last several decades as the discipline has continued to grow and thrive with each new development. From the early days, through the "software crisis" of the late 60s and early 70s and on to the present day, software applications have become increasingly vital to the everyday user. As software applications become a ubiquitous part of everyday life, the demand for larger, faster, and more cost effective software applications continues to grow. This ever increasing demand will lead to greater improvements in this burgeoning discipline. Access to the most up-to-date research findings and firm knowledge of established techniques and lessons learned from other researchers or practicing software developers will facilitate the discovery and invention of more effective methodologies. This will allow for a greater range of applications and the creation of better software to serve a greater number of users.

The diverse and comprehensive coverage of software applications in this six-volume authoritative publication will contribute to a better understanding of all topics, research, and discoveries in this developing, significant field of study. Furthermore, the contributions included in this multi-volume collection series will be instrumental in the expansion of the body of knowledge in this enormous field, resulting in a greater understanding of the fundamentals while fueling the research initiatives in emerging fields. We at *Information Science Reference*, along with the editor of this collection and the publisher, hope that this multi-volume collection will become instrumental in the expansion of the discipline and will promote the continued growth of all aspects of software applications.

Section I
Fundamental Concepts and Theories

This section serves as the foundation for this exhaustive reference tool by addressing crucial theories essential to the understanding of software applications computing. Chapters found within these pages provide an excellent framework in which to position software applications within the field of information science and technology. Individual contributions provide overviews of open source software, software testing, social software, and software quality, while also exploring critical stumbling blocks of this field. Within this introductory section, the reader can learn and choose from a compendium of expert research on the elemental theories underscoring the research and application of software applications.

Chapter 1.1
A Historical Analysis of the Emergence of Free Cooperative Software Production

Nicolas Jullien
LUSSI TELECOM Bretagne-M@rsouin, France

INTRODUCTION

Whatever its name, **Free/Libre or Open Source Software** (FLOSS), diffusion represents one of the main evolutions of the Information Technology (IT) industry in recent years. Operating System Linux, or Web server Apache (more than 60% market share on its market), database MySQL or PHP languages are some examples of broadly-used FLOSS programs. One of the most original characteristics of this movement is its collective, cooperative **software development** organization in which a growing number of firms is involved (some figures in Lakhani & Wolf (2005)). Of course, programs, because they are codified information, are quite easy to exchange, and make the **cooperation** easier than in other industries. But, as pointed out by Stallman (1998), if sharing pieces of software within firms was a dominant practice in the 1950's, it declined in the 1970's, and almost disappeared in the 1980's, before regaining and booming today.

This article aims at explaining the evolution (and the comeback) of a cooperative, non-market production.

In the first part, we explain the decrease of cooperation as a consequence of the evolution of the computer users, of their demand, and of the industrial organization constructed to meet this demand. This theoretical and historical framework is used in the second part to understand the renewal of a cooperative organization, the FLOSS phenomenon, first among computer-literate users, and then within the industry.

SOFTWARE IN THE HISTORY OF THE COMPUTER INDUSTRY

Among the few works of reference existing on the evolution of the **computer industry**, we use the following as our basis: Mowery (1996), Genthon (1995), and Dréan (1996). Richardson (1997) and

Horn (2004) have analyzed the specificities of the software industry.

If these authors do not agree on the number of periods that this industry has gone through since its birth at the end of World War II, they agree on two main ruptures:

- The arrival of the IBM 360 series, in the early 1960's, opening the mainframe and mini period when, thanks to the implementation of an operating system, a standard machine could be sold to different clients, but also a program could be used on a family of computers, of different power, and not abandoned when the machine was obsolete; and
- The arrival of the PC, and specifically the IBM PC, in the early 1980's, when the computer became a personal information management tool, produced by different actors.

Each of these periods is characterized by a technology which has allowed firms to propose new products to new consumers, changing the dominant producer-user relations. This has had an impact on the degree of cooperation in the **software production**.

Period 1: The Industry of Prototypes – Start: Mid-1940's

As pointed out by Langlois and Mowery (1996), there was no real differentiation between hardware and software in that period, and computers were "unique" products, built for a unique project. They were computing tools, or research tools, for research centers (often military in nature, like H-bomb research centers). Each project allowed producers and users to negotiate the characteristics of the machine to be built. Also, the software part was not seen as an independent source of revenue by firms.

Production is Research

Thus, computer and software development were a research activity, conducted by high-skilled users, or Von Hippel (VH) users, in reference to Von Hippel's (1988) user who has the competences to innovate, and being the one who knows best his needs, is the best to do so (Dréan, 1996; Genthon, 1995).

Research is Cooperation

In that non-profit, research environment, we think that cooperation was rather natural, allowing firms to decrease their research costs and better answer to users' requirements. But this cooperation was mainly bilateral cooperation, between the constructor and the user. There was no network to exchange punch cards.

Period 2: Industrialization – Start: Early 1960's

Thanks to technological progress (miniaturization of transistors, compilers, and operating systems), the scope of use extended in two directions in that period: the reduction in size and in the price of computers. This raised the number of organizations that were able to afford a computer.

According to Genthon (1996), the main evolution characterizing the period was that the same program could be implemented in different computers (from the same family), allowing the program to evolve, to grow in size, and to serve a growing number of users. The computer had become a tool for centralized processing of information for organizations (statistics, payment of salaries, etc.).

The Emergence of a Software Industry

In this period, some pieces of software became strategic for producers, especially the operating

system, which was the element allowing them to control the client. In fact, as a program was developed for and worked with one single operating system, it became difficult for a client to break the commercial relation, once initiated, with a producer.

In "exchange," this client no longer even needed to understand the hardware part of the machine and could clearly (increasingly, throughout the period) evaluate the cost of its investments in the software part. This client, increasingly companies, was also more and more reluctant to publish in-house developed programs, for competitive reasons, and because most of the time these programs were so specific that few contributions could be expected.

Increased Cooperation, but for R&D Only

So we can say that the cooperative and open source development of software, and especially of innovative software was very strong in universities (it was during this period that Unix BSD, TCP/IP Internet protocol, etc., were developed), but also in some private research centers, like the Bell Labs (which actually invented the Unix operating system and licensed it very liberally). But this diffusion did not extend beyond the area which Dasgupta and David (1994) called "open science."

Period 3: Specialization – Start: Late 1970's

With the arrival of the micro-processor, the scope of use extended again in two directions: increase in power, and reduction in size and price of low-end computers. The dominant technological concept of this period was that the same program can be packaged and distributed to different persons or organizations, in the same way as for other tangible goods.

The third period was that of personal but professional information processing. As explained by Mowery (1996), this period was dominated by economy of scope thanks to the distribution of standardized computers (PC), but principally because of the development of standardized programs.

Research and Innovation are Strategic Assets to be Valorized

The willingness to close software production and to sell it as product was reinforced, in industry as well as in universities:

- In industry, thanks to the adoption of copyright protection, allowing the closure of the source code, but also because of the growing demand for standard programs, as already explained, and the decreasing skills of PC users; they were unable to develop or to modify their programs, nor to be innovators, and thus unable to cooperate with the producers (Jullien & Zimmermann, 2006); and
- In universities, because of the will at the beginning of the 1980's, due to the feeling of economic and industrial decline in the U.S., to better exploit scientific production, reinforcing its legal protection (see Coriat & Orsi (2002) for a complete analysis of intellectual property (IP) evolution during that period); Coriat and Orsi (2002) have shown that the consequence has been a segmentation of production, and a decrease in the exchange of IP, because of the increased fee to be paid for access to this IP, but also because of the transaction costs that this system has created.

Consequence: Decrease of Cooperation?

Cooperation decreased at the institutional level, but remained vivid and growing between researchers and VH users (especially those using Unix workstations) via mailing lists, or "news" services (Mowery, 1996). We could say that, alongside these increasingly-closing practices, this period (especially the 1980's) led to the structure of cooperative practice, and also its ideology, when Richard Stallman created the GNU project (1983) and the Free Software Foundation to support it, and Linus Torvalds started his Linux project (1991).

The point is thus not to understand why cooperation has come back, as actually it has always existed, but why it has returned to the forefront, and especially to the industrial agenda.

THE FLOSS MOVEMENT AS A NEW FRAMEWORK TO ORGANIZE (INDUSTRIAL) COOPERATION

The creation of the FLOSS movement, in opposition to the closing trend, has been advocated by Stallman (1998). We understand his arguments as the response of a profession (software developers) to an institutional trend (the closure of the sources) opposed to their culture, practices, and productivity. They organized themselves, creating a structured organization of cooperative production. With the diffusion of the Internet, FLOSS products spread, making firms involved in FLOSS production.

Developers' Motivations for a FLOSS Organization

As explained in Lakhani and von Hippel (2003), Lakhani and Wolf (2005), Demazière et al. (2004), individual motivations for initiating a Free software project or for collaborating in an existing project are numerous:

- For high-skilled developers, it sometimes costs less to develop a program from scratch than to use one which does not exactly meet its needs. Once developed, a free publication allows a quicker diffusion and, thus, quicker feedback, which is very useful to track bugs and to improve functionalities; and
- Following the same principle, which guided Stallman's initiative, such developers find it very interesting to use FLOSS products that they can adapt to their needs. The return of the bug found, the correction of those bugs, or the modifications, once done, is relatively inexpensive. And as for freeing a program, the developer knows that this problem solution or program modification will be integrated and maintained in the next versions.

It is clear that the social capital that a hacker can earn participating in a FLOSS project (the "carrier concerns"), pointed out by Lerner and Tirole (2002), is also a factor for keeping these persons motivated to do this task. However, this does not seem to have been anticipated by the developers who were interviewed (Demazière et al., 2004; Lakhani & Wolf, 2005).

In addition to this, the movement appears in an historical context:

- Computer science courses were (and are) partly based on the principle that it is more efficient to reuse what exists than to redevelop from scratch. Raymond (2000) has "codified" it as the hacker philosophy. Mainly concentrated in public and private research centers, this professional culture spread at the same time that students in computer science were being hired by firms;
- A personality, Richard Stallman, has made the difference between a vague feeling of

resentment towards the closure of programs in the IT professional community and the construction of a coordinated "riposte." His convictions in this regard have led him to give up his professional situation to defend them, by creating the Free Software Foundation (FSF). And his charisma allows him to be respectfully heard and followed by his "co-developers"; and

- New technical tools, especially the Internet services (and, before that, the ARPANET/ Usenet), have made this campaign possible and have largely facilitated the diffusion of FSF production and of Stallman's arguments against the closure of sources.

The consequence of this initiative and of the use of the Internet network has been the creation of a structured, organized way of producing software cooperatively.

The Creation of a Structured Organization for Cooperative Production

These voluntary contributions are organized and coordinated as Raymond (1999) advocated. Some in-depth studies of such communities have been conducted, such as the one by Mocus et al. (2000) on the Apache development organization, confirming this point.

Kogut and Metiu (2001) show that each project is led by a core group of developers (the "kernel") which develop the majority of the source code. A larger group follows the development, sometimes reports bugs, and proposes corrections ("patches") or new developments. And an even larger group just uses the program and sometime posts some questions on the use of this program on user mailing lists. If, in details, the organizations differ from one project to another, there are always mechanisms to select the contributions and the questions, so the main developers should not be inundated by peripheral problems. If there were not such mechanisms, the risk is that the most productive developers would progressively dedicate more and more time to addressing basic problems, thus losing interest in the development or even withdrawing from it.

Another very important characteristic is that big projects are split in coordinated, small, easier-to-manage subprojects. Baldwin and Clark (2003) show how this modularity reinforces the will to participate as developers can concentrate on the part/functionalities which interest them the most.

As far as FLOSS production was limited to the community of developers, apart from the commercial sphere, it only met a very small part of the users' needs and had no actual economic significance. Actually, this was close to the cooperation existing in the research/Unix world that we presented in the Increased Cooperation subsection. Things changed with the diffusion of the Internet and the consequent evolution of demand.

According to Porterfield (1999), the Internet and FLOSS are closely linked; the diffusion of the Internet, in non-U.S. research centers first and to the public later, has increased the number of users and developers of Free software, as the Internet tools are free software programs. This diffusion explains the diffusion of FLOSS products within firms, and the involvement of producers in FLOSS development.

When Cooperation Comes Back to Firms

At the beginning of the diffusion of the Internet within organizations (firms, administrations), in the early 1990's, servers were installed by engineers who had discovered these tools when they were students at a university. Additionally, as they often did not have any budgets, they installed what they knew at the lowest cost: "free" software. We can consider that the base installed in universities has initiated the "increasing return to adoption".

Actually, the first "FLOSS" firms entered the market selling Internet applications (Operating System, Web server, e-mail servers), like RedHat in 1994, Sendmail Inc. in 1998, and so forth, and to serve VH users (for instance, RedHat's Linux distributions on CdRom were interesting at that period because of low speed connectivity).

Today, since the announcement made by IBM in 2001 of investing $1 billion in Linux development, free software has been adopted in many commercial offers (Novell, buying Ximian and SuSE, Sun open-sourcing its operating system, IBM open-sourcing its development tool software Eclipse, even Microsoft[1]). Lakhani and Wolf (2005) notice that "a majority of (... the) respondents (of their survey) are skilled and experienced professionals working in IT-related jobs, with approximately 40 percent being paid to participate in the FLOSS project."

This can be explained by the evolutions in demand induced by this diffusion.

Following Steinmueller (1996), we see the generalization of computer networking, both inside and outside organizations, as the main technical evolution in information technology of today, in conjunction with miniaturization which has allowed the appearance of a new range of "nomad" products like Personal Digital Assistants (PDA, such as Psion and Palm), mobile phones, and music players. Network communications and exchanges between heterogeneous products and systems are nowadays crucial and require appropriate standards. Network externalities are becoming the dominant type of increasing return to adoption. According to Zimmermann (1999), to this aim, open solutions are probably the best guarantee for users and producers for product reliability throughout time and the successive releases of the products.

A second aspect stems from the wide diversity of users and users' needs that require software programs (and more particularly, software packages) to be adapted to the needs and skills of every individual without losing the economies of scale. Zimmermann (1998) explains that what characterizes this evolution of demand and the technological evolution of software is the increasing interdependence between software programs built from basic components and modules that have to be more and more reused, thus becoming increasingly refined and specialized. Furthermore, as pointed out by Horn (2004), the related demand for software customization generates a renewed services activity for the adaptation of standard-component software programs.

The split of the FLOSS project into different subprojects makes them quite modular and easier to adapt than monolithic programs, generating commercial offers from new entrants (like RedHat), especially toward those who do not have the skills to adapt the programs by themselves. In return, these adoptions reinforced the product, accelerating the increasing return to adoption phenomenon.

Considering that, incumbent producers have adapted their strategies, integrating the use of FLOSS products when necessary:

- Some of these programs, like Apache, were dominant in the Internet market. It was rational and less expensive to adopt the dominant design. These firms used the standard and contributed to it to make sure that the programs that they produced would certainly be taken into account by the standard; and
- In some markets, challengers saw opportunities to sponsor a standard in competition with the dominating standard. This is the case, for instance, in operating systems: Some competitors (IBM, HP, Novell now), of SUN in the Unix market, or of Microsoft in the operating systems for the servers' market, had strong incentives to support Linux to create a competitor to these firms at a very low cost. Being GPL protected, this program could not be appropriated by a single firm, which would mean coming back to the situation that they resisted by using Linux.

We think that FLOSS and especially GPL-protected software, when standardization effects are important, appears to be a way to solve the "wedding-game" situation: Actors want to impose their view, but less than to see a competitor imposing its own. So when you are not the leader, it can be interesting to favor an open, non-"privatizable" solution. Such "open" organization allows such creation of "public industrial goods" (Romer, 1993) and explain the renewal of cooperation in the software industry.

CONCLUSION

So, we could say that a series of convergent interests in the industrial domain has relayed the professionals' initiative to "revive" cooperation in software production, made easier at the beginning of the 1990's by the diffusion of a new information exchange network (the Internet).

This leads to two questions:

- Is that system transposable to another knowledge or information production industry, such as biotechnologies, increasingly based on database analysis and bio-computing, or hardware design, which uses a computer language, VHDL?

This could be the case if users and producers could find more efficient, less costly ways to share a part of their intellectual property, and if a structure, like FSF, could help to organize this cooperation.

- Are we witnessing the emergence of a new period for software ecology, with new technology, new dominant use (the network), and a new system of production (FLOSS-based production)?

This would be the case if the FLOSS production succeeds in matching the interests of computer professionals, users-innovators, and firms. This means succeeding in constructing a model, granting that these actors should contribute in the long run, and not just free-ride using the product. Developing Ousterhout's (1999), Bessen's (2002), Dahlander's (2004), Feller, Finnegan, and Hayes's (2005), Dahlander and Wallin's (2006), and Jullien and Zimmermann's (2006) works, there is a growing need for studies on business models assuring returns for firms and incentives to contribute, but also for studies on the impact of corporate involvement and agenda on the stability of the open cooperative organization of production.

REFERENCES

Arthur, W. B. (1988). Self-reinforcing mechanisms in economics. In P. W. Anderson, K. J. Arrow, & D. Pines (Eds.), *The economy as an evolving complex system: SFI studies in the sciences of complexity*. Redwood City, CA: Addison-Wesley Publishing Company.

Baldwin, C. Y., & Clark, K. B. (2003). The architecture of cooperation: How code architecture mitigates free riding in the open source development model. *Harvard Business School*. Retrieved from http://opensource.mit.edu/papers/baldwin-clark.pdf

Bessen, J. (2002). Open source software: Free provision of complex public goods. *Research on Innovation*. Retrieved from http://www.researchoninnovation.org/online.htm#oss

Clément-Fontaine, M. (2002). Free licenses: A juridical analysis. In N. Jullien, M. Clément-Fontaine, & J. –M. Dalle (Eds.), *New economic models, new software industry economy* (Tech. Rep. RNTL). French National Network for Software Technologies Project. Retrieved from http://www.marsouin.org/IMG/pdf/fichier_rapporte-3.pdf

Coriat, B., & Orsi, F. (2002, December). Establishing a new intellectual property rights regime in

the United States: Origins, content, and problems. *Research Policy, 31*(7-8).

Dahlander, L. (2004). *Appropriating returns from open innovation processes: A multiple case study of small firms in open source software*. School of Technology Management and Economics, Chalmers University of Technology. Retrieved from http://opensource.mit.edu/papers/dahlander.pdf

Dahlander, L., & Wallin, M. W. (2006). A man on the inside: Unlocking communities as complementary assets. *Research Policy, 35*, 1243-1259.

Dasgupta, P., & David, P. (1994). Toward a new economics of science. *Research Policy, 23*, 483-521.

Dréan, G. (1996). *L'industrie informatique, structure, économie, perspectives.* Paris: Masson.

Feller, J., Finnegan, P., & Hayes, J. (2005). Business models of software development projects that contributed their efforts to the Libre software community. *Calibre Report*. Retrieved from http://www.calibre.ie/deliverables/D3.2.pdf

Genthon, C. (1995). *Croissance et crise de l'industrie informatique mondiale*. Paris: Syros.

Horn, F. (2004). *L'économie des logiciels. repères, la Découverte*.

Jullien, N., & Zimmermann, J. -B. (2006). Free/libre/open source software (FLOSS): Lessons for intellectual property rights management in a knowledge-based economy. *M@rsouin Working Paper n°8-2006*. Retrieved from http://www.marsouin.org/IMG/pdf/Jullien-Zimmermann_8-2006.pdf

Kogut, B., & Metiu, A. (2001). Open-source software development and distributed innovation. *Oxford Review of Economic Policy, 17*(2), 248-264.

Lakhani, K., & von Hippel, E. (2003). How open source software works: Free user to user assistance. *Research Policy, 32*, 923-943.

Lakhani, K., & Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller et al. (Eds.), *Perspectives on free and open source software*. MIT Press.

Langlois, R. N., & Mowery, D. C. (1996). The federal government rôle in the development of the U. S. software industry. In D. C. Mowery (Ed.), *The international computer software industry: A comparative study of industry evolution and structure*. Oxford University Press.

Mowery, D. C. (Ed.). (1996). *The international computer software industry: A comparative study of industry evolution and structure*. Oxford University Press.

Ousterhout, J. (1999, April). Free software needs profit. *Communications of the ACM, 42*(4), 44-45.

Porterfield, K. (1999). Retrieved from http://www.netaction.org/articles/freesoft.html

Raymond, E. S. (1999). *The cathedral & the bazaar: Musing on Linux and open source by accidental revolutionary*. Sebastopol, CA: O'Reilly.

Raymond, E. S. (2000). *A brief history of hackerdom*. Retrieved from http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-history/

Richardson, G. B. (1997). Economic analysis, public policy, and the software industry. In E. Elgar (Ed.), *The economics of imperfect knowledge - Collected papers of G. B. Richardson: Vol. 97-4*. DRUID Working Paper.

Romer, P. (1993). The economics of new ideas and new goods. *Annual Conference on Development Economics, 1992, World Bank, Washington, DC*.

Stallman, R. (1998). *The GNU Project*. Retrieved from http://www.gnu.org/gnu/thegnuproject.html

Steinmueller, W. E. (1996). The U.S. software industry: An analysis and interpretive history. In D. C. Mowery (Ed.), *The international computer software industry: A comparative study of industry evolution and structure*. Oxford University Press.

Zimmermann, J. –B. (1998). Un régime de droit d'auteur: la propriété intellectuelle du logiciel. *Réseaux, 88-89*, 91-106.

Zimmermann, J. –B. (1999). Logiciel et propriété intellectuelle: du copyright au copyleft. *Terminal, 80/81*, 95-116. Special Issue, Le logiciel libre.

KEY TERMS

The ARPANET: "The Advanced Research Projects Agency NETwork developed by ARPA of the United States Department of Defense (DoD) was the world's first operational packet switching network, and the predecessor of the global Internet" (extract from Wikipedia article). It has been designed to connect the U.S. universities working with the DoD to facilitate cooperation.

Free/Libre Open Source software (FLOSS): This is software for which the licensee can get the source code, and is allowed to modify this code and to redistribute the software and the modifications. Many terms are used: *free*, referring to the freedom to use (not to "free of charge"), *libre*, which is the French translation of Free/freedom, and which is preferred by some writers to avoid the ambiguous reference to free of charge, and *open source*, which focuses more on the access to the sources than on the freedom to redistribute. In practice, the differences are not great, and more and more scholars are choosing the term FLOSS to name this whole movement.

Free Software Foundation: "Free software is a matter of liberty, not price. The Free Software Foundation (FSF), established in 1985, is dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs. The FSF promotes the development and use of free software, particularly the GNU operating system, used widely in its GNU/Linux variant" (presentation of the Foundation, http://www.fsf.org)

The GNU project: "The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. Variants of the GNU operating system, which use the kernel called Linux, are now widely used [...] (GNU/Linux systems). GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced *guh-noo*, approximately like *canoe*" (definition by the FSF, http://www.gnu.org/)

The GPL (General Public Licence): The best-known and the most-used FLOSS license; according to the FSF, "the core legal mechanism of the GNU GPL is that of "copyleft," which requires modified versions of GPL'd software to be GPL'd themselves". For an analysis of FLOSS licenses, see Clément-Fontaine (2002).

Hacker: In this text, this term is used in its original acceptation, i.e., a highly-skilled developer.

Increasing return to adoption, defined by Arthur (1988): This means that the more a product is adopted, the more new adopters have an incentive to adopt this product. Arthur (1988) distinguishes five type of increasing returns:

- **Learning effect**: Investment in time, money, and so forth, to learn to use a program, as well as a programming language, makes it harder to switch to another offer;
- **Network externalities:** The choices of the people you exchange with have an impact on the evaluation you make for the quality of a good. For instance, even if a particular text editor is not the one which is most appropriate to your document creation needs, you may

choose it because everybody you exchange with sends you text in that format, and so you need this editor to read the texts;

- **Economy of scale:** Because the production of computer parts involves substantial fixed costs, the average cost per unit decreases when production increases. This is especially the case for software where there are almost only fixed costs (this is a consequence of the fact that it presents the characteristics of a public good);
- **Increasing return to information:** One speaks more of a technology since it is widely distributed; and
- **Technological interrelations:** A piece of software does not work alone, but with other pieces of software. What makes the "value" of an operating system is the number of programs available for this system. And the greater the number of people who choose an operating system, the wider the range of software programs for this very system, and vice versa.

Public Good: This is good which is:

- Non-rivalrous, meaning that it does not exhibit scarcity, and that once it has been produced, everyone can benefit from it; and
- Non-excludable, meaning that once it has been created, it is impossible to prevent people from gaining access to the good. (definition taken in http://www.investordictionary.com/definition/public+good.aspx)

The Usenet (USEr NETwork): "It is a global, distributed Internet discussion system that evolved from a general-purpose UUCP network of the same name. It was conceived by Duke University graduate students Tom Truscott and Jim Ellis in 1979" (extract from Wikipedia article).

This work was previously published in Encyclopedia of Multimedia Technology and Networking, Second Edition, edited by M. Pagani, pp. 605-612, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.2
Free Software Philosophy and Open Source

Niklas Vainio
University of Tampere, Finland

Tere Vadén
University of Tampere, Finland

ABSTRACT

This chapter introduces and explains some of the most relevant features of the free software philosophy formulated by Richard M. Stallman in the 1980s. The free software philosophy and the free software movement built on it historically preceded the open source movement by a decade and provided some of the key technological, legal and ideological foundations of the open source movement. Thus, in order to study the ideology of open source and its differences with regard to other modes of software production, it is important to understand the reasoning and the presuppositions included in Stallman's free software philosophy.

INTRODUCTION

The free software (FS) movement is the key predecessor of the open source (OS) community. The FS movement, in turn, is based on arguments developed by Richard M. Stallman. In crucial ways, Stallman's social philosophy creates the background for the co-operation, co-existence and differences between the two communities. Stallman started the FS movement and the GNU project prompted by his experiences of the early hacker culture and subsequent events at the MIT artificial intelligence lab in the 1980s. The project was founded on a philosophy of software freedom, and the related views on copyright or the concept of *copyleft*. After the creation of the open source movement in 1998, debates between the two movements have erupted at regular intervals. These

debates are grounded in the different ideological perspectives and sociopsychological motivations of the movements. The FS movement has laid technological, legal and ideological cornerstones that still exist as part of the open source movement.

THE SOCIOHISTORICAL BACKGROUND OF THE FREE SOFTWARE PHILOSOPHY

The first computer systems were built in the 1940s and 1950s mainly for military and scientific purposes. One of the earliest research institutes to use and study computers was the Massachusetts Institute of Technology (MIT). The artificial intelligence (AI) lab at MIT was founded in 1958 and became one of the birthplaces of computer science and computer culture.

In *Hackers* (1984), Steven Levy describes the subculture around the AI lab computers in the 1960s. Young male electronics hobbyists devoted their time to programming and studying these machines. They called themselves *hackers*, a word denoting a person who enjoys exploring computer systems, being in control of the systems, and facing the challenges they present. For a hacker, a computer is not just a tool, it is also an end in itself. The computer is something to be respected and programming has an aesthetics of its own (Hafner & Lyon, 1996; Levy, 1984; Turkle, 1982).

A subculture was created among the MIT hackers with traditions and social norms of its own. Important values for the community were freedom, intelligence, technical skills, and interest in the possibilities of computers while bureaucracy, secrecy, and lack of mathematical skills were looked down on. The six rules of this *hacker ethic* as later codified by Levy were:

1. *Access to computers—and anything which might teach you something about the way the world works—should be unlimited and total. Always yield to the hands-on imperative!*
2. *All information should be free.*
3. *Mistrust authority—promote decentralization.*
4. *Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position.*
5. *You can create art and beauty on a computer.*
6. *Computers can change your life for the better.* (Levy, 1984, pp. 40-45)[1]

Computer programs were treated like any information created by the scientific community: Software was free for everyone to use, study, and enhance. Building on programs created by other programmers was not only allowed, but encouraged. On one hand, nobody owned the programs, and on the other, they were common property of the community.

In the early 1980s, a conflict arose in the AI lab when some of the hackers formed a company called Symbolics to sell computers based on technology originally developed in the lab. Symbolics hired most of the hackers, leaving the lab empty. This, together with the fact that the software on Symbolics machines was considered a trade secret, caused a crisis. The community and its way of life had been destroyed and Stallman later described himself as "the last survivor of a dead culture" (Levy, 1984, p. 427; see also Williams, 2002).

Stallman saw an ethical problem in the growing trend of treating software in terms of property. In the AI lab, there was a strong spirit of co-operation and sharing, making the code, in a way, a medium for social interaction. Thus restrictions in the access to code were also limitations on how people could help each other.

In 1984, Stallman published *The GNU Manifesto* announcing his intention to develop a freely available implementation of the Unix operating system. He explained his reasons in a section titled *Why I Must Write GNU*:

I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement…So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. (Stallman, 2002d, p. 32)

The project gained interest and Stallman started receiving code contributions from developers. During the 1980s, major components of an operating system were developed, including a system library, shell, C compiler, and a text editor. However, a core component, the kernel, was still missing until Linus Torvalds began to work on the Linux kernel in 1991. During the 1990s, free software systems based on the Linux kernel gained in popularity, media hype, and venture capital investments.

STALLMAN'S ARGUMENTS IN *THE GNU MANIFESTO* AND THE *FREE SOFTWARE DEFINITION*

Stallman's main argument in *The GNU Manifesto* (1984) is the "golden rule" quoted previously: A useful program should be shared with others who need it. Stallman started GNU in order to "give it away free to everyone who can use it" (Stallman, 2002d, p. 31) in the spirit of co-operation, sharing and solidarity. He criticizes proprietary software sellers for wanting to "divide the users and conquer them" (Stallman, 2002d, p. 32). Stallman's intention here is not anti-capitalist or anti-business. He gives suggestions on how software businesses can operate with free software. The fundamental ethical problem Stallman sees in proprietary software is the effect it has on community and co-operation. For Stallman, himself a master programmer, the "fundamental act of friendship among programmers is the sharing of programs" (Stallman, 2002d, p. 32). Restrictions on sharing would require programmers to "feel in conflict" with other programmers rather than feel as "comrades" (Stallman, 2002d, pp. 32-33).

Stallman suggests that software businesses and users could change the way they produce and use software. Instead of selling and buying software like any other commodity, it could be produced in co-operation between users and companies. Although the software would be free, users would need support, modifications and other related services which companies could sell. Stallman argues this would increase productivity by reducing wasteful duplication of programming work. Also it would make operating systems a shared resource for all businesses. If the business model of a company is not selling software, this would benefit the company. Being able to study source code and copying parts of it would increase the productivity of the programmer.

An important goal in the manifesto is increasing the users' independence from software sellers. When software is free, users are no longer at the mercy of one programmer. Because anyone can modify a free program, a business can hire anyone to fix the problem. There can be multiple service companies to choose from.

For Stallman, the main reason for rejecting software ownership is good civil spirit, but he also argues against the concept of copyright and authorship: "Control over the use of one's ideas' really constitutes control over other people's lives; and it is usually to make their lives more difficult," Stallman (2002d, p. 37) notes. He denies the idea of copyright as a natural, intrinsic right and reminds us that the copyright system was created to encourage literary authorship at a time when a printing press was needed to make copies of a book. At the time, copyright restrictions did little harm, because so few could invest in the equipment required to make a copy. Today, when copies

of digital works can be made at practically zero cost, copyright restrictions cause harm because they put limits on the way the works can benefit society. Stallman (2002d, p. 37) notes that copyright licensing is an easy way to make money but is "harming society as a whole both materially and spiritually." He maintains that even if there was no copyright, creative works would be created because people write books and computer programs on other grounds: fame, self-realization, and the joy of being creative.

Depending on the context, four different meanings of the term *community* can be found in Stallman's argument. The first one is that of a *hacker community* like the one at MIT's AI lab. The second is the *computer-using community* interested in business, wasted resources, and independence from software sellers. The third community is the *society* that will benefit from co-operation and face costs from complicated copyright and licensing mechanisms and enforcement. The fourth level of community Stallman mentions is *humanity*. He argues that because of the copyright restrictions on computer programs, the "amount of wealth that humanity derives" is reduced (Stallman, 2002d, p. 36). In these four meanings of the term, we can see community grow from a small group of friends to an interest group, then to society and finally to humanity as a whole. As the communities grow in size, the temporal perspective is expanded: for hacker friends, the benefits are direct and immediate whereas in the case of humanity change may require decades.

In *The GNU Manifesto*, Stallman mentions that "everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed" (Stallman, 2002d, p. 32). In *Free Software Definition* (Stallman, 2002a), he lists the *four freedoms* which a piece of software must meet in order to be free software. The freedoms are:

- **Freedom 0:** The freedom to run the program, for any purpose
- **Freedom 1:** The freedom to study how the program works, and adapt it to your needs; access to the source code is a precondition for this
- **Freedom 2:** The freedom to redistribute copies so you can help your neighbor
- **Freedom 3:** The freedom to improve the program, and release your improvements to the public, so that the whole community benefits; access to the source code is a precondition for this (Stallman, 2002a, p. 41)

Freedom of software is defined by referring to the rights of the computer user, who may run the program for any purpose, good or evil, study and adapt the software, and distribute copies of the program, modified or original. It should be noted that the definition assumes sharing is always beneficial and desired. It does not matter if the neighbor or the community has any use for the software or the skills to use it.

For a piece of software to be free, it would not be enough to abolish the copyright system. Because a user needs the source code in order to effectively exercise freedom 3, the author must actively promote software freedom by releasing the source code. Therefore, a co-operative community is already needed for software freedom.

Stallman makes an important distinction between *free as in free speech* and *free as in zero price*. The concept of free software is not against selling software, it is against restrictions put on the users. Free software can be sold but the seller may not forbid the users to share or modify it.

COPYLEFT: THE GPL AS LEGAL AND SOCIAL DEVICE

Because Stallman was the copyright holder of the GNU programs that he wrote, he could have handed the programs to the public domain. Thus

the programs would have been free. However, releasing the programs to the public domain would have meant that people would have been able to distribute the programs in ways which would have restricted the freedom of users, for instance, by distributing them without the source code. A free program would have become non-free. Stallman wanted the distribution of his programs or any other free software to stay free forever, and together with Free Software Foundation (FSF) legal counsel Eben Moglen, they devised the GNU General Public License (GPL) for this purpose (Stallman, 1989; Stallman, 2002b). The main idea of the GPL is that anyone is free to use, modify and redistribute a GPLed program on the condition that the same freedom of use, modification, and redistribution is also given to the modified and redistributed program. The easiest way to fulfill the condition is to release the redistributed and modified program under the GPL. The GPL is in this sense "viral": a GPLed program can be unified with other code only if the added code is compatible with the GPL. The purpose of the GPL is to keep free software free and to stop it ever becoming a part of proprietary software (Stallman, 2002a, pp. 89-90; Stallman, 2002c, pp. 20-21). The GPL is called a *copyleft* license, because in a sense it turns around the copyright by giving the user, not only the author, the freedom to use and to continue to build on the copylefted work. In this sense, copyright law and the GPL license built on it are the artifices that make the free software movement possible. There is some irony to the fact that the movement in this sense needs the copyright law in order to function. This is also the reason why it is not correct to describe the movement as being against copyright. Consequently, the GPL has to function well. The original GPL version 1 has been modified into version 2, under which, for instance, the Linux kernel is released. Currently, in 2006, a new version, GPLv3, is being prepared by Stallman and the FSF. The somewhat unorthodox twist that GPL gives to copyright law has sometimes aroused suspicion over whether the GPL is a valid and enforceable license. As Moglen (2001) notes, most often GPL violations are settled without much publicity in negotiations between the FSF and the violator. As the FSF seeks only the freedom of software, a violator can easily rectify the situation by starting to comply with the GPL. It is sometimes argued that the fact that code under GPL can not lose the property of being free does not give the user maximum freedom with the code: the user is not permitted to "close" the code and release it under a proprietary software license. For instance, a typical Berkeley Software Distribution (BSD) license does not require that modifications or derivative works be free. Proponents of BSD see this as a good thing, maybe even as a benefit over the GPL, because the BSD license gives the developer more possibilities. However, for Stallman this is not desired, as closing the source tramples on the possible future uses of the code: "It is absurd to speak of the 'freedom to take away others' freedom'" (Stallman cited in Butler, 2005).

FREE SOFTWARE AS A POLITICAL PHILOSOPHY

As described above, Stallman's free software philosophy goes beyond the freedom and needs of an individual programmer. In Stallman's work, we find a political philosophy that has roots both in the liberalist and the communitarian traditions but accepts neither as such.

Stallman's ideas on user's freedom have roots in the liberalist political philosophy of Thomas Hobbes, John Locke, John Stuart Mill and others. In the *Second Treatise of Government* (1690), Locke argued that societies are built on a social contract in which people agree to give away some of their personal liberty to escape the cruel reality of the "state of nature" and receive protection for the fundamental rights which are life, liberty, and property. Locke's influence on political philosophy can be seen, for example, in the formulation of

the U.S. Declaration of Independence and in the Constitution.

Stallman describes his relation to the liberalist tradition as follows:

The philosophy of freedom that the United States is based on has been a major influence for me. I love what my country used to stand for. ... Science is also an important influence. Other campaigns for freedom, including the French and Russian revolutions, are also inspiring despite the ways they went astray. (Stallman, 2004)

The four freedoms of free software were named after the influential speech given by the U.S. President Franklin D. Roosevelt during the Second World War in 1941, called *The Four Freedoms* (Roosevelt, 1941).

For Locke, freedom means "to be free from restraint and violence from others" (Locke, 1690, para. 57). For Stallman, software freedom means the freedom to run, study, modify, and distribute the program Locke described the time before organized society as a natural state where everybody had complete freedom but had to live in constant danger. Stallman has described the American society as a "dog-eat-dog jungle" where antisocial behavior like competition, greed and exclusion is rewarded instead of co-operation (Levy, 1984, p. 416; Stallman, 2002e). Because sharing of software is forbidden, freedom is restricted in such a society.

The tension between individualism and communitarianism is constant in Stallman's philosophy. He started the GNU project because of his own moral dilemma, but he also argues for it on a collectivist basis. In the first announcement of the GNU project (Stallman, 1983), the perspective was individualist: "So that I can continue to use computers without violating my principles, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free." In *The GNU Manifesto* (Stallman, 2002d), the words "violating my principles" were replaced with the word "dishonor," indicating a move towards a more communal view. The tension also arises if we ask for what and for whom software freedom is intended. Isaiah Berlin (1969) has introduced a distinction between the notions of negative and positive freedom: negative freedom means freedom from obstacles and restrictions while positive freedom means control over one's life and positive opportunities to fulfill a goal. Both the liberalist tradition and Stallman mainly use the negative concept of freedom, but in his emphasis on community we can also see aspects of positive freedom.

Freedom 0, the freedom to run the program, is a pure example of the negative concept of freedom. The user has the right to use the software, whatever the purpose might be. Freedom 1 has two components: having permission to study the program and having the source code. In this sense freedom 0 is not only about absence of restraints, it is also about presence of the source code and in this sense a positive freedom. Likewise, freedom 2 is not only an individualist or negative freedom: the freedom to redistribute copies is necessary to help the neighbour. Freedom 3 to improve the program and release the improvements to the community is also of a positive nature: It is required to build a community.

For a programmer, freedom of software is a fundamental issue related to a way of life, to the identity of a hacker. Is the freedom relevant only for programmers? Bradley Kuhn and Richard Stallman reply:

We formulated our views by looking at what freedoms are necessary for a good way of life, and permit useful programs to foster a community of goodwill, cooperation, and collaboration. Our criteria for Free Software specify the freedoms that a program's users need so that they can cooperate in a community. We stand for freedom for programmers as well as for other users. Most of us are programmers, and we want freedom for

ourselves as well as for you. But each of us uses software written by others, and we want freedom when using that software, not just when using our own code. We stand for freedom for all users, whether they program often, occasionally, or not at all. (Kuhn & Stallman, 2001)

This freedom is for everyone, whether they need it, use it, or not, just like freedom of speech. But freedom of software is just a means to a more important end, which is a co-operative, free society. Stallman wants to contribute to a society that is built on solidarity and co-operation, not exclusion and greed. In a communitarian way, the argument sees morality and the good of the individual co-dependent on the good of the community.

POLITICAL MOVEMENT OR DEVELOPMENT MODEL? A COMPARISON OF FS AND OS IDEOLOGIES

One of the motivations for launching the Open Source Initiative (OSI) was the perception that the ideology and concepts used by the FS movement, in general, and Richard Stallman, in particular, were putting off potential collaborators, especially business partners. Eric S. Raymond explains his motivations as tactical, rather than principal:

The real disagreement between the OSI and the FSF, the real axis of discord between those who speak of "open source" and "free software," is not over principles. It's over tactics and rhetoric. The open source movement is largely composed not of people who reject Stallman's ideals, but rather of people who reject his rhetoric. (Raymond, 1999)

Thus, the aim of the term *open source* is to emphasize the practical benefits of the OS development model instead of the moral philosophy behind the free software ideal. For the actors in the OS movement, the creation of OS software is an utilitaristic venture of collaboration, based on individual needs. According to Eric S. Raymond, "Every good work of software starts by scratching a developer's personal itch" (Raymond, 1999). This is in clear contrast with the intentional, systematic, and collective effort described by Stallman: "essential pieces of GNU software were developed in order to have a complete free operating system. They come from a vision and a plan, not from impulse" (Stallman, 2002c, p. 24).

The main ideological shift was in the professed motivation for writing code. The software itself often stayed the same: by definition, free software is a subset of open source software. For the outside world this ideological shift may present itself as relatively minor, so that in the name of simplification a common name such as FOSS (free/open source software) or FLOSS (free/libre and open source software) is often used. Initially the two communities also overlapped to a large degree, but lately some polarization has been in evidence. For instance, in a recent survey a large majority of Eclipse developers reported that they identify with the OS movement, while a clear majority of Debian developers reported identification with the FS movement (see Mikkonen, Vainio, & Vadén, 2006). This development may be expected to continue, as companies are increasingly taking part and employing programmers in OS development.

A crucial difference between OS and FS has to do with the political economy of software production. However, this distinction is best described as the difference between *business friendly* open source and *ideological/political* free software, or *capitalist* open source and *communist* free software. These are not the correct levels of abstraction. For instance, sometimes the GPL license is more *business friendly* than a given non-GPL-compatible open source license. The fact that the OS community treats code as a *public good* might be perceived as odd in certain types of market economies, while in others such public goods are seen as necessary drivers

of capitalism. By making software a non-scarce resource, OS has an effect on where and how a revenue stream is created. However, this merely reorganizes production and labour, instead of changing their mode.

Schematically put, FS is a social movement, while OS is a method for developing software. Whatever the definitions of systems of economical production—such as capitalism, communism, market economy, and so on—may be, OS is non-committal with regard to the current issues of political economy, such as copyright, intellectual property rights and so on. Individual members of the OS community may or may not have strong views on the issues, but as a community OS is mostly interested in the benefits of openness as a development model. This attitude is well exemplified in the views expressed by Linus Torvalds: "I can't totally avoid all political issues, but I try my best to minimize them. When I do make a statement, I try to be fairly neutral. Again, that comes from me caring a lot more about the technology than about the politics, and that usually means that my opinions are colored mostly by what I think is the right thing to do technically rather than for some nebulous good" (quoted in Diamond, 2003). This pragmatic or "engineering" view on FOSS is intended to work better than ideological zealotry in advancing the quality and quantity of code. In contrast, in order to change a political system one needs a social movement. As noted previously, the FS movement is a social movement based on shared values. While these values are close to the loosely interconnected values of the anti-globalization movement (see Stallman, 2005, 2002f), they are not the defining values of socialist or communist parties or movements. For instance, the FS movement does not have a stand on class relations or on how to treat physical property, and so on. In this sense the FS movement as a social movement is a specialized, one-cause movement like many other post-modern social movements. Again, here lies a crucial distinction: the ethical principles of FS concern only information, and only information that is a tool for something. Typically, a socialist or communist set of values would emphasize the importance of material (not immaterial) things and their organization.

Ideologically proximate groups often behave in a hostile manner towards each other in order to distinguish themselves; the public controversies between the FS and OS communities are a good example. Extra heat is created by the different perspectives on the politics of freedom. The Torvaldsian view of "no politics" is tenable only under the precondition that engineering can be separated from politics and that focusing on the engineering part is a non-political act. Stallman, for one, has consistently rejected this precondition, and claims that the allegedly non-political focus on the engineering perspective is, indeed, a political act that threatens the vigilance needed for reaching freedom.

A good example of these controversies is the one over the name (Linux or GNU/Linux) of the best known FOSS operating system. Since the mid 1990s, Stallman and the FSF have suggested that developers use the name GNU/Linux, arguing that "calling the system GNU/Linux recognizes the role that our idealism played in building our community, and helps the public recognize the practical importance of these ideals" (FSF, 2001). However, true to their pragmatical bent, OS leaders such as Raymond and Torvalds have replied that the name Linux has already stuck, and changing it would create unnecessary inconvenience. Some of the distributions, such as Debian, have adopted the naming convention suggested by the FSF.

CONCLUSION: FS AS A HISTORICAL BACKDROP OF OS

The FS movement initiated by Stallman predates the OS movement by over a decade and the latter was explicitly formed as an offshoot of the former. Consequently, the definition of OS soft-

ware was developed in the context of an ongoing battle between the FS and proprietary software models. Arguments presented by Stallman in the early 1980s still form some of the most lucid and coherent positions on the social and political implications of software development. Most importantly, the polarization of the FOSS community into the FS and OS camps has been only partial. All of these facts point out how FS has acted as a necessary background for OS. This background can roughly be divided in technological, legal and ideological parts.

On the technological side, FS code often forms a basis and ancestry for OS projects. The formation of the operating system Linux or GNU/Linux is one of the examples where the functions of the FS movement form an essential cornerstone of existing OS software. Typically Linux distributions include major technological components (such as glibc [the GNU C Library], Coreutils, and gcc) from the GNU project.[2] It is uncontroverted that without the systematic and prolonged effort by the FSF the development and adoption of Linux (the operating system) would not have been as rapid or widespread as it has been. However, it is equally clear that several key OS projects, such as Apache or Eclipse, are not technologically dependent on GNU.

The legal cornerstone provided to the OS community by the FSF and Stallman is the GPL license, under which Linux (the kernel) and several other key OS projects were developed. The GPL is concurrently clearly the leading FOSS license, comprising over 50% of the code in projects maintained at SourceForge and of major GNU/Linux distributions (Wheeler, 2002). The GPL as a license and the ideal of freedom that it embodies are the legal lifeblood of both the FS and the OS communities, even though several other families of licenses are crucially important.

The ideological foundation provided by the FS movement is difficult to gauge quantitatively. Suffice it to say the OS movement is, according to its own self-image, a tactical offshoot of the FS movement. Many of the sociocultural arguments (openness for reliability, longevity of code, and user control) and ways of functioning (collaborative development based on the GPL) that the OS community uses were spearheaded by the FS community. Moreover, now that OS is moving outside its niche in software production and gaining ground as a *modus operandi* in other fields (such as open content, open medicine, open education, open data, and so on), the OS movement finds itself again in closer proximity to the ideals expressed by the FS movement. However, there are also trends that tend to emphasize the neutral, engineering point of view that created the need for the separation of OS from FS in the first place: as OS software becomes more commonplace and even omnipresent, the ideological underpinnings are often overlooked with or without purpose.

REFERENCES

Berlin, I. (1969). Two concepts of liberty. In I. Berlin, *Four essays on liberty*. Oxford, UK: Oxford University Press.

Butler, T. (2005, March 31). Stallman on the state of GNU/Linux. *Open for Business*. Retrieved February 20, 2006, from http://www.ofb.biz/modules.php?name=News&file=article&sid=353

Diamond, D. (2003, July 11). The Peacemaker: How Linus Torvalds, the man behind Linux, keeps the revolution from becoming a jihad. *Wired*. Retrieved February 20, 2006, from http://www.wired.com/wired/archive/11.07/40torvalds.html

FSF. (2001). *GNU/Linux FAQ*. Retrieved February 20, 2006, from http://www.gnu.org/gnu/gnu-linux-faq.html

Hafner, K., & Lyon, M. (1996). *Where wizards stay up late: The origins of the Internet*. New York: Touchstone.

Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. New York: Random House.

Kuhn, B., & Stallman, R. (2001). *Freedom or power?* Retrieved February 20, 2006, from http://www.gnu.org/philosophy/freedom-or-power.html

Levy, S. (1984). *Hackers: Heroes of the computer revolution*. London: Penguin.

Locke, J. (1690). *Second treatise of government*. Indianapolis: Hackett.

Mikkonen, T., Vainio, N., & Vadén, T. (2006). Survey on four OSS communities: Description, analysis and typology. In N. Helander & M. Mäntymäki (Eds.), *Empirical insights on open source business*. Tampere: Tampere University of Technology and University of Tampere. Retrieved June 27, 2006, from http://ossi.coss.fi/ossi/fileadmin/user_upload/Publications/Ossi_Report_0606.pdf

Moglen, E. (2001). *Enforcing the GNU GPL*. Retrieved February 20, 2006, from http://www.gnu.org/philosophy/enforcing-gpl.html

Raymond, E. S. (1999). *Shut up and show them the code*. Retrieved February 20, 2006, from http://www.catb.org/~esr/writings/shut-up-and-show-them.html

Raymond, E. S. (2003). *The jargon file*. Retrieved June 6, 2006, from http://www.catb.org/jargon/

Roosevelt, F. D. (1941). *The four freedoms*. Retrieved February 20, 2006, from http://www.libertynet.org/~edcivic/fdr.html (May 27, 2004)

Stallman, R. (1983, September 27). *New UNIX implementation. Post on the newsgroup net.unix-wizards*. Retrieved February 20, 2006, from http://groups.google.com/groups?selm=771%40mit-eddie.UUCP

Stallman, R. (1989). *GNU General Public License version 1*. Retrieved February 20, 2006, from http://www.gnu.org/copyleft/copying-1.0.html

Stallman, R. (2002a). Free software definition. In J. Gay (Ed.), *Free software, free society: Selected essays of Richard M. Stallman* (pp. 41-43). Boston: GNU Press.

Stallman, R. (2002b). GNU General Public License version 2. In J. Gay (Ed.), *Free software, free society: Selected essays of Richard M. Stallman* (pp. 195-202). Boston: GNU Press.

Stallman, R. (2002c). The GNU project. In J. Gay (Ed.), *Free software, free society: Selected essays of Richard M. Stallman* (pp. 15-30). Boston: GNU Press.

Stallman, R. (2002d). The GNU manifesto. In J. Gay (Ed.), *Free software, free society: Selected essays of Richard M. Stallman* (pp. 31-39). Boston: GNU Press.

Stallman, R. (2002e). Why software should be free. In J. Gay (Ed.), *Free software, free society: Selected essays of Richard M. Stallman* (pp. 119-132). Boston: GNU Press.

Stallman, R. (2002f). *The hacker community and ethics: An interview with Richard M. Stallman*. Retrieved February 20, 2006, from http://www.uta.fi/~fiteva/rms_int_en.html

Stallman, R. (2004, January 23). *A Q&A session with Richard M. Stallman*. Retrieved February 20, 2006, from http://puggy.symonds.net/~fsug-kochi/rms-interview.html

Stallman, R. (2005, December 18). Free software as a social movement. *ZNet*. Retrieved February 20, 2006, from http://www.zmag.org/content/showarticle.cfm?SectionID=13&ItemID=9350

Turkle, S. (1982). The subjective computer: A study in the psychology of personal computation. *Social Studies of Science, 12*(2), 173-205.

Wheeler, D. (2001). *More than a gigabuck. Estimating GNU/Linux's size*. Retrieved February 20, 2006, from http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html

Wheeler, D. (2002). *Make your open source software GPL-compatible: Or else*. Retrieved February 20, 2006, from http://www.dwheeler.com/essays/gpl-compatible.html

Williams, S. (2002). *Free as in freedom: Richard Stallman's crusade for free software*. Sebastopol, CA: O'Reilly.

KEY TERMS

Communitarianism: A philosophical view holding that the primary political goal is the good life of the community.

Copyleft: The practice of using copyright law in order to remove restrictions on the distribution of copies and modified versions of a work for others and require the same freedoms be preserved in modified versions.

Free Software (FS): Software that can be used, copied, studied, modified, and redistributed without restriction.

General Public License (GPL): A widely used free software license, originally written by Richard M. Stallman for the GNU project.

Hacker Community: A community of more or less likeminded computer enthusiasts that developed in the 1960s among programmers working on early computers in academic institutions, notably the Massachusetts Institute of Technology. Since then, the community has spread throughout the world with the help of personal computers and the Internet.

Liberalism: A philosophical view holding that the primary political goal is (individual) liberty.

ENDNOTES

1 For alternative formulations of the hacker ethos, see the entry "hacker ethic" in *The Jargon File*, edited by Raymond (2003) and *The Hacker Ethic* by Himanen (2001), who gives the concept a more abstract scope.

2 For a view of the complexity of a GNU/Linux distribution (see Wheeler, 2001).

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 1-11, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.3
Open Source Software Basics:
An Overview of a Revolutionary Research Context

Eirini Kalliamvakou
Athens University of Economics and Business, Greece

ABSTRACT

The open source software (OSS) development area of research presents a fresh and generous domain for analysis and study. In line with this, it is important to have a high-level understanding of the "open source phenomenon" before being able to delve deeper into specific niches of research. OSS presents a rich picture and because of that, both academics and practitioners have shown intense interest in producing high-quality literature. This chapter provides an initial understanding of what OSS is and how it has come to be the exciting research platform that it is today, attracting attention from various sources. In addition, we take an overview of the research streams that have formed in recent years, and the basic findings of attempts made to transfer lessons from OSS to other research areas.

OPEN SOURCE SOFTWARE AT A GLANCE

Open source software (OSS) has received growing attention in recent years from various perspectives. The thriving numbers behind OSS adoption and contribution have captured the attention of academic research that, in the past years, has been trying to decipher the phenomenon of OSS, its relation to already-conducted research, and its implications for new research opportunities.

OSS has a definition that focuses on specific characteristics that software has to serve in order to be labeled as "open source." The Open Source Initiative (OSI) is a nonprofit corporation dedicated to managing and promoting the OSS definition for the good of the community; thus, acting as the official organization behind OSS. Based on the OSS definition provided by OSI, any software that has the characteristics listed below is considered to be OSS, and vice versa:

- Free redistribution
- Access to source code
- Derived works allowed under the same license
- Integrity of the author's source code
- No discrimination against persons or groups
- No discrimination against fields of endeavor
- Distribution of license
- License must not be specific to a product
- License must not restrict other software
- License must be technology-neutral

The current OSS landscape presents a very interesting picture. Although the idea behind OSS dates back to the 1960s and the UNIX era in the 1980s, the official term of OSS was coined in 1998 and, at the same time, the OSI was created. Since then, the OSS movement has evolved at a very fast pace. Prime examples of successful OSS projects include operating systems (Linux, FreeBSD, OpenBSD, NetBSD), Web browsers (Firefox, Konqueror), graphical environments (KDE, Gnome), productivity applications (OpenOffice), programming languages and infrastructure (Apache, MySQL), and development tools (GNU toolchain, Eclipse). These widely accepted OSS endeavors show that, today, a wide range of OSS applications are available and they present a viable and robust alternative to proprietary software solutions.

In addition to the presence of prime examples in the OSS environment, the plethora of OSS projects is impressive. Project support sites are online environments that provide tools for listing and managing OSS projects while supplying information such as developer teams, maturity stage, latest versions, and so forth. Two of the biggest and most well-known support sites, *Sourceforge.net* and *Freshmeat*, have reported to have more than 125,000 and 40,000 listed projects in the summer of 2006, respectively. Although many of these projects are still in designing stages, these numbers reveal the dynamics behind OSS and its evolution/adoption. Furthermore, this striking progress of OSS provides an intricate motive for conducting research in such a context.

OPEN SOURCE SOFTWARE AS A RESEARCH CONTEXT

Open source software is developed in a way different than proprietary software. Development is done inside communities of developers that work on code for their personal satisfaction or need. However, lately, a trend has formed inside large companies, that pay their employees to contribute to OSS projects, using this as a platform that enables them to affect the introduction of new software features so that the final OSS product is better aligned with the company's interests and needs. An open question remains as to how expanded this trend currently is, and whether the "paid volunteers" are involved in the process of developing OSS software primarily out of their own satisfaction or are assigned to it exclusively by their employers.

Independent of this fact, the community-oriented development leads to the efficient production of high-quality software available for use by anyone interested. This is an important element of OSS, the prime motivation for developers is not to "make software" as requested by clients or employers, but mainly to satisfy their own software needs, which cannot be fulfilled by vendor-supplied software. After the software is prototyped, it can be made public for anyone who wishes to use, modify, and redistribute.

OSS communities have significant similarities with professional software engineering teams utilized by software houses. However, these two organizational structures also portray critical differences that show that they stand quite far apart. It is important to note that since not all companies organize their software development efforts in the same mode, and also, there is a wide variety

Table 1. Differences between OSS and proprietary software projects (Based on Crowston & Howison, 2005)

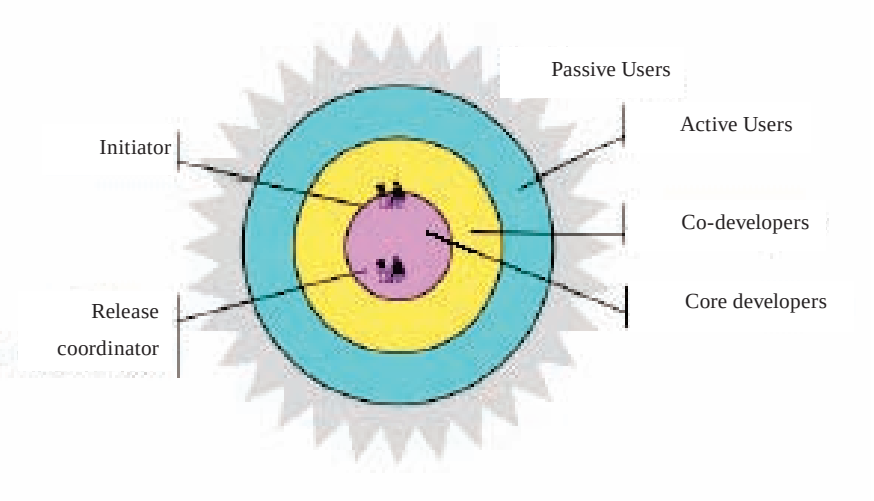
	OSS projects	**Proprietary software projects**
Release planning process	Own ideals about quality and features	Time-to-delivery pressure
Quality assurance	Developers do not have write access to the repository	Developers have write access to the repository
Leadership	Leaders are required to have proven competence via previous contributions	Project leadership is a hierarchical level where people are promoted by other than technical criteria
Tools and standards	Use of standardized tool chains (not modern modeling tools and techniques)	Each project may opt for another technology leading to different set of tools (code re-use between projects is limited)
Motivation of developers	Desire to learn and establish new skills (fun-to-code)	Typical task assignment by hierarchical superiors or salary incentives (not as efficient)
Roles of members	Members assume roles according to personal interests	Tasks are assigned

of organizational structures inside OSS projects, similarities and differences between them can be valid upon different occasions. The most basic differences are pictured in Table1.

Inside OSS communities, there are rules to be accepted and respected by both the developers and the users. Also, there is always some hierarchy of roles, although this hierarchy can take many forms, depending on the internal organization of each project and the profiles of its developers and owners. Through various case studies (Cox, 1998; Gacek et al., 2004; Moon & Sproull, 2000; Mockus, Fielding, & Herbsleb, et al., 2002; Moon & Sproull, 2000), a hypothesized model of OSS project structure is suggested. The model consists of concentric circles of persons that serve different roles within the project and have different responsibilities. At the center of this onion-like form, we find the team of core developers that contribute most of the code, and also have the role of overseeing the project's evolution and design. Core developers also review and check-in the patches submitted by co-developers that belong to the consecutive circle. In the next circle, we find active users of the project's product that do not write code but provide bug reports, testing of new releases, and also use-cases. Finally, in the outer circle (for which there is no specific border), we find the software's passive users that do not have an active presence in the project's mailing lists and forums. This onion-like structure is shown in Figure 1.

What is surprising and initially hard to explain in the case of OSS is *how it is possible to build software of top quality that, in many cases, outperforms the products of multimillion-dollar companies with a vast amount of resources.* The answer seems to lie beyond issues of money and resources and closer to self-organizing communities and networks between developers. The OSS phenomenon has brought forth a different perspective in the way software is created. New paradigms regarding technical, legal, team formation, and organization and knowledge dissemination issues have converted OSS development into an interesting and intriguing research topic.

Figure 1. The OSS onion-like model of organization (Crowston & Howison, 2005)



Although the majority of studies, at least at the beginning of the spread of OSS, were mostly concerned with the technical elements behind its success, some other aspects related to OSS soon presented interesting research questions. Hence, concepts like *community* and *social interaction* are currently under investigation and reveal their great importance when interpreting and understanding many questions.

Since OSS is presenting such an interesting research area with useful conclusions and implications also for other domains of research, it is important that a researcher has at least a brief overview of discussions and studies that have formed the OSS literature. In the next section, we will take a look of at a limited number of very influential studies that portray the multidisciplinary approach to OSS research and the intriguing results that it can provide. Through this approach, we will be able to gain a deeper understanding of the research context offered by OSS.

OSS LITERATURE

As it is expected, the core of researchers of open source software, at the beginning, mostly consisted of software engineers with the basic aim to understand the processes associated with OSS development and get a global picture of it. However, as it is evident now, along with the technical aspects of OSS, also social, economic, organizational, and other issues form its rich picture. This is why in the last few years the "OSS phenomenon" has attracted the attention of not only software engineers, but also researchers from many other disciplines and backgrounds.

OSS became a matter of study in the late 1990s, although it can be traced back to the 1980s and even earlier (Salus, 2005). The first studies came from individuals actively involved in OSS development communities (rather than academic researchers) who published results regarding the OSS development processes and outcomes as well as its economic implications for the software development industry/scenery in general. Also, popular voices at that time were those of OSS advocates that discussed ethical and philosophical aspects of OSS (Stallman, 1999), although published later, is a good example). A third group of references regarded the use of applications, with little attention to the development process itself. Academic papers related to OSS began to appear around 2000 in workshops and in conferences and

a little later in scientific journals and magazines. After extensive research on the literature on OSS development, we have found almost no significant references before 1998.

Early Works Try to Decipher the Development Process

The most widely-known, representative and influential text of that time is the well-known essay by Eric S. Raymond "The Cathedral and the Bazaar" (Raymond, 1998). Although the text is not supported by an empirical study of many projects (only the *Linux kernel* and *fetchmail* projects are discussed), "The Cathedral and the Bazaar" made an important contribution by establishing a metaphor. Traditional software development projects were modeled as cathedrals, where the development process is centralized and highly dependent on just a few persons whose roles are clearly defined. Release management follows a closed planning, user feedback is limited, and contributions by external developers are not fostered. In opposition to the cathedral we find the bazaar, where there is lack of planning (at least in the first stages of the project) and everything starts with the need of a developer and his/her ability to create software that meets that need. Developers publish often and users are regarded as co-developers under the bazaar model. Obviously, the bazaar model represents a more flexible and open way for the software development process that stands contrary to the cathedral model. It is important to note that the cathedral model should not be mistaken as representing only proprietary software; some OSS projects from the GNU project (such as the GNU Compiler Collection and the GNU Emacs editor) fall under the cathedral model of development although they are OSS.

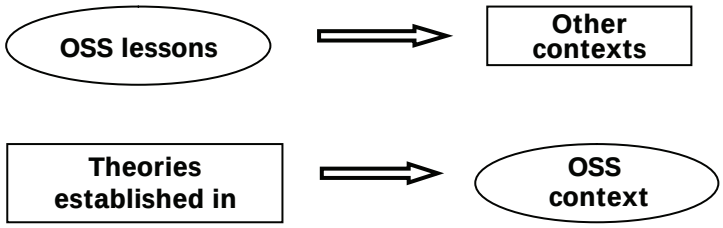
Almost immediately after "The Cathedral and the Bazaar" was released (first through the Internet, later published), other authors started either to build on top of it or severely criticize it. One well-known follow-up came from Bezroukov (Bezroukov, 1997), who made a point saying that the bazaar development model is not some revolutionary phenomenon, but just a different form of "scientific community." Bezroukov considered Raymond's ideas as too simple to match reality, and proposed other models based on academic research that could better explain the phenomenon.

As far as development models for OSS are concerned, Vixie compared the classical waterfall model with the processes used in the production of OSS (Vixie, 1999). According to Vixie, the waterfall model is composed of a set of a complete and logical sequence of steps aiming at obtaining a final product, while the OSS development model relies on the absence of formal methods. Such methods prove unsatisfactory for programmers who voluntarily devote time to program. This lack of formality, Vixie argues, is compensated for by user feedback and by the introduction of software engineering methods (developers define a methodology to face a specific problem for which they are experienced enough). Therefore, it is obvious through Vixie's argument that no ad hoc development model exists for OSS; the methods and practices evolve with the project and usually show a tendency to formalization of tasks.

Identifying Research Streams in OSS

Although, as it is evident above, the early works in OSS literature were concerned with the development model of OSS, this picture quickly changed. This coincided with, and was the result of, the rapid expansion of OSS and its successful use and adoption by a growing mass of interested parties. This success needed both to be explained and also to investigate whether it could prove useful for other research areas (related or not to software engineering that was the principal environment for the birth of OSS). Therefore, the extensive review of literature for OSS led us to discover and reveal two main research streams that confirm these inherent needs. These two research streams can be visualized in Figure 2.

Figure 2. Research streams in OSS literature



Applying OSS Research Conclusions to Other Domains

The first research stream uses the conclusions drawn from OSS case studies and the lessons learned via studying and investigating OSS, and tries to identify whether these successful lessons can be utilized in other contexts. One of the first concerns has been whether for-profit organizations could profit by adopting the seemingly effective practices of OSS. Seeing how the OSS development model proves to be successful (judging from the results it achieves). it would useful for organizations to follow similar models to make their own software development models more efficient. Several studies point that such a shift (from "traditional" to OSS model of development) can be achieved given that the organizations will be willing to change in terms of processes, internal organization, and philosophy.

Sharma et al. (Sharma, Sugurmaran, & Rajagopalan, 2002) argued that the success of the OSS model is setting the stage for a structural change in the software industry, transforming it from manufacturing to a service industry. Therefore, they explored how for-profit organizations can foster an environment similar to OSS in order to reap its numerous advantages to manage their own software development efforts. After examining the OSS environment using a framework based on dimensions of structure, process, and culture (Galbraith, 1973; Miles & Snow, 1978; Robey, 1991) from organizational theory literature, Sharma et al. (2002) offered their own framework for creating hybrid-OSS communities inside organizations. Their proposed framework included properties of community building, community governance, and community infrastructure. It appears from this work that the ability of organizations to move to a hybrid-OSS environment will depend on:

1. The ability of management and workers to understand the OSS philosophy
2. The development of mutual trust between management and workers
3. The workers' perception of being involved in challenging and innovative projects
4. The motivation of workers to participate in such projects

The arguments of Sharma et al. (2002) concerning the internalization of OSS development model characteristics by organizations is also asserted by another study published a little earlier. In a technical report by the Software Engineering Institute of Carnegie Mellon University, it is stated that "industries, specifically those in the software-development business, are beginning to support the idea of development in the style of OSS." The report analyses several OSS case studies and provides useful conclusions, as well as a set of guidelines for organizations that consider adopting a model of development or internal structure similar to an OSS project.

The organizations' interest to incorporate the successful OSS mechanisms in their own practices was also discussed by Cook (2001), who described the motivators for those involved in OSS, and then defined the problems that need to be overcome for this incorporation to happen.

OSS motivators, according to Cook, are the same factors that appear as contradictions between OSS and CSS (closed-source software). The different "organizational structure" (the benevolent dictatorship), the pool of resources, the level of talent, the motivations for contribution, and the fact that OSS is not attempting to redesign the wheel are the contradictions that are cited as separating OSS projects from commercial organizations. As a result, Cook proposes the features that can be imbibed from OSS environments to traditional organizations:

1. Interacting with customers in certain domains
2. Rewarding talent
3. Allowing individuals to register as potential contributors

Following the same line of thought, several studies have been concerned with whether the successful patterns of online and offline collaboration, team building, and coordination can be utilized in other contexts besides the OSS development environment, for example in organizations. Yamauchi, Yokozawa, Shinohara, and Ishida (2000), based on observation, interviews, and quantitative analysis of two OSS projects (FreeBSD Newconfig Project and GNU GCC Project), found evidence that suggests that spontaneous work coordinated afterward is effective, rational organizational culture helps achieve agreement among members, and communications media moderately support spontaneous work.

These findings could imply a new model of dispersed collaboration. Although the same argument is shared by other researchers as well, like Crowston and Howison (2005), Gallivan (2001) seems to counter this belief by proposing that trust and effective online collaboration of OSS developers is not a critical factor to a project's success, but that it is "various control mechanisms [that] can ensure the effective performance of autonomous agents who participate in virtual organizations" (OSS projects are viewed as virtual organizations by the author). However, Gallivan's study is not backed by empirical data to support this claim, but rather a content analysis of already published OSS case studies and the literature that they are based on.

The general effects of OSS on software engineering have been the subject of considerable analysis. Jorgensen (2001) studied the incremental software development model followed in the FreeBSD project where a stream of contributions goes into a single branch in the repository and is required to preserve the software in a working state. This is a process that creates a succession of development releases, akin to the practices of OSS that utilize frequent releases, but different from the commercial software development line of thought. This fact was also mentioned in "The Cathedral and the Bazaar," where the "release early, release often" (Raymond, 1998) principle of the bazaar model was discussed for its superiority and more effective results. This incremental model has had its effects on more and more commercial software development efforts that have incorporated it.

Scacchi (2002) described and analyzed four OSS development communities to understand the requirements for OSS development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering. He discovered eight kinds of *software informalisms* (software development processes that do not follow the typical route of software development companies) that play a critical role in the elicitation, analysis, specification, validation, and management of requirements for developing open source software systems. This enables considering a reformulation of the requirements engineering process, having a large effect on software engineering in general.

All studies presented in this section share the common characteristic of investigating the applicability of OSS principles and conclusions to other areas of research and practice. In the

following section, we will review studies that belong to the second of the two research streams mentioned earlier.

Investigating how Other Disciplines Apply to the OSS Context

The second research stream is interested in almost the opposite approach. Here we see studies that build on theories established in various disciplines, and review their applicability in the context of OSS. The main disciplines accounting for the majority of the studies of this type are economics, sociology/psychology, software engineering, and network analysis.

Lerner and Tirole (2004) acknowledged the initial puzzlement that OSS causes to an economist. However, they argued that existing economic frameworks can explain OSS- related activities. In their study, they draw on labor and industrial organization literature to give alternative views to the OSS trend. Here, programmers are seen as incurring opportunity costs of time, and the long-term benefits of participating in OSS projects are shown. These long-term incentives are further empowered under three conditions: (a) the more visible the performance to the relevant audience, (b) the higher the impact of effort on performance, and (c) the more informative the performance about talent, for example, Holmström (1999). As a result, Lerner and Tirole conclude that there are economic incentives entangled in the OSS processes, although they are different from the incentives in software development companies and hence may not be directly recognizable.

In a study by Johnson (2002), OSS development is modeled as the private provision of a public good. Such models of public good provision have been studied by many researchers (Bergstrom, Blume, & Varian, 1986; Bliss & Nalebuff, 1984; Chamberlin, 1974; Palfrey & Rosenthal, 1984; Bergstrom, Blume, & Varian, 1986; Bliss & Nalebuff, 1984) and are at the center of economic theory. Based on that, Johnson shows that the superior ability of the open source method to access the Internet talent pool, and to utilize more private information, provides an advantage over the closed source method in some situations. Nonetheless, free riding (which is a crucial problem when discussing public goods) implies that some valuable projects will not be produced, even when the community of developers becomes large.

Bitzer, Wolfram, and Schroder (2004) adapted a dynamic private-provision-of-public-goods model to reflect key aspects of the OSS phenomenon. In particular, instead of relying on extrinsic motives for programmers (e.g., signaling), his model was driven by intrinsic motives of OSS programmers, arguing that since programming software is associated with the risk of failure (e.g., in terms of the development of the software is not successful or the project does not become famous), extrinsic motives (signaling) are unable to explain the OSS phenomenon in full, and can rarely be linked to the motives of initiators of OSS projects. This approach, in a sense, challenges earlier views that analyzed the economic aspects of OSS based on extrinsic motives. According to Bitzer et al., the motives for a programmer to initiate a project are the mix of:

1. The need for a particular software solution
2. Fun or play
3. Gift culture, social standing

Motivation of OSS developers has been a recurrent theme for studies either from an economic (Torvalds & Diamond, 2001; Hars & Ou, 2002; Hertel, Nieder, & Herrmann, et al., 2003; Krishnamuturthy, 2002; Lakhani & Wolf, 2003) or a sociological/psychological perspective (Weber, 2004).

In discussing a framework for analyzing OSS, Feller and Fitzerald (2000) study and use two previous frameworks which that have been very influential in the IS field: Zachman's IS architecture (ISA) and Checkland's CATWOE framework from soft systems methodology (SSM).

Furthermore, Lawrie and Gacek (2002) use basic software engineering principles and metrics to discuss dependability issues regarding OSS. There is a large number of studies discussing technical or evaluation issues in OSS that draw on the software engineering scientific area, but their technicality does not support our analysis, and thus they are not presented in this literature review.

Through this overview, it became evident that the area of OSS-related research spans a number of disciplines and contexts. Researchers of OSS today are not strictly software engineers, and issues of interest are not limited to development and engineering principles. A lot of different perspectives have been considered in order to explain OSS expansion and philosophy and, at the same time, a lot of different domains of research have stepped up with the desire to utilize all the successful practices that OSS development has to offer. It is a collective aim for the research and analysis of OSS development to continue, and for knowledge and best practices to be transferred to other areas as well.

CONCLUSION

The purpose of this chapter was really introductory. Many of the FOSS historical issues as well as research topics will be covered in detail in the next chapters, where special emphasis will be paid to the knowledge and learning management context. From this point of view, it is really exciting to reveal the mutual beneficial relationships of FOSS and knowledge and learning management and their linkages that support sustainable performance and development.

REFERENCES

Bergstrom, T., Blume, L., & Varian, H. (1986). On the private provision of public goods. *Journal of Public Economics, 29,* 25-49.

Bezroukov, N. (1997). A second look at the cathedral and the bazaar. *First Monday, 4*(12). Retrieved on 20/11/2006 http://www.firstmonday.org/issues/issue4_12/bezroukov/index.html

Bitzer, J., Wolfram, S., & Schroder, P. H. J. (2004). *Intrinsic motivation in open source software development.* MIT Working Paper.

Bliss, C., & Nalebuff, B. (1984). Dragon-slaying and ballroom dancing: The private supply of a public good. *Journal of Public Economics, 25,* 1-12.

Chamberlin, J. (1974). Provision of collective goods as a function of group size. *American Political Science Review, 68,* 707-716.

Cook, J. E. (2001). Open source development: An Arthurian legend. In J. Feller, B. Fitzgerald, & A. van der Hoek (Eds.), *Making sense of the bazaar: Proceedings of the 1st Workshop on Open Source Software Engineering.* Retrieved May 19, 2006, from http://opensource.ucc.ie/icse2001/papers.htm

Cox, A. (1998). *Cathedrals, bazaars and the town council.* Retrieved March 22, 2004, from http://slashdot.org/features/98/10/13/1423253.shtml

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday, 10*(2). Retrieved on 20/11/2006 http://www.firstmonday.org/issues/issue10_2/crowston/index.html

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In W. Orlikowski, P. Weill, S. Ang, & H. Krcmar (Eds.), *Proceedings of the 21st Annual International Conference on Information Systems* (pp. 58-69). Brisbane, Queensland, Australia.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE software, 21*(1), 34–40.

Galbraith, J. R. (1973). *Designing complex organizations.* Reading, MA: Addison-Wesley

Gallivan, M. J. (2001). Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal, 11*(4), 277-304.

Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce, 6*(3), 25-39.

Hertel, G., Nieder, S., & Herrmann, S. (2003). Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux Kernel. *Research Policy* (Special Issue: Open Source Software Development), *32*(7), 1159-1177.

Homström, B. (1999). Managerial incentive problems: A dynamic presective. *Review of Economic Studies, 66,* 169-182.

Johnson, J.P. (2002). Economics of open source software: Private provision of a public good. *Journal of Economics & Management Strategy, 11*(4), 637-662.

Jørgensen, N. (2001). Putting it all in the trunk: Incremental software development in the FreeBSD open source project. *Information Systems Journal, 11*(4), 321-336.

Krishnamurthy, S. (2002). Cave or community? An emprical examination of 100 mature open source projects. *First Monday, 7*(6). Retrieved May 19, 2006, from http://www.firstmonday.org

Lakhani, K., Arim, R., & Wolf, R. G. (2003). *Why hackers do what they do: Understanding motivation effort in free/open source software projects.* MIT Sloan School of Management Working Paper, no. 4425-03.

Lawrie, T., & Gacek, C. (2002). Issues of dependability in open source software development. *ACM SIGSOFT Software Engineering Notes, 27*(3), 34-37.

Lerner, J., & Tirole, J. (2004). *The economics of technology sharing: Open source and beyond.* NBER (Working Paper 10956).

Miles, R. E., & Snow, C. C. (1978). *Organizational strategy, structure, and process.* New York: McGraw-Hill.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of Linux kernel. *First Monday, 5*(11). Retrieved on 20/11/2006 http://www.firstmonday.org/issues/issue5_11/moon/index.html

Palfrey, T. R., & Rosenthal, H. (1984). Participation and the provision of discrete public goods: A strategic analysis. *Journal of Public Economics, 24,* 171-193.

Pappas, J. (2001). *Economics of open source software.* Working Paper. Retrieved May 19, 2006, from http://opensource.mit.edu

Raymond, E. S. (1998). The cathedral and the bazaar. *First Monday, 3*(3). Retrieved May 19, 2006, from http://www.firstmonday.org/issues/issue3_3/raymond/

Robey, D. (1991). *Designing organizations* (2nd ed.). Burr Ridge, IL: Irwin.

Salus, P. (2005). *The daemon, the gnu and the penguin.* (Published as a series of articles in Groklaw). Retrieved May 19, 2006, from http://www.groklaw.net/article.php?story=20050623114426823

Stallman, R. (1999). The GNU operating system and the free software movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution.* Cambridge, MA: O'Reilly and Associates.

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEEE Proceedings - Software, 48*(1), 24-39.

Sharma, S., Sugurmaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-open source software communities. *Information Systems Journal, 12*(1), 7-25.

Torvalds, L., & Diamond, D. (2001). *Just for fun: The story of an accidental revolutionary.* HarperBusiness.

Vixie, P. (1999). Open source software engineering. In C. DiBona, S.Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution*. Cambridge, MA: O'Reilly and Associates.

Weber, S. (2004). *The success of open source*. Cambridge: Havard University Press.

Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. (2000). Collaboration with lean media: How open source software succeeds. In *Proceedings of the ACM Conference on Computer-Supported Work* (pp. 329-338).

APPENDIX I: USEFUL URLS

A free/open source research community (also provides a database of online papers) **http://opensource.mit.edu/**

The Free Software Foundation
http://www.fsf.org/

Freshmeat
http://freshmeat.net/

Libresoft
http://libresoft.urjc.es/index

The Open Source Initiative (OSI)
http://www.opensource.org/

SourceForge.net
http://sourceforge.net/

APPENDIX II: FURTHER READING

DiBona, C., Ockman, S., & Stone, M. (1999). *Open sources: Voices from the open source revolution.* Sebastopol, CA: O'Reilly & Associates.

Glyn Moody, G. (1997). The greatest OS that (n)ever was. *Wired, 5*(8). Retrieved from http://pauillac.inria.fr/~lang/hotlist/free/wired/linux.html

Raymond, E. S. (2003). *The art Of Unix programming.* Addison-Wesley.

Senyard, S., & Michlmayr, M. (2004). How to have a successful free software project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, Busan, Korea (pp. 84-91). IEEE Computer Society.

This work was previously published in Open Source for Knowledge and Learning Management: Strategies Beyond Tools, edited by M. Lytras and A. Naeve, pp. 1-15, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 1.4
Open-Source Software Issues

Sofiane Sahraoui
American University of Sharjah, UAE

INTRODUCTION

Two major research publications have recently dedicated special issues to the emerging field of OSS (Research policy, 2003; Management Science, 2006). Likewise, major information systems conferences are starting to list OSS as a research track (IRMA 2004, Working IFIP 8.6) translating the heightened importance of this phenomenon in the business world. Undoubtedly, OSS has been admitted as a legitimate field of study in the realm of business academics, but OSS research remains largely trailing the gigantic developments in the open-source industry. For instance, there are recurrent speculations in specialized IT magazines that Microsoft will go down the drain with the OSS phenomenon (Fontana, 2003); Oracle will have a hard time maintaining its supremacy in the database market (Bank, 2003); and Apple might come out the big winner by latching on to OSS (Brockmeire, 2003). However, there is hardly any solid piece of academic research to forecast the outlook of the IT industry in light of the surging OSS phenomenon. Existing research, including the two special issues mentioned above, is focusing on validating models of innovation in a virtual environment (Franke, 2003; Von Krogh, Spaeth, & Lakhani, 2003; Hippel, 2001); tracking project management dynamics in OSS development (Hertel, Neidner, & Herrmann, 2003; O'Mahony, 2003); examining the intellectual property, ethical and legal implications of OSS (Evers, 2000; Faldetta, 2002); or reworking the economics underlying software development in the case of OSS (Zeitlyn, 2003). Much less has been done in critical areas pertaining to the new competitive game introduced by OSS; the sustainability of the OSS business model or models; the strategies for OSS licensing; the economic and business viability of OSS in light of potential challenges and opportunities; and the nascent national and government IT strategies centered on OSS; plus

a variety of other issues that are beyond the scope of this short article.

This article seeks to clarify the critical factors that will increasingly determine the success of OSS in becoming a mainstream choice for software procurement processes. Along with a definition of each of these factors, potential research avenues are indicated. However, these factors are not meant to be exhaustive in any fashion.

OSS: A BRIEF DEFINITION

The most basic definition of open source software is software for which the source code is distributed along with the executable program, and which includes a license allowing anyone to modify and redistribute the software.

Actual licenses for OSS vary between different companies and development projects, but they have certain characteristics in common. The Open Source Initiative, a group of developers who disseminate information on the benefits of open source (see www.opensource.org) has posted on its Web site a "meta-definition" of basic conditions they feel should be included in an OSS license. These include:

- Allowing free redistribution of the software without royalties or other fees to the author.
- Requiring that source code be distributed with the software or otherwise made available for no more than the cost of distribution.
- Allowing anyone to modify the software or derive other software from it, and to redistribute the modified software under the same license terms.
- Any software distributed under a license that conforms to these requirements is open-source software, according to the Open Source Initiative.

Although OSS solutions are increasingly available for an increasing number of business applications, it is very unlikely, however, that users are currently undertaking comparisons between alternative solutions. Nonetheless, it is a matter of time—and probably a short time—until open-source solutions become a must-consider item in software procurement decisions.

OPEN-SOURCE LICENSING

Many different types of OSS licenses are in use, almost as many as 50. The increasing number of license types under which open-source software is distributed could prove damaging to the overall development of OSS. Nonetheless, as a group OSS licenses could be clearly distinguished from conventional proprietary licenses. The latter are generally designed to take away the user freedom to share and change the software, which is the object of the license. By contrast, open-source licenses explicitly guarantee the freedom to share and change software without any permission from its original owner (Evers, 2000).

The General Public License (GPL; see www.opensource.org/licenses/gpl-license.html is the most important license, as key open-source software solutions are distributed under its terms; most notably, the Linux kernel (Evers, 2000). Not only does the GPL guarantee the freedom to share and change software, but it also requires that anything linked with the concerned software be distributed as free software, as well. This is known as the "virus" effect (Evers, 2000). A consequence of this is that any software developed based on the Linux Kernel, for example, has to be shared back with the open-source community, hence released under a GPL itself. This has had a very positive consequence on the development of Linux as a major player in the server open-source market and even as a potential contender to unsettle Windows desktop hegemony. Conversely, other operating systems distributed under open-source licenses

that do not require re-channeling changes, such as the BSD license, have led to dispersed programming efforts and even an appropriation of open-source code by proprietary vendors. Apple's latest version of its proprietary operating system (i.e., Mac OS X) is heavily based on Darwin, a code that is freely available under the BSD license.

One major research avenue would be to analyze the open-source strategies that proprietary vendors are currently undertaking and the opportunities offered by various licensing schemes to further such strategies. Such a study will survey the existing or potential open-source strategies of proprietary vendors and match them to appropriate licensing schemes, hence performing a competitive outlook of the OSS industry. The study would also cover the competitive alternatives, if any, of proprietary vendors that will not likely go the open-source way.

BUSINESS MODEL

When users acquire software through an open-source license, they truly become owners of the software, which means they inherit the right to modify it, share it, redistribute it and even resell it if they have made significant additions, which they can license themselves. Software in the open-source licensing model, hence, is treated like a commodity rather than intellectual property that has to be traced back to its original author. Indeed, major applications in open source and databases are in widespread use (i.e., mass market); the industry has settled on common standards, and new features are less important than price and performance, hence reducing the importance of the intellectual content of the software and turning it more into a commodity, just like what generic drugs have done for branded ones (Karp, 2003). Proprietary licensing restrictions would not allow any of this.

While the concept is appealing from an intellectual and maybe an ethical point of view, it raises questions about the sustainability of its business model (Zieger, 2003). As software is steadily turning into a commodity, will sales volume be enough to secure the survival and growth of OSS vendors, especially software that can be virtually replicated for free? How can OSS vendors make money from software that is essentially free and does not require much servicing, then?

Besides pinning down the business model for OSS, this line of research would analyze existing OSS business models, both successful and unsuccessful, and speculate on an industry structure for the OSS market in the long term.

SECURITY AND MARKET OPPORTUNITIES FOR OSS

OSS would be less susceptible to hackers than proprietary software. OSS transparency allegedly increases security because "back doors" used by hackers can be exposed and programmers can root out bugs from the code (The Economist Group, 2003). On the other hand, security, a top concern for software users, is increasingly proving to be the Achilles' wheel of proprietary software, and especially for Windows. A number of governments around the world are wary of repeated computer-virus attacks that target Microsoft's Windows operating system (Yamada, 2003). The latest attacks by the Blaster and SoBig viruses have indeed increased concerns about Windows security (The Economist Group, 2003).

At this level, one can investigate the magnitude of the security problem with proprietary software and the opportunities that OSS could present in this respect. Along with other factors such as cost, performance and service support among others, such a study could give a clear indication on the true potential of OSS in a particular market or region.

GOVERNMENT AND NATIONAL IT STRATEGIES

Security has yet another aspect that is important for government clients especially. They usually try to avoid relying on a single open source, vendor or center of operation. For instance, the ministry of interior in Germany justified the country's decision to adopt OSS by the need to raise the level of IT security by avoiding monocultures (Rajani, 2003). Likewise, the government of China has been working on a local version of Linux, on the grounds of self-sufficiency, security and to avoid being too dependent on a single foreign supplier (The Economist Group, 2003). OSS is being increasingly used in politics as an extension of nationalist discourses advocating national sovereignty, the right to access to knowledge and national security. The free software movement in Brazil, for example, has gained momentum since the Leftist Workers Party took office in January 2003 (Karp, 2004). The governments of Japan, China and South Korea will collaborate with major high-tech companies to develop open-source software products that will offer an alternative to the Windows operating system (Yamada, 2003). Already, 60 countries around the world are either considering or have passed legislation that encourages—or requires consideration—of free software. Even in the United States (U.S.), where proprietary software giants like Microsoft and Oracle yield massive lobbying power, the U.S. Congress has established an entity called the National Technology Alliance to give agencies guidance on how to contract with vendors using open source, and what kind of technology is available (Zieger, 2003).

Research is needed to determine the motives and configurations of national OSS strategies that have sprung up worldwide. Security is probably only one element of the equation, alongside cost, reducing the digital divide and so forth.

TOTAL COST OF OWNERSHIP (TCO)

Several studies have been done or are under way and have reported conflicting findings. Studies tend to favor those who paid for them (Maguire, 2003) and findings are generally disputed (Varghese, 2003).

TCO means the total amount of money that the decision of introducing new software costs, which can exceed the selling price of the software. Other cost factors include system preparation, including hardware and other necessary software; man-hours to handle the software installation, operation and maintenance; user training; updates; cost of migration to the new software, including any changes to business processes; and so forth (Evers, 2000). The general argument of proprietary software vendors is that the cost of software itself does not exceed 20% to 30% of the TCO, and even much less than that for organizations such as schools benefiting from special pricing schemes (Maguire, 2003b). Beyond that, the OSS show is allegedly dismal and carries a heavier cost than proprietary software. Unfortunately, no methodologically sound research is available to corroborate or deny such findings. Research is urgently needed to sift the debate on TCO and develop costing models for various contexts.

THE ETHICAL ARGUMENT

From an academic point of view, the ethical debate surrounding OSS is likely to revive the conventional and overheated debate of software piracy and its ethical connotation. The OSS model is indeed throwing the debate open about the legitimacy and even the ethics of restricting the "free flow of software." Many have questioned the proprietary model of software development and distribution and deemed it unethical (see www.netaction.org/msoft/world/). Microsoft and its allies have counter attacked and sought to discredit OSS,

likening its challenge of proprietary ownership to communism and suggesting that its openness makes it insecure and therefore vulnerable to terrorism (Fontana, 2003). Microsoft-supported lobby groups, such as the Initiative for Software Choice and the Business Software Alliance, have been staging major campaigns worldwide to confuse the public about the OSS phenomenon and have blocked legislation supporting the adoption of OSS by governments in different countries of the world (Adelstein, 2003).

The ethical argument extends beyond the issue of copying software, however. The City of Munich, for instance, did not wish to place the functioning of government in the hands of a commercial vendor with proprietary standards that is accountable to shareholders rather than to citizens (Fontana, 2003). Besides security, OSS seems to provide an alternative to governments not to lock their citizens onto the technology of one overpowering vendor. Bridging the digital divide, another ethical imperative for governments worldwide, seems to be more attainable with free software.

The issues are many and are worth debating. Research is first needed to delineate the most pressing items, lay out the stakes and unfold rational discourses to engage long-held and perhaps erroneous perceptions and beliefs. Deconstructing the classical argument underlying the ethics of software piracy should be a thrilling endeavor for any researcher.

CONCLUSION

This outline of a research agenda makes no claim of starting a new research stream that is well underway. It simply summarizes the author's newfound interest and perceptions of the issues surrounding the OSS phenomenon. In addition, many important avenues for research have not been indicated here, simply because the work is at its inception phases and is based on a limited number of readings in the area. This shall be expanded while issues are uncovered and more research is published. Finally, only business-related issues have been addressed; issues relating to software engineering, standards, architecture and so forth have been purposefully left out.

REFERENCES

Aldestein, T. (2003). Linux access in state and local government. *Linux Journal*, 1-5. Retrieved from *www.Linuxjournal.com/print.php?sid=6927*

Bank, D. (2003). 'Open source' database poses Oracle threat. *Wall Street Journal (Eastern Edition)*, B1.

Brockmeire, J. (2003). Is open source Apple's salvation?. *NewsFactor Network*, 1-2. Retrieved from *www.linuxenterprisenews.com/perl/printer/21244/*

The Economist Group. (2003). Microsoft at the power point. Retrieved from *www.economist.com/business/printerfriendly.crm/Story_ID=2054740*

Evers, S. (2000). An introduction to open source software development. Retrieved from *http://user.cs.tu-berlin.de/~tron/opensource*

Faldetta, G. (2002). The content of freedom in resources: The open source model. *Journal of Business Ethics*, *39*(1), 179-188.

Fontana, J. (2003). Linux marches on: Microsoft marshals forces to try to stem open source momentum. *Network World, 20*, 1.

Franke, N., & Hippel, V. Satisfying heterogeneous user needs via innovation toolkits: The case of Apache security software. *Research Policy, 32*(7), 1199-1215.

Hertel, G., Neidner, S., & Herrmann, S. (2003). Motivation of software developers in open source projects: An Internet-based survey of contributors

to the Linux kernel. *Research Policy, 32*(7),1159-1177.

Hippel, V. (2001). Innovation by user communities: Learning from open-source software. *Sloan Management Review, 42*(4), 82-86.

Karp, J. (2003). A Brazilian challenge for Microsoft: The government's preference for open source software may till the playing field. *Wall Street Journal (Eastern Edition),* A.14.

Maguire, J. (2003a). Has Linux eclipsed open source. *Enterprise Linux IT,* 1-2.

Maguire, J. (2003b). Open source on the brink. *Enterprise Linux,* 1-2. Retrieved from *www.Linux enterprise news.com/perl/printer/22278/*

O'Mahony, S. (2003). Guarding the commons: How community managed software projects to protect their work. *Research Policy, 32*(7), 1179-1198.

Rajani, N. (2003). Free as in education: Significance of free/libre and open source software for developing countries. Retrieved from *www.maailma.kaapeli.fi/OSSReport1.0.html#mozTocId13212*

Varghese, S. (2003). Gartner findings on desktop Linux disputed. *SMH,* 1-3. Retrieved from *www.smh.com.au/articles/2003/09/16/1063625013703.html*

Von Krogh, G., Spaeth, S., & Lakhani, K. (2003). Community, joining, and specialization in open source software innovation: A case study. *Research Policy, 32*(7), 1217-1241.

Yamada, M. (2003). Asian countries seek Windows alternative. *Wall Street Journal (Eastern Edition),* B.10.

Zeitlyn, D. (2003). Gift economies in the development of open source software: Anthropological reflections. *Research Policy, 32*(7), 1287-1291.

Zieger, A. (2003). Open-minded. *Information Week,* 25-28.

KEY TERMS

Linux Kernel: Basic programming code for the Linux operating system.

Linux: Open source operating system.

Open Source Community: Community of developers that interact virtually to write code collaboratively.

Open Source Initiative: Group of developers who disseminate information on the benefits of open source.

Open Source License: Guarantees the freedom to share and change software without any permission from its original owner.

Open Source Software (OSS): Software for wish the source code is distributed along with the executable program.

Proprietary Software: Its source code is not available.

Source Code: Programming code prior to execution.

TCO: Total Cost of Ownership (TCO): The total amount of money that the decision of introducing new software costs.

This work was previously published in Encyclopedia of Virtual Communities and Technologies, edited by S. Dasgupta, pp. 368-371, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 1.5
Open Source Software:
Strengths and Weaknesses

Zippy Erlich
The Open University of Israel, Israel

Reuven Aviv
The Open University of Israel, Israel

ABSTRACT

The philosophy underlying open source software (OSS) is enabling programmers to freely access the software source by distributing the software source code, thus allowing them to use the software for any purpose, to adapt and modify it, and redistribute the original or the modified source for further use, modification, and redistribution. The modifications, which include fixing bugs and improving the source, evolve the software. This evolutionary process can produce better software than the traditional proprietary software, in which the source is open only to a very few programmers and is closed to everybody else who blindly use it but cannot change or modify it. The idea of open source software arose about 20 years ago and in recent years is breaking out into the educational, commercial, and governmental world. It offers many opportunities when implemented appropriately. The chapter will present a detailed definition of open source software, its philosophy, its operating principles and rules, and its strengths and weaknesses in comparison to proprietary software. A better understanding of the philosophy underlying open source software will motivate programmers to utilize the opportunities it offers and implement it appropriately.

INTRODUCTION

Open source software (OSS) has attracted substantial attention in recent years and continues to grow and evolve. The philosophy underlying OSS is to allow users free access to, and use of, software source code, which can then be adapted, modified, and redistributed in its original or modified form for further use, modification, and redistribution. OSS is a revolutionary software development methodology (Eunice, 1998) that involves developers in many locations throughout

the world who share code in order to develop and refine programs. They fix bugs, adapt and improve the program, and then redistribute the software, which thus evolves. Advocates of OSS are quick to point to the superiority of this approach to software development. Some well-established software development companies, however, view OSS as a threat (AlMarzouq, Zheng, Rong, & Grover, 2005).

Both the quality and scope of OSS are growing at an increasing rate. There are already free alternatives to many of the basic software tools, utilities, and applications, for example, the free Linux operating system (Linux Online, 2006), the Apache Web server (Apache Software Foundation, 2006; Mockus, Fielding, & Herbsleb, 2000), and the Sendmail mail server (Sendmail Consortium, 2006). With the constant improvement of OSS packages, there are research projects, even complex ones, that entirely rely on OSS (Zaritski, 2003). This opens new research and educational opportunities for installations and organizations with low software budgets.

Incremental development and the continuity of projects over long periods of time are distinctive features of OSS development. The software development processes of large OSS projects are diverse in their form and practice. Some OSS begins with releasing a minimal functional code that is distributed for further additions, modification, and improvement by other developers, as well as by its original authors, based on feedback from other developers and users. However, open source projects do not usually start from scratch (Lerner & Tirole, 2001). The most successful OSS projects, like Linux and Apache, are largely based on software provided by academic and research institutions. In recent years, more and more OSS has been derived from original software provided by for-profit companies.

A large potential-user community is not enough to make an OSS project successful. It requires dedicated developers. In Raymond's (1998) words, "The best OSS projects are those that scratch the itch of those who know how to code." For example, the very successful Linux project attracted developers who had a direct interest in improving an operating system for their own use. Similarly, webmaster developers contributed to the development of the Apache Web server project.

Despite the characterization of the OSS approach as ad hoc and chaotic, OSS projects appear, in many cases, to be highly organized, with tool support that focuses on enhancing human collaboration, creativity, skill, and learning (Lawrie & Gacek, 2002). The good initial structural design of an OSS project is the key to its success. A well-modularized design allows contributors to carve off chunks on which they can work. In addition, the adoption of utility tools and the use of already existing OSS components are necessary if an OSS project is to succeed.

The growing interest of commercial organizations in developing and exploiting OSS has led to an increased research focus on the business-model aspects of the OSS phenomenon. There are a number of business models for OSS, all of which assume the absence of traditional software licensing fees (Hecker, 2000). The economics of OSS projects is different from that of proprietary projects (Lerner & Tirole, 2002). Models of effort and cost estimation in the development of projects involving OSS are needed (Asundi, 2005).

In the past, most OSS applications were not sufficiently user friendly and intuitive, and only very knowledgeable users could adapt the software to their needs. Although the use of OSS is growing, OSS is still mainly used by technically sophisticated users, and the majority of average computer users use standard commercial proprietary software (Lerner & Tirole, 2002). The characteristics of open source development influence OSS usability (Behlendorf, 1999; Nichols, Thomson, & Yeates, 2001; Raymond, 1999), which is often regarded as one of the reasons for its limited use. In recent years, the open source community has shown increased awareness of

usability issues (Frishberg, Dirks, Benson, Nickell, & Smith, 2002). Existing human-computer interface (HCI) techniques and usability improvement methods appropriate for community-based software development on the Internet can be used to leverage distributed networked communities to address issues of usability (Nichols & Twidale, 2003). Some OSS applications, such as the Mozilla Web browser (Mozilla, 2006; Reis & Fortes, 2002) and OpenOffice (OpenOffice, 2006), have made important advances in usability and have become available for both Windows and Linux users.

As the stability and security of open source products increase, more organizations seem to be adopting OSS at a faster rate. There are many open source community resources and services online. When implemented appropriately, OSS offers extensive opportunities for government, private-sector, and educational institutions. OSS appears to be playing a significant role in the acquisition and development plans of the U.S. Department of Defense and of industry (Hissam, Weinstock, Plakosh, & Asundi, 2001).

For many organizations, integrating the revolutionary OSS developmental process into traditional software development methods may have a profound effect on existing software development and management methodologies and activities.

The remainder of this chapter will review the history of OSS and define some key terms and concepts. It will discuss the incentives to engage in OSS and its strengths and weakness. Finally, it will review some OSS business models.

BACKGROUND

Software source code that is open has been around in academic and research institute settings from the earliest days of computing. Feller and Fitzgerald (2002) provide a detailed historical background to open source since the 1940s. The source code of programs developed in universities, mainly as learning and research tools, was freely passed around. Many of the key aspects of computer operating systems were developed as open source during the 1960s and 1970s in academic settings, such as Berkeley and MIT, as well as in research institutes, such as Bell Labs and Xerox's Palo Alto Research Center, at a time when sharing source code was widespread (Lerner & Tirole, 2002).

The free software (FS) movement began in the 1980s in academic and research institutes. The Free Software Foundation (FSF) was established by Richard Stallman of the MIT Artificial Intelligence Laboratory in 1984. The basic idea underlying the foundation was to facilitate the development and free dissemination of software. It is important to note that free in this case relates not to price, but to freedom of use. The developers of the Linux operating system bought in to the FS concept. Linux, initiated by Linus Tovalds in 1991, was the first tangible achievement of the FS movement (Stallman, 1999). This successful operating system has, through the collaboration of the global FS community, grown into the second most widely used server operating system.

The OSS movement evolved from the FSF during the 1990s. OSS has more flexible licensing criteria than the FSF. The widespread use of the Internet led to acceleration in open source activities. Numerous open source projects emerged, and interaction between commercial companies and the open source community became commonplace. Unlike the FS community, the OSS movement does not view itself as a solution for proprietary software, but rather as an alternative to it (Asiri, 2003). This has led to the acceptance of selective open sourcing, in which companies may elect to make specific components of the source code, rather than the entire code, publicly available, an approach which appeals to the business community. This allows companies to package available OSS products with other applications and extensions, and sell these to customers. Profit can also be made on the exclusive support pro-

vided with the retail packages, which may include manuals, software utilities, and support help lines. For example, Red Hat Software (Red Hat, 2006), the leading provider of the Linux-based operating system, founded in 1995, based its business model on providing Linux software for free and selling extras such as support, documentation, and utilities, making it easy for users to install and use the software.

Definitions: FS and OSS

According to the FSF, FS involves users' freedom to run, copy, distribute, study, change, and improve software. The FSF defined four kinds of freedom for software users (Free Software Foundation, 2006).

1. The freedom to run the program for any purpose
2. The freedom to study how the program works, and adapt it to one's needs; access to the source code is a precondition for this
3. The freedom to redistribute copies so one can help a neighbor
4. The freedom to improve the program and release improvements to the public so that the whole community benefits; access to the source code is a precondition for this

The FSF recommends the GNU (a Unix-compatible operating system developed by the FSF) General Public License (GPL; Free Software Foundation, 1991) to prevent the GNU operating system software from being turned into proprietary software. This involves the use of "copyleft," which Stallman (1999) defines as follows:

The central idea of copyleft is that we give everyone permission to run the program, copy the program, modify the program, and distribute modified versions—but not permission to add restrictions of their own. Thus, the crucial freedoms that define "free software" are guaranteed to everyone who has a copy; they become inalienable rights.

The GPL permits the redistribution and reuse of source code for unfettered use and access as long as any modifications are also available in the source code and subject to the same license.

The term OSS was adopted in large part because of the ambiguous nature of the term FS (Johnson, 2001). On the most basic level, OSS simply means software for which the source code is open and available (Hissam et al., 2001), and that anyone can freely redistribute, analyze, and modify while complying with certain criteria (AlMarzouq et al., 2005). However, OSS does not just mean access to source code. For a program to be OSS, a set of distribution terms must apply.

A comprehensive Open Source Definition (OSD) was published by the Open Source Initiative (OSI). The OSD differentiates itself from FS by allowing the use of licenses that do not necessarily provide all the freedoms granted by the GPL. According to the updated version of the OSD (1.9), the distribution terms of OSS must comply with all 10 of the following criteria (Open Source Initiative, 2005).

1. **Free redistribution:** The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Source code:** The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost—preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the

output of a preprocessor or translator are not allowed.

3. **Derived works:** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of the author's source code:** The license may restrict source code from being distributed in modified form only if the license allows the distribution of patch files with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No discrimination against persons or groups:** The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavor:** The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. **Distribution of license:** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License must not be specific to a product:** The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
9. **License must not restrict other software:** The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open source software.
10. **License must be technology neutral:** No provision of the license may be predicated on any individual technology or style of interface.

Although there are some differences in the definitions of OSS and FS, the terms are often used interchangeably. Neither OSS nor FS pertains to the source code and its quality, but rather to the rights that a software license must grant. Various licensing agreements have been developed to formalize distribution terms (Hecker, 2000). Open source licenses define the privileges and restrictions a licensor must follow in order to use, modify, or redistribute the open source software. OSS includes software with source code in the public domain and software distributed under an open source license. Examples of open source licenses include the Apache license, Berkeley Source Distribution (BSD) License, GNU GPL, GNU Lesser General Public License (LGPL), MIT License, Eclipse Public License (EPL), Mozilla Public License (MPL), and Netscape Public License (NPL).

Table 1 provides a comparison of several common licensing practices described in Perens (1999).

The OSI has established a legal certification for OSS, called the OSI certification mark (Open Source Initiative, 2006b). Software that is distributed under an OSI-approved license can be labeled "OSI Certified."

Incentives to Engage in OSS

A growing body of literature addresses the motives for participation in OSS projects. Lerner and Tirole

Table 1. Comparison of licensing practices (Source: Perens, 1999)

License	Can be mixed with non-free software	Modifications can be made private and not returned	Can be relicensed by anyone	Contains special privileges for the original copyright holder over others' modifications
GPL				
LGPL	X			
BSD	X	X		
NPL	X	X		X
MPL	X	X		
Public Domain	X	X	X	

(2001) describe incentives for programmers and software vendors to engage in such projects:

Programmers' Incentives

- Programmers are motivated by a desire for peer recognition. Open source programmers' contributions are publicly recognized. By participating in an OSS project, programmers signal their professional abilities to the public.
- Programmers feel a duty to contribute to a community that has provided a useful piece of code.
- Some programmers are motivated by pure altruism.
- Some sophisticated OSS programmers enjoy fixing bugs, working on challenging problems, and enhancing programs.
- OSS is attractive to computer science students who wish to enter the market as programmers in higher positions.

Software Vendors' Incentives

- Vendors make money on OSS complementary services such as documentation, installation software, and utilities.
- By allowing their programmers to get involved in OSS projects, vendors keep abreast of open source developments, which allows them to better know the competition.
- Vendors benefit from efficient use of global knowledge. Many companies can collaborate on a product that none of them could achieve alone.

MAIN FOCUS OF THE CHAPTER

OSS Strengths and Weaknesses

OSS has a number of strengths and weaknesses compared to traditional proprietary software.

Strengths

The strengths of OSS can be classified into five main categories: freedom of use; evolution of software; time, cost, and effort; quality of software; and advantages to companies and programmers.

Freedom of Use

- It allows free access to the software source code for use, modification, and redistribution in its original or modified form for further use, modification, and redistribution.
- OSS users have fundamental control and flexibility advantages by being able to modify and maintain their own software to their liking (Wheeler, 2005).

- OSS allows independence from a sole source company or vendor. It provides users with the flexibility and freedom to change between different software packages, platforms, and vendors, while secret proprietary standards lock users into using software from only one vendor and leave them at the mercy of the vendor at a later stage (Wong & Sayo, 2004).
- It eliminates support and other problems if a software vendor goes out of business.
- It prevents a situation in which certain companies dominate the computer industry.
- Users can get free upgrade versions of the software, switch software versions, and fix and improve software (Perens, 1999).

Evolution of Software

- OSS contributes to software evolution due to the parallel process of many developers being simultaneously involved rather than a single software team in a commercial proprietary software company (Feller & Fitzgerald, 2002).
- It enables programmers all over the world to fix bugs.
- It evolves continuously over time as opposed to proprietary software whose development takes place in a series of discrete releases under the control of the authors.
- OSS represents a viable source of components for reuse and to build systems.

Time, Cost, and Effort

- It involves a joint effort by contributors from countries all over the world, collaborating via the Internet.
- There is a lower cost of software development in comparison to proprietary software.
- Open source initiatives allow software to be developed far more quickly and permits bugs to be identified sooner.
- The OSS approach is not subject to the same level of negative external process constraints of time and budget that can often undermine the development of dependable systems within an organizational setting (Lawrie & Gacek, 2002).
- OSS reduces the cost of using the software as the licensing is not limited compared to the limited licensing of proprietary software. The licensing cost, if any, is low, and most OSS distributions can be obtained at no charge. On a licensing cost basis, OSS applications are almost always much cheaper than proprietary software (Wong & Sayo, 2004). Open source products can save not-for-profit organizations, such as universities and libraries, a lot of money.
- It reduces development time, cost, and effort by reusing and building on existing open source code.
- It reduces maintenance and enhancement costs by sharing maintenance and enhancements among potential users of the same software application.

Quality of Software

- OSS reduces the number of bugs and enhances software quality by using the feedback of many users around the world and other qualified developers who examine the source code and fix the bugs.
- OSS is under constant peer review by developers around the world. Linus' law states the following: "Given enough eyeballs, all bugs are shallow" (Raymond, 1998).
- Programmers, knowing in advance that others will see the code they write, will be more likely to write the best code they can possibly write (Raymond, 1998).
- Security vulnerabilities are more quickly solved when found in OSS than in proprietary software (Reinke & Saiedian, 2003).

- OSS represents an alternative approach to distributed software development able to offer useful information about common problems as well as possible solutions (Johnson, 2001).

Advantages to Companies and Programmers

- There is an efficient use of global knowledge.
- Programmers learn from existing source code how to solve similar problems.
- Students, especially computer science students, can gain excellent programming experience and make contributions to open source software by becoming involved in open source projects (Zaritski, 2003).
- OSS allows groups of companies to collaborate in solving the same problem.
- Companies gain leverage from developers who contribute free improvements to their software.
- Companies using OSS benefit from its very rapid development, often by several collaborating companies, much of it contributed by individuals who simply need an improvement to serve their own needs (Perens, 1999).

Weaknesses

OSS weaknesses are mainly related to management, quality, and security.

Management

- Given the difficulty in managing resources in closed source proprietary software projects, planning and delivering projects based on an open source community can be a much bigger challenge (Asundi, 2005). The separation between distributed developers creates difficulties in coordination and collaboration (Belanger & Collins, 1998; Carmel & Agarwal, 2001).
- Some OSS projects are developed without concern for the process of accepting or rejecting changes to the software.
- Resource allocation and budgeting are more complex than in proprietary software projects.
- There is higher fluidity in the membership of the development team. OSS developers are not bound to projects by employment relationships and therefore may come and go more often (Stewart, Darcy, & Daniel, 2005).
- Existing effort and cost models for proprietary projects are inadequate for OSS projects, and there is a need to develop new models.
- Commercial proprietary projects generate income and thus enable companies to hire high-quality and motivated programmers. This is not the case in open source projects.

Quality and Security

- OSS programmers are not always enthusiastic about providing and writing documentation, therefore some OSS have inadequate documentation, far below commercial standards.
- Some OSS applications are not sufficiently intuitive and user friendly, and are thus accessible only to very knowledgeable users.
- It appears that there is sometimes a race among many current OSS projects, which often results in rapid releases with the software consequently containing many bugs (Hissam et al., 2001).
- The OSS movement has made the life of cyberterrorists somewhat easier. Since the source code is open and available, cyberterrorists can learn about vulnerabilities in both OSS and proprietary closed source software

(CSS) products. The knowledge that some components of CSS are descendants of similar OSS components, or share the same root code base or the same architecture, design, or specification provides clues as to what attacks could be possible against such software (Hissam et al., 2001).

- There is less variety of applications as compared to proprietary applications.

OSS Business Models

The open source model has a lot to offer the business world. For a company considering adopting an open source strategy, open source needs to be evaluated from a business point of view. It requires being clear on the advantages and disadvantages of open source relative to the traditional proprietary model. There are a number of business models for OSS, all of which assume the absence of traditional software licensing fees. As published by the Open Source Initiative (2006a), there are at least four known business models based on OSS.

1. **Support sellers:** In this model, the software product is effectively given away, but distribution, branding, and after-sales service are sold. This is the model followed by, for example, Red Hat (2006).
2. **Loss leader:** The open source is given away as a loss leader and market positioner for closed software. This is the model followed by Netscape.
3. **Widget frosting:** In this model, a hardware company (for which software is a necessary adjunct but strictly a cost rather than profit center) goes open source in order to get better drivers and cheaper interface tools. Silicon Graphics, for example, supports and ships Samba (2006).
4. **Accessorizing:** This involves selling accessories such as books, compatible hardware, and complete systems with open source software preinstalled. It is easy to trivialize this (open source T-shirts, coffee mugs, Linux penguin dolls), but at least the books and hardware underlie some clear successes: O'Reilly Associates, SSC, and VA Research are among the companies using this model.

So far, the exemplars of commercial success have been service sellers or loss leaders. Nevertheless, there is good reason to believe that the clearest near-term gains in open source will be in widget frosting. For widget makers (such as semiconductor or peripheral-card manufacturers), interface software is not even potentially a revenue source. Therefore, the downside of moving to open source is minimal. (Hecker, 2000, proposes more models potentially usable by companies creating or leveraging OSS products.)

CONCLUSION

OSS is an alternative method of development that makes efficient use of global knowledge. It has captured the attention of academics, software practitioners, and the entire software community. Some OSS products have proven to be as reliable and secure as similar commercial products, and are a viable source of components from which to build OSS and CSS systems. Unfortunately, through OSS products, cyberterrorists also gain additional information about these components and discover vulnerabilities in products based on them.

There are a number of business models for OSS. Software development companies are beginning to support OSS-style development. They tend to try to profit through providing additional value to OSS products, such as value-added software, professional documentation, packaging, and support.

Both the quality and scope of OSS are growing at an increasing rate and there are already free alternatives to many of the fundamental software

tools, utilities, and applications that are able to compete with traditional proprietary software. However, there is still controversy about whether OSS is faster, better, and cheaper than proprietary software. Adopters of OSS should not enter the realm blindly and should know its benefits and pitfalls. Further empirical and theoretical research is needed on developing and managing OSS projects. Identifying and explicitly modeling OSS development processes in forms that can be shared, modified, and redistributed appears to be an important topic for future investigation (Jensen & Scacchi, 2005). The open development process can provide a suitable environment for investigation of software development processes.

LIST OF ACRONYMS

BSD: Berkeley Source Distribution
CSS: Closed source software
FS: Free software
FSF: Free Software Foundation
GNU: GNU Not Unix (recursive acronym)
GPL: General Public License
LGPL: Lesser General Public License
MPL: Mozilla Public License
NPL: Netscape Public License
OSD: Open Source Definition
OSI: Open Source Initiative
OSS: Open source software

REFERENCES

AlMarzouq, M., Zheng, L., Rong, G., & Grover, V. (2005). Open source: Concepts, benefits, and challenges. *Communications of the Association for Information Systems (CAIS), 16*, 756-784.

Apache Software Foundation. (2006). *Apache HTTP server project*. Retrieved January 8, 2006, from http://httpd.apache.org/

Asiri, S. (2003). Open source software. *ACM SIGCAS Computers and Society, 33*(1), 2.

Asundi, J. (2005). The need for effort estimation models for open source software projects. In *Proceedings of the Fifth Workshop on Open Source Software Engineering (5-WOSSE),* 1-3.

Behlendorf, B. (1999). Open source as a business strategy. In M. Stone, S. Ockman, & C. DiBona (Eds.), *Open sources: Voices from the open source revolution* (pp. 149-170). Sebastopol, CA: O'Reilly & Associates.

Belanger, F., & Collins, R. W. (1998). Distributed work arrangements: A research framework. *The Information Society, 14*(2), 137-152.

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software, 18*(2), 22-29.

DiBona, C., Ockman, S., & Stone, M. (Eds.). (1999). *Open sources: Voices from the open source revolution*. Sebastapol, CA: O'Reilly and Associates.

Eunice, J. (1998). *Beyond the cathedral, beyond the bazaar*. Retrieved January 10, 2006, from http://www.illuminata.com/public/all/catalog.cgi/cathedral

Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development*. London: Addison Wesley.

Free Software Foundation. (1991). *GNU general public license, version 2*. Retrieved January 8, 2006, from http://www.gnu.org/licenses/gpl.html

Free Software Foundation. (2006). *Definition of free software*. Retrieved January 8, 2006, from http://www.fsf.org

Frishberg, N., Dirks, A. M., Benson, C., Nickell, S., & Smith, S. (2002). Getting to know you: Open source development meets usability. In *Extended Abstracts of the Conference on Human Factors in Computer Systems (CHI 2002)* (pp. 932-933).

Hecker, F. (2000). *Setting up shop: The business of open source software.* Retrieved May 31, 2006, from http://www.hecker.org/writings/setting-up-shop.html

Hissam, S., Weinstock, C. B., Plakosh, D., & Asundi, J. (2001). *Perspectives on open source software* (Tech. Rep. No. CMU/SEI-2001-TR-019). Retrieved January 10, 2006, from http://www.sei.cmu.edu/publications/documents/01.reports/01tr019.html

Jensen, C., & Scacchi, W. (2005, May 27). Experiences in discovering, modeling, and reenacting open source software development processes. In M. Li, B. W. Boehm, & L. J. Osterweil (Eds.), *Unifying the software process spectrum, ISPW 2005*, Beijing, China (LNCS Vol. 3840, pp. 449-462). Berlin, Germany: Springer-Verlag.

Johnson, K. (2001). *Open source software development.* Retrieved January 8, 2006, from http://chinese-school.netfirms.com/computer-article-open source.html

Lawrie, T., & Gacek, C. (2002). Issues of dependability in open source software development. *Software Engineering Notes (SIGSOFT), 27*(3), 34-37.

Lerner, J., & Tirole, J. (2001). The open source movement: Key research questions. *European Economic Review, 45*(4-6), 819-826.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics, 46*(2), 125-156.

Linux Online. (2006). *Linux.* Retrieved January 8, 2006, from http://www.linux.org/

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000). A case study of open source software development: The Apache server. In *Proceedings of the 22nd International Conference on Software Engineering* (pp. 263-272).

Mozilla. (2006). *Mozilla.* Retrieved January 8, 2006, from http://www.mozilla.org/

Nichols, D. M., Thomson, K., & Yeates, S. A. (2001). Usability and open source software development. In *Proceedings of the Symposium on Computer Human Interaction* (pp. 49-54).

Nichols, D. M., & Twidale, M. B. (2003). The usability of open source software. *First Monday, 8*(1). Retrieved from http://firstmonday.org/issues/issue8_1/nichols/index.html

OpenOffice. (2006). *OpenOffice.* Retrieved January 10, 2006, from http://www.openoffice.org/

Open Source Initiative. (2005). *The open source definition.* Retrieved January 8, 2006, from http://www.opensource.org/docs/definition.php

Open Source Initiative. (2006a). *Open source case for business.* Retrieved May 31, 2006, from http://www.opensource.org/advocacy/case_for_business.php

Open Source Initiative. (2006b). *OSI certification mark and program.* Retrieved January 8, 2006, from http://www.opensource.org/docs/certification_mark.php

Perens, B. (1999). The open source definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (1st ed., pp. 171-188). Sebastopol, CA: O'Reilly and Associates.

Raymond, E. S. (1998). *The cathedral and the bazaar.* Retrieved January 8, 2006, from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Raymond, E. S. (1999). The revenge of the hackers. In M. Stone, S. Ockman, & C. DiBona (Eds.), *Open sources: Voices from the open source revolution* (pp. 207-219). Sebastopol, CA: O'Reilly & Associates.

Red Hat. (2006). *Red Hat.* Retrieved May 31, 2006, from http://www.redhat.com/

Reinke, J., & Saiedian, H. (2003). The availability of source code in relation to timely response to

security vulnerabilities. *Computers & Security, 22*(8), 707-724.

Reis, C. R., & Fortes, R. P. d. M. (2002). An overview of the software engineering process and tools in the Mozilla project. In C. Gacek & B. Arief (Eds.), *Proceedings of the Open Source Software Development Workshop* (pp. 155-175).

Samba. (2006). *Samba*. Retrieved May 31, 2006, from http://www.sgi.com/products/software/samba/

Sendmail Consortium. (2006). *Sendmail™*. Retrieved January 10, 2006, from http://www.sendmail.org/

Stallman, R. (1999). The GNU operating system and the free software movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 53-70). Sebastopol, CA: O'Reilly & Associates.

Stewart, K. J., Darcy, D. P., & Daniel, S. L. (2005). Observations on patterns of development in open source software projects. In *Proceedings of the Fifth Workshop on Open Source Software Engineering (5-WOSSE)* (pp. 1-5).

Wheeler, D. A. (2005). *Why open source software/free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers!* Retrieved January 10, 2006, from http://www.dwheeler.com/oss_fs_why.html

Wong, K., & Sayo, P. (2004). *Free/open source software: A general introduction.* UNDP, Asia-Pacific Development Information Programme. Retrieved January 10, 2006, from http://www.iosn.net/downloads/foss_primer_print_covers.pdf

Zaritski, R. M. (2003). Using open source software for scientific simulations, data visualization, and publishing. *Journal of Computing Sciences in Colleges, 19*(2), 218-222.

KEY TERMS

Closed Source Software (CSS): Non-OSS for which the source code is not available and not open. It is closed to modification and distribution by licenses that explicitly forbid it. The term CSS is typically used to contrast OSS with proprietary software.

Copyleft: Permission for everyone to run, copy, and modify the program, and to distribute modified versions, but no permission to add restrictions of one's own.

Free Software (FS): Free relates to liberty and not to price. It is similar to OSS but differs in the scope of the license. FS does not accept selective open sourcing in which companies may elect to make publicly available specific components of the source code instead of the entire code.

General Public License (GPL): License that permits the redistribution and reuse of source code for unfettered use and access as long as any modifications are also available in the source code and subject to the same license.

Open Source Software (OSS): Software for which the source code is open and available. Its licenses give users the freedom to access and use the source code for any purpose, to adapt and modify it, and to redistribute the original or the modified source code for further use, modification, and redistribution.

Proprietary Software (PS): Software produced and owned by individuals or companies, usually with no provision to users to access to the source code, and licensed to users under restricted licenses in which the software cannot be redistributed to other users. Some proprietary software comes with source code—users are free to use and modify the software, but are restricted by licenses to redistribute modifications or simply share the software.

Source Code: The original human-readable version of a program, written in a particular programming language. In order to run the program, the source code is compiled into object code, a machine-readable binary form.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 184-196, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.6
Open Source Software Evaluation

Karin van den Berg
FreelancePHP, The Netherlands

ABSTRACT

If a person or corporation decides to use open source software for a certain purpose, nowadays the choice in software is large and still growing. In order to choose the right software package for the intended purpose, one will need to have insight and evaluate the software package choices. This chapter provides an insight into open source software and its development to those who wish to evaluate it. Using existing literature on open source software evaluation, a list of nine evaluation criteria is derived including community, security, license, and documentation. In the second section, these criteria and their relevance for open source software evaluation are explained. Finally, the future of open source software evaluation is discussed.

INTRODUCTION

The open source software market is growing. Corporations large and small are investing in open source software. With this growth comes a need to evaluate this software. Enterprises need something substantial to base their decisions on when selecting a product. More and more literature is being written on the subject, and more will be written in the near future.

This chapter gives an overview of the available open source evaluation models and articles, which is compounded in a list of unique characteristics of open source. These characteristics can be used when evaluating this type of software. For a more in-depth review of this literature and the characteristics, as well as a case study using this information, see van den Berg (2005).

OPEN SOURCE SOFTWARE EVALUATION LITERATURE

The name already tells us something. Open source software is open—not only free to use but free to change. Developers are encouraged to participate in the software's community. Because of this unique process, the openness of it all, there is

far more information available on an open source software package and its development process. This information can be used to get a well-rounded impression of the software. In this chapter we will see how this can be done.

Though the concept of open source (or free software) is hardly new, the software has only in recent years reached the general commercial and private user. The concept of open source evaluation is therefore still rather new. There are a few articles and models on the subject, however, which we will introduce here and discuss more thoroughly in the next section.

Open Source Maturity Models

Two maturity models have been developed specifically for open source software.

The first is the Capgemini Expert Letter open source maturity model (Duijnhouwer & Widdows, 2003). The model "allows you to determine if or which open source product is suitable using just seven clear steps." Duijnhouwer and Widdows first explain the usefulness of a maturity model, then discuss open source product indicators and use these in the model. The model steps start with product research and rough selection, then uses the product indicators to score the product and determine the importance of the indicators, combining these to make scorecards. Finally it ends with evaluation.

Second, there is the Navica open source maturity model, which is used in the book *Succeeding with Open Source* (Golden, 2005). This model uses six product elements in three phases: assessing element maturity, assigning weight factors, and calculating the product maturity score.

Open Source Software Evaluation Articles

Aside from the two models, a number of articles on open source software evaluation have been written.

Crowston et al. (2003) and Crowston, Annabi, Howison, and Masango (2004) have published articles in the process of researching open source software success factors. In these articles, they attempt to determine which factors contribute to the success of open source software packages.

Wheeler's (n.d.) *How to Evaluate Open Source/Free Software (OSS/FS) Programs* defines a number of criteria to use in the evaluation of open source software, as well as a description of the recommended process of evaluation. Wheeler continues to update this online article to include relevant new information.

Another article defining evaluation criteria for open source software is *Ten Rules for Evaluating Open Source Software* (Donham, 2004). This is a point-of-view paper from Collaborative Consulting, providing 10 guidelines for evaluating open source software.

Finally, Nijdam (2003), in a Dutch article entitled "Vijf Adviezen voor Selectie van OSS-Componenten" ("Five Recommendations for Selection of OSS Components"), gives recommendations based on his own experience with selecting an open source system.

Literature Summary

Table 1 summarizes the criteria derived from the literature mentioned in the previous two sections and how they are discussed.

EVALUATING OPEN SOURCE SOFTWARE

The open source software market is in some ways very different from the traditional software market. One of the differences is that there is an abundance of information available concerning the software and its development process that is in most cases not available for traditional software.

The evaluation of traditional software is usually focused on the functionality and license cost of the software. In the open source world, the

Table 1.

Criterion	Duijnhouwer and Widdows (2003)	Golden (2005)	Crowston et al. (2004)	Wheeler, (2005)	Donham (2004)	Nijdam (2003)
Community	Y	Y	Team size and activity level	In support	-	Active groups
Release Activity	-	Activity level	Activity level	Maintenance	-	Active groups
Longevity	Age	Y	-	Y	Maturity	Version
License	Y	In risk	-	Y	Y	Y
Support	Y	Y	-	Y	Y	-
Documentation	In ease of deployment	Y	-	In support	Y	-
Security	Y	In risk	-	Y	Y	-
Functionality	Features in time	Y	-	Y	Y	Y
Integration	Y	Y	-	In functionality	In infrastructure	-

evaluation includes information from a number of other resources, giving a well-rounded picture of the software, its development, and its future prospects.

Using the existing evaluation models and articles discussed in the previous section, an overview is given here of the characteristics of open source software relevant to software evaluation and the information available on an open source software project concerning these characteristics.

Community

According to Golden (2005, p. 21), "One of the most important aspects of open source is the community."

The user community for most open source projects is the largest resource available. The community provides developers, user feedback, and ideas, and drives the project team. An active community helps the project move forward. It also shows the level of interest in the project, which can provide a measurement of quality and compliance with user requirements. A well-provided-for community also shows the team's interest in the user, allows the user to participate, and gives voice to the user's wishes and requirements.

The user community of an open source project consists of the people that use the software and participate in some way, from answering user questions to reporting bugs and feature requests. Users in the community sometimes cross the line into the developer community, which is often a line made very thin by encouraging participation and making the developer community accessible to anyone who is interested. In some cases, the user and developer community interact fully in the same discussion areas.

The community of an open source project is very important because it is the community that does most of the testing and provides quality feedback. Instead of using financial resources to put the software through extensive testing and quality assurance (QA), like a proprietary vendor will do, the open source projects have the community as a resource. The more people that are interested in a project, the more likely it is that it will be active and keep going. A large and active community says something about the acceptance of the software. If the software was not good enough to use, there would not be so many people who cared about its development (Duijnhouwer & Widdows, 2003).

The community is mostly visible in terms of the following (Crowston et al., 2004; Duijnhouwer & Widdows, 2003; Golden, 2005; Nijdam, 2003):

- **Posts:** Number of posts per period and number of topics
- **Users:** Number of users and the user-developer ratio in terms of the number of people and number of posts; if only users post, the developers are not as involved as they should be
- **Response time:** If and how soon user questions are answered
- **Quality:** The quality of posts and replies; are questions answered to the point, and are the answers very short or more elaborate? Is there much discussion about changes and feature additions?
- **Friendliness:** How friendly members are toward each other, especially to newcomers, also known as "newbies"; the community should have an open feel to it, encouraging people to participate

The depth of conversations, as mentioned in the fourth item, gives a good impression of how involved the community is with the ongoing development of the project. Much discussion about the software, in a friendly and constructive manner, encourages the developers to enhance the software further. The community activity is also reflected in other areas such as support and documentation.

Release Activity

The activity level of a project consists of the community activity and the development activity. The community was discussed above. The development activity is reflected in two parts:

- The developer's participation in the community
- The development itself—writing or changing the source code

The latter activity is visible mostly in the release activity. All software projects release new versions after a period of time. The number of releases per period and their significance, meaning how large the changes are per release (i.e., are there feature additions or just bug fixes in the release), illustrates the progress made by the developers. This gives a good indication of how seriously the developers are working on the software.

The open source repositories SourceForge[1] and FreshMeat[2], where project members can

share files with the public, provide information that could be useful to evaluate the release activity (Wheeler, n.d.).

An open source project often has different types of releases:

- **Stable releases:** These are the most important type for the end user. They are the versions of software that are deemed suitable for production use with minimal risk of failure.
- **Development versions:** These can have different forms, such as beta, daily builds, or CVS (Concurrent Version System) versions, each more up to date with the latest changes. These versions are usually said to be used "at your own risk" and are not meant for production use because there is a higher possibility of errors. A project that releases new versions of software usually publishes release notes along with the download that list all the changes made in the software since the previous release. Other than the release notes, the project might also have a road map, which usually shows what goals the developers have, how much of these goals are completed, and when the deadline or estimated delivery date is for each goal. Checking how the developers keep up with

this road map shows something about how well the development team can keep to a schedule.

Though a project might stabilise over time as it is completed, no project should be completely static. It is important that it is maintained and will remain maintained in the future (Wheeler, n.d.).

The project's change log can give the following information (Chavan, 2005):

- **The number of releases made per period of time:** Most projects will make several releases in a year, sometimes once or twice a month. A year is usually a good period in which to count the releases.
- **The significance of each release:** The change log or release notes explain what has changed in the release. These descriptions are sometimes very elaborate, where every little detail is described, and sometimes very short, where just large changes are listed. A good distinction to make is whether the release only contains bug fixes or also contains enhancements to features or completely new features. One thing to keep in mind here is that fewer, more significant releases is in most cases better than a large number of less significant releases leading to the same amount of change over time since the users will have to upgrade to new versions each time a release is made, which is not very user friendly. There should be a good balance between the number of releases and the releases' significance. If the project is listed on SourceForge and/or FreshMeat, some of the release activity information is available there.

Longevity

The longevity of a product is a measure of how long it has been around. It says something about a project's stability and chance of survival. A project that is just starting is usually still full of bugs (Golden, 2005). The older a project, the less likely the developers will suddenly stop (Duijnhouwer & Widdows, 2003). However, age is not always a guarantee of survival. First of all, very old software may be stuck on old technologies and methods, from which the only escape is to completely start over. Some software has already successfully gone through such a cycle, which is a good sign in terms of maturity. One thing that needs to be taken into account when products are not very young is whether or not there is still an active community around it.

The age and activity level of a project are often related. Young projects often have a higher activity level than older ones because once a project has stabilised and is satisfactory to most users, the discussions are less frequent and releases are smaller, containing mostly bug and security fixes. This does not mean that the activity should ever be slim to none. As mentioned before, no project is ever static (Wheeler, n.d.). There is always something that still needs to be done.

Longevity is checked using the following criteria (Golden, 2005; Nijdam, 2003):

- **Age of the product:** The date of the first release
- **Version number:** A 0.x number usually means the developers do not think the software is complete or ready for production use at this time.

If the project is very old, it is worthwhile to check if it has gone through a cycle of redesign, or if it is currently having problems with new technology.

Keep in mind that the version number does not always tell the whole story. Some projects might go from 1.0 to 2.0 with the same amount of change that another project has to go from 1.0 to 1.1. The fast progression of the version number might be used to create a false sense of progress.

Other software products are still in a 0.x version even after a long time and after they are proved suitable for production use (Nijdam, 2003).

License

The licenses in the open source world reflect something of the culture. The most important term in this context is "copyleft," introduced by Richard Stallman, which means that the copyright is used to ensure free software and free derivative works based on the software (Weber, 2004). In essence, a copyleft license obligates anyone who redistributes software under that license in any way or form to also keep the code and any derivative code under the license, thus making any derivatives open source as well.

The most well-known example of a copyleft license is the GNU GPL (General Public License; Weber, 2004). This is also one of the most used licenses. On SourceForge, a large open source public repository where over 62,000 projects reside, almost 70%[3] of projects use the GNU GPL as their license. There are some large and well-known products that do not use SourceForge, and some of these have their own license, such as Apache, PHP, and Mozilla (Open Source Initiative [OSI], 2005).

Because copyleft in the GNU GPL is very strong, an additional version was made called the LGPL (library GPL, also known as lesser GPL), which is less restrictive in its copyleft statements, allowing libraries to be used in other applications without the need to distribute the source code (Weber).

A non-copyleft license that is much heard of is the BSD (Berkeley source distribution) license. It has been the subject of much controversy and has had different versions because of that. Components that are licensed under the BSD are used in several commercial software applications, among which are Microsoft products and Mac OS X (Wikipedia, 2005a). The license of the software in use can have unwanted consequences depending on the goal of the use. If the user plans to alter and redistribute the software in some way but does not want to distribute the source code, a copyleft license is not suitable. In most cases, however, the user will probably just want to use the software, perhaps alter it to the environment somewhat, but not sell it. In that case, the license itself should at least be OSI approved and preferably well known. The license should fit with the intended software use.

As just mentioned, the license should preferably be an OSI-approved license. If it uses one of the public licenses, the better known the license, the more can be found on its use and potential issues (Wheeler, n.d.).

Support

There are two types of support for a software product:

- **Usage support:** The answering of questions on the installation and use of the software
- **Failure support or maintenance:** The solving of problems in the software

Often, the two get mixed at some level because users do not always know the right way to use the product. Their support request will start as a problem report and later becomes part of usage support (Golden, 2005).

The way support is handled is a measure of how seriously the developers work on the software (Duijnhouwer & Widdows, 2003). One way to check this is to see if there is a separate bug tracker[4] for the software and how actively it is being used by both the developers and the users. When the developers use it but hardly any users seem to participate, the users may not be pointed in the right direction to report problems. Aside from community support, larger or more popular projects may have paid support options. The software is free to use, but the user has the option to get professional support for a fee, either on a

service-agreement basis where a subscription fee is paid for a certain period of time, or a per-incident fee for each time the user calls on support. The project leaders themselves may offer something like this, which is the case for the very popular open source database server MySQL (2005).

There are companies that offer specialised support for certain open source software. This is called third-party support. For example, at the Mozilla support Web page, it can be seen that DecisionOne offers paid support for Mozilla's popular Web browser FireFox, the e-mail client Thunderbird, and the Mozilla Suite (Mozilla, 2005). The fact that paid support exists for an open source product, especially third-party support, is a sign of maturity and a sign the product is taken seriously.

Support for open source software is in most cases handled by the community. The community's support areas are invaluable resources for solving problems (Golden, 2005). Mature products often have paid support options as well if more help or the security of a support contract is required.

Community Support

The usage support is usually found in the community. Things to look for include the following (Golden, 2005):

- Does the program have a separate forum or group for asking installation- and usage-related questions?
- How active is this forum?
- Are developers participating?
- Are questions answered adequately?
- Is there adequate documentation (see the documentation section)?

Responses to questions should be to the point and the responders friendly and helpful. In the process of evaluating software, the evaluator will probably be able to post a question. Try to keep to the etiquette, where the most important rule is to search for a possible answer on the forum before posting a question and to given enough relevant information for others to reproduce the problem (Golden, 2005; Wheeler, n.d.).

The way the community is organised influences the community support's effectiveness. A large project should have multiple areas for each part of the project, but the areas should not be spread to thin. That way, the developers that are responsible for a certain part of the project are able to focus on the relevant area without getting overwhelmed with a large amount of other questions. If the areas are too specialised and little activity takes place in each, not enough people will show interest and questions are more likely to remain unanswered.

Failure support within the project is often handled by a bug tracker by which problems are reported and tracked. Statistical studies have shown that in successful projects, the number of developers that fix bugs in open source software is usually much higher than the number of developers creating new code (Mockus, Rielding, & Herbsleb, 2000).

Paid Support

Paid support might be available from the project team itself (Golden, 2005). There may have been people who have given their opinion about the quality of this support.

One of the strong signs of the maturity of open source software is the availability of third-party support: companies that offer commercial support services for open source products (Duijnhouwer & Widdows, 2003). Some companies offer service contracts, others offer only phone support on a per-incident basis. Check for paid support options whether they will be used or not (Duijnhouwer & Widdows). How the situation may be during actual use of the software is not always clear and it can give a better impression of the maturity of the software.

Documentation

There are two main types of documentation (Erenkratz & Taylor, 2003):

- User documentation
- Developer documentation

User documentation contains all documents that describe how to use the system. For certain applications, there can be different levels in the user documentation, corresponding with different user levels and rights. For example, many applications that have an administrator role have a separate piece of documentation for administrators. Additionally, there can be various user-contributed tutorials and how-tos, be it on the project's Web site or elsewhere. The available documentation should be adequate for your needs. The more complex the software, the more you may need to rely on the user documentation.

The other main type of documentation, which plays a much larger role in open source software than in proprietary applications, is developer documentation. A voluntary decentralised distribution of labour could not work without it (Weber, 2004). The developer documentation concerns separate documents on how to add or change the code, as well as documentation within the source code by way of comments. The comments usually explain what a section of code does, how to use and change it, and why it works like it does. Though this type of documentation may exist for proprietary software, it is usually not public.

If it is possible that you may want to change or add to the source code, this documentation is very valuable. A programmer or at least someone with some experience in programming will be better able to evaluate whether this documentation is set up well, especially by the comments in the source code. It is a good idea to let someone with experience take a look at this documentation (n.d., 2005).

A third type of documentation that is often available for larger server-based applications is maintainer documentation, which includes the install and upgrade instructions. These need to be clear, with the required infrastructure and the steps for installing the software properly explained. This documentation is needed to set up the application. For this type, again, the complexity of the application and its deployment determines the level of documentation that is needed. Documentation is often lagging behind the status of the application since it is often written only after functionality is created, especially user documentation (Scacchi, 2002). It is a good idea to check how often the documentation is updated, and how much the documentation is behind compared to the current status of the software itself.

The documentation for larger projects is often handled by a documentation team. A discussion area may exist about the documentation, giving an indication of the activity level of that team.

Security

Security in software, especially when discussing open source software, has two sides to it. There are people who believe security by obscurity is better, meaning that the inner workings of the software are hidden by keeping it closed source, something that open source obviously does not do. The advocates of security by obscurity see the openness of open source software as a security hazard. Others argue that the openness of open source actually makes it safer because vulnerabilities in the code are found sooner. Open source software gives both attackers and defenders great power over system security (Cowan, 2003; Hoepman & Jacobs, 2005).

Security depends strongly on how much attention the developers give to it. The quality of the code has much to do with it, and that goes for both proprietary and open source software. If the code of proprietary software is not secure, the vulnerabilities may still be found. There are

plenty of examples where this occurs, such as the Microsoft Windows operating system (OS). The vulnerabilities are often found by hackers who try to break the software, sometimes by blunt force or simple trial and error. In this case, a vulnerability might get exploited before the vendor knows about it. The attack is the first clue in that case. The open source software's vulnerabilities, however, could be found by one of the developers or users just by reviewing the code; he or she can report the problem so it can be fixed (Payne, 2002). It is important that the developers take the security of their software seriously and respond swiftly to any reported vulnerabilities.

There are various security advisories to check for bugs in all types of software that make it vulnerable to attacks. A couple of well-known advisories are http://www.securityfocus.com and http://www.secunia.com. Keep in mind that more popular software will have a higher chance of having vulnerability reports, so the mere lack of reports is no proof of its security. On the project's Web site, it can be seen, for instance in the release notes, how serious the project is about security.

Functionality

Though functionality comparison is not specific to open source software evaluation and is properly covered in most traditional software evaluation models, there are some points to take into consideration. Open source software often uses the method described by the phrase "release early and often" (Raymond, 1998). This method enables faster error correction (Weber, 2004) by keeping the software up to date as much as possible. It also encourages people to contribute because they see the result of their work in the next release much sooner (Raymond). However, this often means that the software is incomplete during the first releases, at least more so than is customary with proprietary software. Where vendors of proprietary software will offer full functionality descriptions for their software, open source projects might not have the complete information on the Web site (Golden, 2005). Just like with documentation, the information on the Web site might be lagging behind the actual functionality. Other means of checking the current functionality set might be needed. Fortunately, open source software that is freely available gives the added option of installing the software to enable the full testing of the functionality, an option that is mostly not available with proprietary software, for which at most only limited versions, in terms of functionality or time, are given freely for trying it out.

One problem with open source projects is that the documentation is not always up to date with the latest software. Look beyond the feature list on the Web site to find out what features the software has. Two options are to query the developers and ask the user community (Golden, 2005). Eventually the software itself should be investigated. If it is a Web-based application, an online demo might be available, though installing it on a test environment could be useful because it also gives insight on how well the software installs.

A list of functional requirements for the goal of the software can be used to check if the needed functionality is available. If such a list is not given, there may be one available from technology analyst organisations (Golden, 2005). It is wise to make a distinction in the list between features that are absolutely necessary, where the absence would lead to elimination, and those that would be a plus, which results in a higher score. If there is something missing, there is always the option to build it or have it built.

When comparing functionality, those features that are part of the functional requirements should take priority, but additional features may prove useful later. The features used or requested by the users in the future are not really predictable. While evaluating the software, features may be found in some of the candidates that are very useful for the goal. These can be added to the functional requirements.

Part of the functionality is localisation. The languages to which the interface and documentation are translated are a sign of the global interest taken in the software.

Integration

Duijnhouwer and Widdows (2003) mention three integration criteria. These are most important for software that is being used in collaboration with other software, and for people who are planning on adapting the software to their use, such as adding functionality or customising certain aspects so that it fits better in the organisation's environment. The three criteria are discussed in the next three subsections.

Modularity

Modularity of software means that the software or part of the software is broken into separate pieces, each with its own function. This type of structure has the following advantages:

- Modular software is easier to manage (Garzarelli, 2002; Mockus, Fielding, & Herbsleb, 2002).
- With a base structure that handles the modules well, people can easily add customised functionality without touching the core software.
- Modular software enables the selection of the needed functionality, leaving out those that are not necessary for the intended use. This way, the software can be customised without the need for a programmer.
- Modular software can be used in commercial applications. By making software modular, not everything needs to be given away as open source. It is can be used to give away only parts of software as open source while the add-on modules are sold as proprietary software (Duijnhouwer & Widdows, 2003). This is also called the razor model, as in giving away the razor for free and charging for the blade (Golden, 2005).

Evidence of a modular structure can often be found in several places, such as the source code, the developer documentation, or the download section, where modules might be available for download separate from the core software.

Standards

In the software market, more and more open standards emerge to make cooperation between software easier (Golden, 2005). If the software vendors use these standards in their software, it makes it easier to communicate between different software packages, and to switch between software packages. In some industries, standards are far more important than in others. For some software, there may not even be an applicable standard.

The use of current and open standards in open source software is a sign of the software's maturity (Duijnhouwer & Widdows, 2003). The feature list of the software usually lists what standards are used and with which the software complies.

Collaboration with Other Products

Closely connected to standards is the collaboration with other products. As mentioned before, not every software type has applicable standards, and sometimes the formal standards are not used as much as other formats. Examples of such formats are the Microsoft Word document format, and Adobe's PDF (portable document format). The office suite OpenOffice.org (2005) has built-in compatibility for both formats.

Software Requirements

Most software is written for a specific OS, for example, Microsoft Windows or Linux (Wheeler, n.d.). Certain types of software also rely on other

software, such as a Web server or a database. The requirements of the software will state which software and which versions of that software are compatible. If these requirements are very specific, it could lead to problems if they are incompatible with the organisation's current environment.

THE FUTURE OF OPEN SOURCE SOFTWARE EVALUATION

Open Source Software Evaluation Literature

More is being written on open source software evaluation at the time of writing. For example, another model called the business readiness rating (OpenBRR, 2005), aimed at open source software, was released recently. The research of Crowston and others is still ongoing, so there will be more results in the near future to include in the open source software evaluation process. Given how recent the rest of the literature discussed in this chapter is, it is likely that more will be published on the subject in the next few years.

The Future of Open Source Software

Open source software is being used increasingly by corporations worldwide. There is now some literature available to help with the evaluation of open source software, and the number of articles and models is increasing. With this growth in the field comes more attention from companies, especially on the enterprise level, which will cause more demand for solid evaluation models. Because open source software and the process around it provide much more information than traditional software, there is certainly a need for such models.

This literature will help justify and solidify the position of open source software evaluation in a corporate setting, giving more incentive to use open source software. Most likely, more companies will be investing time and money in its development, like we are seeing today in examples such as Oracle investing in PHP and incorporating this open source Web development language in its products (Oracle, 2005), and Novell's acquisition of SUSE Linux (Novell, 2003). The open source software evaluation literature can help IT managers in adopting open source.

CONCLUSION

The field of open source software evaluation is growing, and with that growth more attention is gained from the large enterprises. With this attention comes more demand for evaluation models that can be performed for these corporations, which will give more growth to the open source software market as well. In this chapter, an overview is given of the current literature and the criteria derived from that literature that can be used in open source software evaluation. For each of the criteria—community, release activity, longevity, license, support, documentation, security, and functionality—this chapter explains why it is important in the market and what to do to evaluate it. This information can be used on its own or in conjunction with more traditional evaluation models and additional information referenced here by companies and individuals that wish to evaluate and select an open source software package. It helps to give insight into the open source software sector.

REFERENCES

Chavan, A. (2005). Seven criteria for evaluating open source content management systems. *Linux Journal*. Retrieved August 9, 2005, from http://www.linuxjournal.com/node/8301/

Cowan, C. (2003). Software security for open source systems. *Security & Privacy Magazine*, *1*(1), 38-45.

Crowston, K., Annabi, H., & Howison, J. (2003). Defining open source software project success. In *Twenty-Fourth International Conference on Information Systems, International Conference on Software Engineering (ICIS 2003)* (pp. 29-33). Retrieved March 30, 2005, from http://opensource.mit.edu/papers/crowstonannabihowison.pdf

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Towards a portfolio of FLOSS project success measures. *Collaboration, Conflict and Control: The Fourth Workshop on Open Source Software Engineering, International Conference on Software Engineering (ICSE 2004)*, 29-33. Retrieved March 30, 2005, from http://opensource.ucc.ie/icse2004/Workshop_on_OSS_Engineering_2004.pdf

Donham, P. (2004). Ten rules for evaluating open source software. *Collaborative Consulting.* Retrieved August 8, 2005, from http://www.collaborative.ws/leadership.php?subsection=27

Duijnhouwer, F., & Widdows, C. (2003). *Capgemini open source maturity model.* Retrieved February 12, 2006, from http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf

Erenkratz, J. R., & Taylor, R. N. (2003). *Supporting distributed and decentralized projects: Drawing lessons from the open source community* (Tech. Rep.). Institute for Software Research. Retrieved August 9, 2005, from http://www.erenkrantz.com/Geeks/Research/Publications/Open-Source-Process-OSIC.pdf

Garzarelli, G. (2002, June 6-8). *The pure convergence of knowledge and rights in economic organization: The case of open source software development.* Paper presented at the DRUID Summer Conference 2002 on Industrial dynamics of the new and old economy—Who embraces whom?, Copenhagen.

Golden, G. (2005). *Succeeding with open source.* Boston: Addison-Wesley Pearson Education.

Hoepman, J., & Jacobs, B. (2005). *Software security through open source* (Tech. Rep.). Institute for Computing and Information Sciences, Radboud University Nijmegen. Retrieved August 9, 2005, from http://www.cs.ru.nl/~jhh/publications/oss-acm.pdf

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000). A case study of open source software development: The Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000).* Retrieved on March 30, 2005, from http://opensource.mit.edu/papers/mockusapache.pdf

Mockus, A., Fielding, R. T., & Herbsleb, J. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Mozilla. (2005). *Mozilla.org support.* Retrieved February 16, 2005, from http://www.mozilla.org/support/

MySQL. (2005). *MySQL support Web site.* Retrieved February 16, 2005, from http://www.mysql.com/support/premier.html

Nijdam, M. (2003). Vijf adviezen voor selectie van oss-compontenten. *Informatie: Maandbladvoor Informatieverwerking, 45*(7), 28-30.

Novell. (2003). *Novell announces agreement to acquire leading enterprise Linux technology company SUSE LINUX.* Retrieved August 8, 2005, from http://www.novell.com/news/press/archive/2003/11/pr03069.html

OpenBRR. (2005). *Business readiness rating for open source: A proposed open standard to facilitate assessment and adoption of open source software (RFC1).* Retrieved August 10, 2005, from http://www.openbrr.org/docs/BRR_whitepaper_2005RFC1.pdf

OpenOffice.org. (2005). *OpenOffice.org writer product information*. Retrieved August 10, 2005, from http://www.openoffice.org/product/writer.html

Open Source Initiative (OSI). (2005). *Open Source Initiative: Open source licenses*. Retrieved August 9, 2005, from http://opensource.org/licenses/

Oracle. (2005). *Oracle and Zend partner on development and deployment foundation for PHP-based applications*. Retrieved February 12, 2006, from http://www.oracle.com/corporate/press/2005_may/05.16.05_oracle_zend_partner_finalsite.html

Payne, C. (2002). On the security of open source software. *Information Systems Journal*, *12*(1), 61-78.

Raymond, E. S. (1998). The cathedral and the bazaar. *First Monday*, *3*(3). Retrieved March 30, 2005, from http://www.firstmonday.org/issues/issue3_3/raymond/

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. In *IEEE Proceedings: Software*, *149*, 24-29. Retrieved March 30, 2005, from http://www1.ics.uci.edu/wscacchi/Papers/New/Understanding-OS-Requirements.pdf

Van den Berg, K. (2005). *Finding open options: An open source software evaluation model with a case study on course management system*. Unpublished master's thesis, Tilburg University, Tilburg, The Netherlands. Retrieved August 30, 2005, from http://www.karinvandenberg.nl/Thesis.pdf

Weber, S. (2004). *The success of open source*. Cambridge, MA: Harvard University Press.

Wheeler, W. (n.d.). *How to evaluate open source/free software (OSS/FS) programs*. Retrieved February 17, 2005, from http://www.dwheeler.com/oss_fs_eval.html

KEY TERMS

Community: A group of people with shared interests that interact. In case of open source software, the community is the group of developers and users that come together, mostly on a Web site, to discuss, debug, and develop the software.

Documentation: The documents that are associated with a piece of software. There is usually user documentation, in the form of help files, tutorials, and manuals, and there can be developer documentation, such as programming guidelines and documents explaining the structure and workings of the software (source code). In some cases there is administrator documentation, which explains how to install and configure the software. The latter is more important for large pieces of software, where one installation will be used by many users, such as Web applications.

License: An agreement that is attached to the use of a product. In case of software, the software license agreement defines the terms under which you are allowed to use the software. For open source software, there are a number of common licenses, not bound to a specific piece of software, that can be used for almost any type of open source software. These licenses are well known so users and developers usually know the conditions of these licenses.

Maturity Model: Not to be confused with the capability maturity model (CMM), a maturity model as discussed in this chapter is a model that can be used to assess the maturity of a software package, evaluating the software using several criteria.

Software Longevity: The life expectancy of software, measured by various factors among which is its age.

Software Release Activity: The number and significance of releases that are made for a certain software package. A release can be a minor

change such as a bug fix, or a major change such as added functionality.

Software Security: How well a piece of software is built in terms of vulnerabilities and defense against them. Any software will have some type of security hole in it that allows a person, often with hostile intentions, to break into the software and use it for purposes that are unwanted. It is necessary for developers to minimize these holes and fix them if they are discovered. In case of open source software, because the source is public, the users may help in discovery by examining the source code. This, however, also means that a person with hostile intentions can also find these holes by examining the source code. Thus, it is always important to keep a close eye on security.

ENDNOTES

1. http://www.sourceforge.net
2. http://www.freshmeat.net
3. Established using the SourceForge Software Map on April 20, 2005, at http://sourceforge.net/softwaremap/trove_list.php?form_cat=13
4. A bug tracker is an application, often Web based, through which the users can report problems with the software, the developers can assign the bug to someone who will handle it, and the status of the bug can be maintained. Bugzilla is one such package that is often used for this purpose.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 197-210, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.7
On the Role of Public Policies Supporting Free/Open Source Software

Stefano Comino
University of Trento, Italy

Fabio M. Manenti
University of Padua, Italy

Alessandro Rossi
University of Trento, Italy

ABSTRACT

Governments' interest in free/open source software is steadily increasing. Several policies aimed at supporting free/open source software have been taken or are currently under discussion all around the world. In this chapter, we review the basic (economic) rationales for such policy interventions and we present some summary statistics on policies taken within the European countries. We claim that in order to evaluate correctly the consequences of such interventions one has to consider both the role and the administrative level at which such decisions are taken as well as the typology of software that is involved. Moreover, we argue that the level playing field cannot be taken for granted in software markets. Therefore, non-intrusive public policies that currently prevail at the European level in terms, for instance, of the promotion of open standards or in terms of campaigns aimed at informing IT decision-makers, are likely to be welfare enhancing.

INTRODUCTION

Governments' interest in free/open source (F/OS) software is steadily increasing. In Europe, this interest has become visible in the Lisbon Strategy and in the corresponding eEurope Action Plans 2002 and 2005 approved by the European Commission where it has been clearly stated the key role of open source software and open standards in pursuing the general objective of giving all citizens the opportunity to participate in the global information society.[1]

All over the world governments are considering various policies to support F/OS software; these policies go from the provision of "best practices" for the usage of open source to information cam-

paigns aimed at making markets participants aware of all software alternatives, from simple expressions of preference towards F/OS software to large scale adoption of open source solutions in governments' offices and schools.

The role of the public sector in the software market is of primary importance. Governments not only set the legal and regulatory framework where economic agents interact, but they are also big software purchasers;[2] this double role makes governments key players in determining the future evolution of software markets and it is therefore of crucial interest to understand both the motivations and the effects of governments' interventions in this sector.

This chapter critically reviews the main arguments in favor or against public intervention supporting F/OS; we also provide some empirical evidence about the various public interventions that are already in place in Europe. The chapter is structured in three parts: in the first part, we provide a general analytical framework; public interventions may occur at different administrative levels (i.e., from municipalities to national or supra-national level), and they may have different motivations. These complexities have not received enough attention in the previous analyses on public interventions towards F/OS; the aim of this section is to offer a possible taxonomy for governmental policies in the software market and to discuss the many rationales for intervention but also the counterarguments that often have been put forward. In the following section, we present some evidence concerning the main public initiatives in Europe. Rather than focusing on any specific case study, we have collected information from the European IDABC, the program documenting the major initiatives supporting F/OS within the European Union. In this way, we have been able to draw some general considerations on the motivations and the characteristics of governments interventions implemented all across the EU. The subsequent section concludes by bridging the theoretical discussion with the empirical analysis.

We claim that, if one considers that the largest share of the software market is represented by self-developed or customized products, the existing literature has placed too much emphasis on packaged software and arguments against public support of F/OS might be improperly grounded. Moreover, we believe that the level playing field cannot be taken for granted in software markets. Therefore, non-intrusive public policies that currently prevail at the European level in terms, for instance, of the promotion of open standards or in terms of campaigns aimed at informing IT decision makers, are likely to be welfare enhancing.

BACKGROUND: A GENERAL FRAMEWORK

It is useful to start our analysis by providing a general framework for discriminating the large heterogeneity of public interventions in the software market. In particular, we claim that, in order to judge correctly rationales, motivations, and consequences of public interventions, it is important to distinguish between the various roles played by policy makers and the various categories of software involved. We argue that many existing contributions, both in the scholarly and in the practitioners' debate, have not clearly taken into account these distinctions.

Public administrations, institutions, and governments play a double role in the software industry. On the one side, being big spenders for software licenses and software development, their adoption/use decisions represent a significant share of the demand thus having a major impact on market equilibrium. On the other side, by acting as legislators and regulators, governments do in various ways determine the evolution of the market; for instance, it is quite evident that the legislation towards intellectual property rights, either based on strong patent protection as in the U.S. or on weaker copyright legislations as it is within the EU, has a major influence on the functioning of

the market and the diverging experiences on the two sides of the Atlantic stand as a clear example of this role. Similarly, as we discuss later in the chapter, governments frequently intervene mandating the adoption of open standards/interfaces; these policies are usually aimed at promoting compatibility and interoperability between different software platforms, thus creating a level playing field between different competitors; this kind of intervention clearly affects the efficiency of the market and therefore suggests a regulatory intention of the proponents.[3]

Obviously, it is often difficult to disentangle interventions of public authorities as adopters/users from those motivated on regulatory scopes; being large users, the decision to adopt a certain software package taken by public bodies affects the dynamic evolution of the industry and the equilibrium outcome, thus having regulatory consequences on the overall functioning of the market.

Irrespectively of the role played by a public administration, interventions may produce different consequences depending on the nature of the product involved. Software is not a commodity and the industry is extremely heterogeneous; indeed, the vast majority of software is either self-developed or custom while packaged software represents a minor share of the market.[4] The structure, the players, and the dynamics of mass-market and custom segments of the software industry are very different as well as different are likely to be the effects induced by the various public interventions.

In Table 1 we provide four examples of interventions distinguishing among different roles of public administrations and different typologies of software: three of these interventions are directly related to the promotion of F/OS, while the fourth refers to the well-known Microsoft European antitrust case. This last example relates to the F/OS world since, as a consequence of the antitrust action, Microsoft has recently announced its decision to allow access to some parts of the source code of its operating system.[5]

Rationales for Intervention: Review of the Literature

The literature on F/OS software in public administrations is quite substantial. Supporters of F/OS software have mainly focused on adoption of such technologies in the public sector and have based their arguments on technical, cost-efficiency or political-idealistic grounds. Regulatory scopes and therefore those rationales based on the consequences of F/OS public adoption on the overall functioning of the market have been receiving a much more limited attention by this stream of research.

Conversely, most of the critical voices in this

Table 1. Examples of public interventions supporting F/OS or regulating the market[10]

	Adoption/Development	Market Regulation
Custom	August 2005: the French Ministry of Foreign Affairs starts developing an open source architecture in order to integrate its computing system.[6]	October 2004: the Belgian administration published its white book concerning the use of open standards and open specifications for public sector purchased software.[7]
Packaged	September 2004: the Education Council of Castilla - La Mancha signed an agreement with Sun Microsystems to distribute Star Office 6.0 to the region's schools.[8]	EU's 2004 antitrust decision: Microsoft is required to disclose complete and accurate interface documentation which would allow non-Microsoft work group servers to achieve full interoperability with Windows PCs and servers. This will enable rival vendors to develop products that can compete on a level playing field in the work group server operating system market.[9]

debate have warned against detrimental consequences of both direct support/intervention and adoption of F/OS by public administrations on market performance.

In what follows, we briefly summarize the debate on F/OS software in the public sector; we devote the first subsection to provide a general overview of the most frequent motivations that have been proposed to justify public support towards open source. In the second subsection we look at the issue from a more critical viewpoint and we present the (often) skeptical view held by some economists and closed source practitioners.

Why Supporting F/OS?

Advocates of the F/OS movement put forward several rationales for public policies in the software market. Leaving aside pure idealistic-philosophical motives,[11] governments should support F/OS because of its intrinsic superiority with respect to closed source software. F/OS is considered to outperform proprietary software in terms of, for instance, higher reliability, security, flexibility, and maintainability of the code.[12] These superior features stem both from the organizational mode of F/OS which is characterized by the presence of a community of developers that continuously reviews the source code and fixes possible bugs, as well as from the fact that the availability of the source code makes it possible for the user to adapt the software to her/his own personal needs and to check every possible defect. Cost-efficiency is a second common rationale for policy interventions which is especially important for those public administrations that are pressured by budget concerns. The public sector would benefit from F/OS because of a number of reasons: net savings due to the reduced or non-existing licensing fees, the opportunity of freely contracting with software developers for subsequent code maintenance/upgrade without being locked into the relationship with the initial provider, or the possibility of profiting from economies of reuse/collaborative development.[13] Similarly, a further beneficial effect would follow from a more efficient employment of public resources that would be shifted from license costs towards human capital investments.

With respect to the issue of innovation dynamics in the software industry, F/OS advocates also stress the importance and benefits of public intervention. Open source licenses guarantee the availability of the source code and the same legal rights as those of the original developer to every individual who is interested in a certain software product. This wide availability of the "updated state-of-art," within an industry characterized by cumulative generation of knowledge, is perceived to be of crucial importance to spur innovation. In this respect, Varian and Shapiro (2003) argue that, being typically based on open interfaces, F/OS encourages third-party innovation in terms of development of, for instance, adds-on and complementary products.[14] Similarly, Benkler (2002) considers self-organization in the distributed peer production model more efficient in "acquiring and processing information about human capital available to contribute to information production projects" than traditional institutions, such as markets and hierarchies. Henkel and von Hippel (2004) push this argument further, claiming that "user innovation," a fundamental trait in F/OS software development, is welfare enhancing.

From the national perspective, those countries, whose software industry is lagging behind or is not competitive in the international markets, may consider public support to F/OS a viable way to cultivate a domestic software industry, therefore reducing their dependency from foreign suppliers; this rationale for public intervention seems to be ranked particularly high in the agenda of both emerging[15] and developed[16] countries. Varian and Shapiro (2003) sponsor this opinion and emphasize that the GNU/Linux operating system is "an open platform on which commercial or open source applications can be built, thereby spurring the development of a robust domestic

industry."[17]

Another common motivation for intervening in support of the F/OS movement is the stimulus of competition in the software market; this motive seems particularly relevant for those segments of the market characterized by the presence of dominant firms such as in the packaged software segment[18] and, more generally, in software procurement markets where dominant proprietary systems tie users to single suppliers, thus restricting competition.[19]

A More Critical View

During the last few years, several economists and other scholars have scrutinized the possible role of public policies in support of F/OS software. Apart from some relevant exceptions, the majority of authors seem to be rather skeptical about the welfare benefits that would accrue from governments directly stimulating F/OS.[20] One leading argument is that open source has emerged and, in many cases, has been extremely successful even without any intervention in place; therefore, there seems to be no need for public policies in order for F/OS to flourish. On top of that, focusing on closed source software, many authors claim that there is no clear evidence of significant failures in the software market and, consequently, there is no urge for governments' intervention. Evans (2002) and Evans and Reddy (2002) point out that the software industry is highly competitive[21] and also its performances in terms of growth, productivity, and R&D expenditures have been impressively high.[22] In other terms, software markets appear to be an example of well-functioning markets and, therefore, public funding to stimulate the emergence of alternatives to closed source software are prone to pick the "wrong winner." Moreover, a strong support to F/OS software may seriously undermine the incentives of commercial firms to innovate or to improve the quality of their software (Schmidt & Schnitzer, 2003).

One of the main arguments in favor of F/OS is that it guarantees to public administrations significant reductions in software expenses; various authors point out that cost savings obtainable by adopting F/OS rather than proprietary software are by far smaller that those expected. The licensing fees represent only a minor part of software costs and a meaningful comparison between F/OS and commercial software has to be done in terms of the total cost of ownership (TCO) which also includes user training, technical support, maintenance, and possible upgrades of the software. On these grounds, the overall cost advantage of F/OS is less evident.[23]

The higher degree of innovativeness that, according to supporters, characterizes the F/OS development mode is also a strongly debated issue. Smith (2002) acknowledges the brilliant performances of proprietary software companies in terms of R&D expenditures and resulting innovation and declares himself rather skeptical about F/OS being able to replicate such figures.[24] Evans (2002) and Evans and Reddy (2002) go even further and claim that the theoretical argument according to which open source implies more innovation completely lacks of solid empirical evidence, given that many successful F/OS software projects draw strong inspiration from already existing closed source counterparts.

This discussion reveals a widespread skepticism among economists and closed source advocates about direct government policies in favor of F/OS software; nonetheless, there is a general consensus on the need of a broader set of interventions that somehow ensure the level playing field in the software market. In particular, various authors are making strong arguments against the current system of protection of intellectual property rights. A long series of decisions taken by U.S. courts during the last twenty years has extended software patent protection and has made it easier for applicants to obtain patents even for obvious inventions. These facts have induced large firms to accumulate sizable numbers of software patents, the so-called patent thickets, that can be

strategically used in order to block competitors' innovation. As Bessen (2002, p. 13) points out, U.S. patent legislation may actually "sabotage the otherwise healthy open source movement" therefore potentially undermining competition from F/OS solutions.[25]

Finally, an issue that has drawn the attention of several contributors relates to the public funding of software R&D based on open source solutions. In this case, the non-rival and non-excludable nature of software goods, largely due to negligible replication costs, may induce policy makers to sponsor F/OS software projects as a means to increase social welfare.[26] While there is some consensus on the beneficial effects of this kind of interventions, the usage of restrictive licensing schemes (such as the GPL), is still very much debated: the software developed within publicly funded R&D projects should be made available to the widest possible audience but such restrictive licensing terms may undermine private appropriation of publicly funded basic science efforts.[27] In particular, closed source firms may be prevented from adopting and developing complementary applications for software distributed under GPL-like licensing schemes. Lessig (2002) suggests that governments should employ a non-discriminatory approach: publicly funded code should be released in the public domain or employing non-restrictive open source licenses (such as BSD-like ones).

MAIN FOCUS OF THE CHAPTER

Major Interventions in the EU

All across Europe, governments and public agencies are intervening in the software market in various ways; since September 2003, the major initiatives are registered on the Open Source Observatory, a dedicated Web site compiled by the European Commission within the IDABC program.[28] For each intervention registered on this Web site a brief abstract and, usually, a series of official documents and press releases describing the content of the policy are available. In order to derive useful information, we have reviewed the existing documentation focusing on the most important interventions registered on the IDABC site, therefore disregarding public initiatives taken by very small municipalities. The dataset we have compiled starting from the IDABC documentation has been complemented with the information recovered from an independent investigation by the Center for Strategic and International Studies (see Lewis, 2004).

It should be noted that given the methodology used within the IDABC program, the information we have gathered does not represent the complete set of initiatives taken in the European public sector. Some typologies of policies or some countries might be underrepresented in the sample. However, we believe that our effort to summarize the existing policies in favor of F/OS software represents a useful starting point to analyze the major European initiatives within a unified setting.

Overall, we have collected information about 105 interventions, distributed across 14 European countries; France is by large the most active country with more than 28% of the interventions in our sample.[29] Around 8.5% of the policies have been taken at the EU level and therefore they should be common to all European countries.

To summarize the information derived from our dataset, we have grouped policies according to:

- **Type of software involved by the intervention:** We have distinguished between *custom*, *packaged* software, and broader interventions aimed at supporting the use of *open standards*/*interfaces.*
- **Political and administrative levels at which the intervention is taken:** We have applied a two-tier classification distinguishing both between *government* and *public agencies*/bureaus (e.g., central government

vs. postal services) and between *central* and *local*/regional level of intervention (e.g., central government vs. local municipality).

- **Type of intervention:** We have grouped interventions into three broad categories: *adoption* when the government/agency has decided to adopt a certain software, *advisory* when the policy consists of a general claim of preference towards open source and/or encourages the use of F/OS or it is aimed at informing potential adopters of the existence and characteristics of open source and, finally, *development* when the government actively promotes the creation of new software.
- **Rationale for intervention:** We have classified policies into seven non-exclusive broad categories: *cost-efficiency,* that pools together motivations such as savings in license fees, economies of reuse of the software, savings from collaborative development of projects, and more efficient employment of public resources (e.g., shift from license fees to investment in human capital); *code availability,* combining motivations connected to the technical advantages assured by transparency, security, robustness, and quality of the code; *interoperability,* in which the rationale for intervention lies in stimulating the diffusion of open standards and in promoting interoperability in the software market; *flexibility,* in which motivations are linked to flexibility advantages assured by, for instance, the possibility of tailoring the code to the user's needs, to assure integration and compatibility with existing systems, and so on; *enhanced competition*, combining interventions motivated by levelling the playing field, creating alternatives to proprietary companies, supporting domestic industries, stimulating technical independence from dominant vendors, introducing competition in support, maintenance, and upgrade of systems and so forth; *efficiency in the public sector*, gathering motivations specifically related to the diffusion of best practices in public administration bodies; and, finally, *information diffusion*, a category representing those interventions motivated by the aim of increasing the available information and of raising consciousness about F/OS in the general public or, more specifically, in public administrations.

Table 2 shows the sample distribution of the various policies with respect to their type. F/OS adoption and advisory are the most common interventions in Europe: together they represent more of the 80% of the whole sample.

In Table 3 we go further into the detail and we present how the three types of policies are distributed between central and local decisional levels and between governmental authorities and public bureaus/agencies. More than 80% of the interventions in our sample are taken at the governmental level (both local and central) while agencies have played a much more limited role. Advisory policies aimed at suggesting and promoting F/OS prevail in central governments while at the other levels adoption is the most common type of intervention. This is not surprising once considered that central governments often provide "guidelines" for action while operative decisions are effectively endorsed at the local level and in agency bodies.

In Table 4 interventions are grouped according to the kind of software they are directed to:

Table 2. Public policies classified in terms of type of intervention

Intervention	Freq.	%
Adoption	47	44.8
Advisory	39	37.1
Development	19	18.1
TOTAL	105	100

Table 3. Policies classified in terms of type of intervention and administrative level

	Intervention			
Level	**Development**	**Adoption**	**Advisory**	**TOTAL**
Central Gov.	8 (17.8%)	9 (20%)	28 (62.2%)	45 (100%)
Central Agency	1 (8.3%)	9 (75%)	2 (16.7%)	12 (100%)
Local Gov.	9 (21.4%)	24 (57.1%)	9 (21.4%)	42 (100%)
Local Agency	1 (16.7%)	5 (83.3%)	0 (0%)	6 (100%)
TOTAL	19 (18.1%)	47 (44.8%)	39 (37.1%)	105 (100%)

Table 4. Policies classified in terms of software type and administrative level

	Software		
Level	**Custom**	**Packaged**	**Open Std.**
Central Gov.	69%	73%	0%
Central Agencies	33%	66%	17%
Local Gov.	38%	78%	5%
Local Agencies	83%	33%	0%
TOTAL	53%	72%	8%

either software custom or packaged or towards the implementation of open standards. Note that in many cases, the intervention is not restricted to a unique type of software but it may involve two or all of them.[30] Table 4 suggests that local governments are more active towards packaged software while central governments do not seem to follow any particular pattern.

Restricting the analysis to central governments and central agencies, we have looked more closely at the motivations behind interventions. According to the available information, only in 37 out of 57 of the cases it was possible to collect official statements explicitly accounting for the rationales for intervention. The information we have gathered is presented in Table 5. Clearly, given the small number of observations, some caution has to be exerted when interpreting these data; however, it is worthwhile to highlight the major trends that characterize European policies.

Total figures in Table 5 show that cost-efficiency motivations are the most popular, followed by interoperability and code availability ones. Regarding specific policies, adoption policies are largely motivated by interoperability (viewed at the level of the single adopter) and cost-efficiency rationales (in particular, savings on license fees) while rationales regarding technical advantages of code availability and flexibility (all subcategories equally represented) are less cited, therefore suggesting that short-term advantages might be more salient than long-term ones in the stated motivations. On the other hand, pure regulatory motivations (such as stimulating market competi-

Table 5. Public policies classified in terms of rationale for intervention (central government and agencies only)

	Type of Motivation							
Intervention	**Total**	**Cost Efficiency**	**Code Availability**	**Interoperability**	**Flexibility**	**Enhanced Competition**	**Efficiency in Public Sector**	**Information Diffusion**
Development	**6 (100%)**	1 (17%)	4 (67%)	1 (17%)	1 (17%)	1 (17%)	0 (0%)	1 (17%)
Adoption	**11 (100%)**	7 (64%)	2 (18%)	6 (55%)	3 (27%)	2 (18%)	1 (9%)	1 (9%)
Advisory	**20 (100%)**	11 (55%)	7 (35%)	11 (55%)	2 (10%)	8 (40%)	5 (25%)	7 (35%)
TOTAL	**37 (100%)**	19 (51%)	13 (35%)	18 (49%)	6 (16%)	11 (30%)	6 (16%)	9 (24%)

tion) are not explicitly accounted for. As far as advisory policies are concerned, interoperability (also considered at the market level) and cost-efficiency (all subcategories equally represented) are still fundamental rationales, but other regulatory motivations are popular as well (in particular, enhancing competition and raising awareness in markets). Finally, technical advantages of code availability represents the major rationale for R&D policies, while, surprisingly, motivation regarding cost-efficiency are rather infrequent.

FUTURE TRENDS

As we have briefly discussed in a prior section, economists are rather critical about intrusive public policies into the software market and, to some extent, we adhere to this skepticism.

Just to mention some arguments, the software industry has really proved to be extremely dynamic, characterized by high rates of growth and, while competition in some software segments might result in "winner-takes-all" outcomes, dominant positions have been frequently displaced by new comers (see Schmalensee, 2000); in a word, markets have performed reasonably well. Moreover, it is not yet clear if the production mode of open source is really more innovative than the proprietary one and empirical evidence on this issue is far from being clear-cut.

However, we believe that looking at the F/OS movement from an economic viewpoint, many relevant aspects have not received so far the attention that they should have deserved and the evidence on the EU experience reported above suggests some of the directions towards which the analysis should look at in order to better understand the actual effects of these policies.

For example, we believe that the distinction between custom and packaged software has not been properly taken into account in the literature. One of the main concerns against public support towards open source is based on the allegation

that such policies would be detrimental for the incentives to innovate by commercial firms. We have already pointed out that almost two thirds of the market is represented by software that has been developed internally or that is customized and, as shown in Table 4, more than half of the interventions in our sample relates to this latter type of software. We are convinced that the above allegation cannot apply to this kind of software: customized software is by definition software "on demand" and the incentives to develop new lines of code arise at the moment of the call for tender, regardless of the open or close nature of the source code.

From the evidence presented in a previous section, it emerges that across the EU, together with cost saving reasons, public interventions in support of F/OS founded their motivations primarily on the desire of stimulating an open standard environment for software applications but also on the relevance of source code availability and on the intention to promote more competitive software markets.

It is recognized that proprietary software is likely to create important lock-in positions; the unavailability of the source code renders adopters dependant on the original software provider for further maintenance/development/upgrade of the code. Moreover, the use of closed standards, a typical solution employed by proprietary vendors, makes it more difficult for adopters to disengage themselves from software vendors. The absence of complete and public documentation regarding file and data storage formats and other communication standards might substantially increase the switching costs thus rendering unprofitable the migration to other software packages. Lock-in is certainly a source of a relevant increase in life-cycle costs but these costs are extremely difficult to evaluate when one wants to compute correctly the total cost of ownership of a given software product.

On the contrary, a relevant feature of both open source code and open standards is that competition may be created in the aftermarket, and this may significantly reduce the cost of service, support, maintenance and interoperability.[31] Moreover, according to this view, fears of picking "wrong winners" through governmental advisory or adoption of F/OS solutions should be lessened if one takes into account that F/OS software is based on open formats that are commonly available and that might be employed by closed source vendors to develop compatible value-added proprietary solutions or interoperable adds-on and complementary products.[32]

While the above arguments apply to custom software in particular, a regulatory policy in support of open standards may found solid justifications also in the context of mass-market software; as a consequence of strong network effects, these segments of the software industry are often characterized by the presence of dominant players whose platforms have the typical features of "essential facilities." Controlling an interface (the key input) allows the dominant firm to protect its position and possibly to extend it to other complementary products. Similarly to the current practice in other industries, also for the case of software the provision of open access to the essential facility should be seriously considered in order to promote competition and to improve market efficiency.

CONCLUSION

The bottom line is to ensure that markets lead to efficient outcomes and therefore to exclude, based on economic grounds, that public interventions might be beneficial relates to the assumption that all potential adopters are properly informed about the alternatives that are available in the market. A recent empirical study on F/OS in the public sector shows that this is not necessarily the case. Ghosh and Glott (2005) show that a large share of IT administrators in the public sector ignore that in their agencies F/OS was actually employed.[33]

More interestingly, the fact of being aware or not about the current usage of open source software has a major impact on the evaluation of the potential benefits of F/OS adoption. Nearly 70% of the "aware IT administrators" finds it useful to extend the use of open source in their agencies. This percentage shrinks to 30% among the IT administrators that were unaware that F/OS software was already employed in their institutions. Clearly, this evidence provides strong support for policies aimed at informing potential adopters about the characteristics and the availability of open source solutions.[34]

ACKNOWLEDGMENT

The authors would like to thank Bruno Caprile, Vincenzo D'Andrea, Sebastian Spaeth, Ruben van Wendel de Joode and two anonymous referees for their helpful comments on earlier drafts of this chapter. The usual disclaimer applies. Financial supports from Progetto di Ateneo 2006—University of Padua (for Stefano Comino and Fabio Manenti) and from MIUR under the projects FIRB03 and PRIN05 (for Alessandro Rossi) are gratefully acknowledged.

REFERENCES

Benkler, Y. (2002). Coase's penguin, or, Linux and the nature of the firm. *Yale Law Journal, 112*(3), 369-446.

Bessen, J. (2002). What good is free software? In R. Hahn (Ed.), *Government policy toward open source software* (pp. 12-33). Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

Bessen, J., & Hunt, R. M. (2004). *An empirical look at software patents* (Working Paper 03-17). Federal Reserve Bank of Philadelphia.

Comino, S., & Manenti, F. M. (2005). Government policies supporting open source software for the mass market. *Review of Industrial Organization, 26*, 217-240.

Danish Board of Technology. (2002). *Open source software in e-government.* Retrieved from http://www.tekno.dk/pdf/projekter/p03_open-source_paper_english.pdf

DeLong, J. B., & Froomkin, A. M. (2000). Speculative microeconomics for tomorrow's economy. *First Monday*, 5(2).

Evans, D. S. (2002). Politics and programming: Government preferences for promoting open source software. In R. Hahn (Ed.), *Government policy toward open source software* (pp. 34-49). Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

Evans, D. S., & Layne-Farrar, A. (2004). Software patents and open source: The battle over intellectual property rights. *Virginia journal of law and technology, 9*(10), 1-28.

Evans, D. S., & Reddy, B. (2002, May 21). *Government preferences for promoting open-source software: A solution in search of a problem.* NERA Economic Consulting report. Retrieved from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=313202

Finnish Minister of Finance. (2003). *Recommendation on the openness of the code and interfaces of state information systems* (Working Paper 29/2003). Retrieved from http://www.vm.fi/vm/en/04_publications_and_documents/01_publications/03_working_group_memoranda/20031015Recomm/65051.pdf

Forge, S. (2005). Towards an EU policy for open-source software. In M. Wynants & J. Cornelis (Eds.), *How open is the future?* (pp. 489-503). Brussels, Belgium: VUB Brussels University

Press.

Ghosh, R., & Glott, R. (2005). *Free/libre and open source software: policy support.* (Results and policy paper from survey of governments authorities.) Maastricht, The Netherlands: University of Maastricht, MERIT.

Ghosh, R., Krieger, B., Glott, R., & Robles, G. (2002, June). *Free/libre and open source software: Survey and study* (Deliverable D18, Final report, Part 2B: Open source software in the public sector: policy within the European Union). Maastricht, The Netherlands: University of Maastricht, International Institute of Infonomics.

Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy, 35*(7), 953-969.

Henkel, J., & von Hippel (2004). Welfare implications of user innovation. *The Journal of Technology Transfer, 30*(1-2), 73-87.4

Lessig, L. (2002). Open source baselines: Compare to what? In R. Hahn (Ed.). *Government policy toward open source software* (pp. 50-68). Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

Lewis, J. A. (2004, August 1). *Global policies on open source software*. Center for Strategic and International Studies report. Retrieved from http://www.csis.org/index.php?option+com_csis_pubs&task_view&id=3046

Schmalensee, R. (2000). Antitrust issues in schumpeterian industries, *American Economic Review, 90*, 192-196.

Schmidt, K., & Schnitzer, M. (2003). Public subsidies for open source? Some economic policy issues of the software market. *Harvard Journal of Law and Technology, 16*(2), 473-505.

Schmitz, P.E. (2001). *Use of open source in Europe, an IDA study.* European Commission, DG Enterprise. Retrieved from http://europa.eu.int/idabc/servlets/Doc?id=1973

Schmitz, P.E., & Castiaux, S. (2002). *Pooling open source software, An IDA feasibility study. Interchange of data between administrations.* European Commission, DG Enterprise. Retrieved from http://europa.eu.int/idabc/en/document/2623/5585#feasibility

Smith, B.L. (2002). The future of software: Enabling the marketplace to decide? In R. Hahn (Ed.), *Government policy toward open source software* (pp. 69-86). Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

U.S. Federal Trade Commission. (2003). *To promote innovation: the proper balance of competition and patent law and policy.* Retrieved from http://www.ftc.gov/os/2003/10/innovationrpt.pdf

Varian, H., & Shapiro, C. (2003). *Linux adoption in the public sector: an economic analysis,* mimeo. University of Berkeley, California.

von Hippel, E. (2005). *Democratizing innovation.* Cambridge, MA: MIT Press.

Wheeler, D. A. (2005). *Why open source software / free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers!* Retrieved from http://www.dwheeler.com/oss_fs_why.html

KEY TERMS

Customers' Lock-In: A situation in which a customer is so dependent on a vendor for products and services that he/she cannot move to another vendor without substantial switching costs, real and/or perceived.

Economic Regulation: Set of restrictions promulgated by government administrative agencies through rulemaking supported by a threat of sanction or a fine. The main scope for government's regulation is to prevent markets' failures, in

other words, situations in which markets do not efficiently organize production or allocate goods and services to consumers (as in the presence of a monopoly/dominant firm).

Essential Facility: In a vertically related market, it is defined as a facility, function, process, or service that meets three criteria: it is monopoly controlled; a potential competitor requires it as an input to provide services and to compete downstream with the monopoly supplier; and it cannot be economically or technically duplicated. Facilities that meet this definition shall be subject to mandatory unbundling and mandated pricing.

Intellectual Property Rights (IPRs): Intellectual property is a term used to refer to the object of a variety of laws, including patent law, copyright law, trademark law, trade secret law, and industrial design law. These laws provide exclusive rights to certain parties over intangible subject matter or over the product of intellectual or creative endeavor; many of them implement government-granted monopolies.

Proprietary Software (PS): Software products that are designed in such a way that others cannot access or view a product's source coding/the programming that allows the software to perform certain functions.

Source Code: The programming that allows software programs to perform certain actions or functions.

Total Cost of Ownership (TCO): Financial estimate aimed at helping consumers and enterprise managers to assess direct and indirect costs related to the purchase of any capital investment, such as (but not limited to) computer software or hardware.

ENDNOTES

1 Further details are available at: http://europa.eu.int/information_society/eeurope/2005/index_en.htm. All the URLs provided in this chapter are active at the moment of writing the chapter (June 2006).

2 Just to give a relevant example, the Dutch public sector spent around 400 million euros on software in 1997; see http://www.ososs.nl.

3 For an example at the transnational level see the European Interoperability Framework for pan-European eGovernment services, mandating a series of policies, standards and guidelines aimed at "facilitating […] the interoperability of services and systems between public administrations, as well as between administrations and the public" (http://europa.eu.int/idabc/en/document/2319/5644). For an application at the national level the reader may refer to the Dutch manual on open standards and open source software (OSOSS) in the procurement process, encouraging the adoption of open standards in the public sector (http://www.ososs.nl).

4 According to Bessen (2002), packaged software has never accounted for more than a third of software expenses.

5 See, for instance, Microsoft's Jan. 25, 2006 press release available at http://www.microsoft.com/presspass/press/2006/jan06/01-25EUSourceCodePR.mspx and the comments of Neelie Kroes (European Union's antitrust chief), stating that documentation enabling interoperability, rather than mere code disclosure, is at issue in order to meet EU's requirements (http://today.reuters.com/business/newsArticle.aspx?type=technology&storyID=nL26331447).

6 http://europa.eu.int/idabc/en/document/4549/469

7 http://europa.eu.int/idabc/en/document/3336/469.

8 http://europa.eu.int/idabc/en/document/1766/469.

9 http://europa.eu.int/rapid/pressRe-

leasesAction.do?reference=IP/04/382&format=HTML&aged=l&language=EN&guiLanguage=en.

10 For a brief but comprehensive review of various national initiatives and policies on open source software see the links provided by the IDABC Open Source Observatory at http://europa.eu.int/idabc/en/document/1677/471.

11 A notable example of this kind of motivations can be found in the programs and activities of the Free Software Foundation, aimed at affirming the primacy of freedom ideals in the development and diffusion of software.

12 For a comprehensive survey on this topic see Wheeler (2005).

13 Reuse economies are savings due to recycling previously developed code as a basis for a new project; collaborative development economies are strategies of mutualization consisting in partnerships for joint development by the public sector, motivated by the needs of pooling efforts and sharing costs in building, maintaining and upgrading large software projects of common interest. See Schmitz and Castiaux (2002) for an assessment applied to FO/S software.

14 Bessen (2002) holds a similar view.

15 Support to domestic software industry lies at the core of the IT national policies of India and China. See, for instance, the remarks of the Indian President, A.P.J. Abdul Kalam, on the future challenges of information technology for developing countries (http://news.com.com/2100-1016-1011255.htmlnews.com.com/2100-1016-1011255.html) or the speech of the Ministry of Science and Technology at the 2004 International Conference on Strategies for Building Software Industries in Developing Countries (http://www.iipi.org/Conferences/Hawaii_SW_Conference/Li%20Paper.pdf).

16 This occurs both at the national as well as at the local levels. See the statement by the Finnish Ministry Kyösti Karjula (http://www.linuxtoday.com/news_story.php3?ltsn=2002-06-17-011-26-NW-DP-PB) as an example of the first type and the deliberation of the autonomous province of Trento on the adoption of open standards and open source software (http://www.linuxtrent.it/Members/napo/deliberaPAT_n1492.pdf) as an instance of the second type.

17 Smith (2002) contrasts this view arguing that in a large number of countries, not only in the developed ones, a flourishing (proprietary) software industry already exists.

18 Among others, see the statement made by Boris Schwartz, deputy leader of the SPD parliamentary group, during the debate about the transition towards open source systems of the city of Munich (http://www.linuxtoday.com/infrastructure/2003052600126NWSWPB).

19 See, for instance, the recommendations of the Danish Board of Technology (2002) on supporting the emergence of alternatives in custom built software markets as means to foster competition and the recommendations of the Finnish Minister of Finance (2003), suggesting to include the possession of the source code in tender drafts in order to assure competitive bidding in future development and maintenance.

20 One notable exception is represented by Lessig (2002) who claims that government preference towards F/OS is justified by the presence of externalities that market forces do not internalize. For instance, software developed for or adopted by some branches of the government could be employed usefully also by other branches if it is free or open source; the initial development/adoption decision should take into account also the potential benefits for future users.

21 These authors provide several figures to support their argument. In the US the Herfindahl-Hirschman index (HHI) for the

software industry is smaller that the average HHI computed for the US manufacturing industries; furthermore, during the period 1996-2000 there has been a decrease by 27% in the quality-adjusted prices for the packaged software.

22 According to Evans (2002), in the year 2000 the R&D expenditure of software companies represented one tenth of the overall R&D undertaken within the industrial sectors while fifteen years before it accounted for only 1%.

23 The empirical evidence comparing the TCO of open vs. close software solutions does not seem to be conclusive. For a comprehensive overview the reader may refer to the FlossPols report on policy support (Ghosh & Glott, 2005).

24 Smith, Microsoft's senior vice president, also claims that often, in order to bring the software to the market, additional investments have to be done and these can not accrue from the F/OS world but can only come from the commercial one.

25 For an empirical analysis on software patents see Bessen and Hunt (2004). According to these authors, the strategic accumulation of patent thickets seems to be the most convincing explanation for the large increase of software patenting in the US. Similarly, several panelists, according to a recent US Federal Trade Commission (2003) report, support the view that the patent protection system poses threats to innovation in the software industry. Lessig (2002) and von Hippel (2005) argue in favor of lessening the extent of patent protection in the software industry. According to Evans (2002) and Evans and Layne-Farrar (2004), even though some (minor) reform of the patent legislation might be beneficial, software patents should not be banned altogether.

26 See, for instance DeLong and Froomkin (2000) for an application to digital goods markets.

27 Smith (2002) and Lessig (2002) hold the view that government should finance R&D activities but the resulting software should not be distributed under restrictive licensing schemes. On the contrary, Varian and Shapiro (2003) focusing on the Linux case argue that the adoption of GPL does not necessarily prevents the development of complementary applications. Henkel (2006) provides empirical evidence that, despite GPL's strict requirements in releasing derived works, firms can adopt several successful strategies in order to protect their own code enhancements.

28 IDABC stands for Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens; the information available on the Open source Observatory is collected by a special Web-team from staff members of the European public sector and also by searching the Internet for relevant information. The documentation we have collected is available at the following URL http://europa.eu.int/idabc/en/chapter/491.

29 The large interest of public authorities in France has been documented also in a previous IDABC report, see Schmitz (2001).

30 This fact explains why rows sum up to more than 100%.

31 On these lines, Ghosh, Krieger, Glott, and Robles (2002) suggest that whenever it is feasible governments and public institutions should opt for software open source, for example, by granting unlimited access to the source code, the right to modify the software and that to reproduce and distribute an unlimited amount of copies of the modified version under the same license restrictions. Forge (2005, p. 492) argues that policy-markers should mandate "backward compatibility, open access to program interfaces, and separation between operating

systems and applications".

[32] Moreover, it is worth mentioning that in some cases policies supporting F/OS software are inspired by neutrality principles, therefore suggesting joint use rather than full substitution of closed source software by migrating to F/OS systems.

[33] According to the authors 30% of IT administrators were unaware of F/OS software usage and this figure increases in the case of small budget public agencies.

[34] A welfare analysis of the impact of various policies supporting F/OS in the presence of "unaware" potential adopters can be found in Comino and Manenti (2005).

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited byK. St. Amant; and B. Still, pp. 412-427, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.8
Open Source Survey Software

Jason D. Baker
Regent University, USA

ABSTRACT

One of the significant advances in software design afforded by the Internet has been the open source movement, an effort to collaboratively create software and make it widely and freely available to the online community. Although the open source movement started with Unix-like computer operating systems, it has expanded to include a wide variety of software programs, including tools to publish and analyze online surveys. This chapter introduces the open source movement, and then profiles three leading open source survey programs: php Easy Survey Package (phpESP), PHP Surveyor, and the Moodle course management system.

BACKGROUND

The open source movement has its roots in the Unix community and, in particular, the development of the GNU Project back in 1984. The goal of this idealistic group of software developers was to create an entirely free Unix operating system so users would not be dependent on commercial versions from Sun, IBM, and others. Here is the introduction from the GNU Web page:

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. (GNU is a recursive acronym for 'GNU's Not Unix'; it is pronounced 'guh-NEW'.) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as 'Linux,' they are more accurately called GNU/Linux systems. (http://www.gnu.org)

Since then, the movement has grown far beyond computer scientists writing operating system code, and has become an ideology in which people freely contribute to content that is made freely available, and where changes are not only acceptable, but encouraged, just as long as the results are offered back to the community. In many ways, it is a sophisticated countercultural response to the limitations of copyrights and patents. As Raymond (1998) declared, "Perhaps in the end the open-source culture will triumph…simply because the commercial world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem."

Not surprisingly, the open source approach has moved beyond computer operating systems into high-demand computer and Internet-based applications, including survey software. For those individuals or organizations unable or uninter-

ested in paying commercial hosting companies to administer online surveys and tests, there are open source equivalents. Popular open source survey programs include the php Easy Survey Package (phpESP) and PHP Surveyor. For educational-style tests, the Moodle open source course management system is one of the more popular.

PROFILES AND COMMENTARY

phpESP, PHP Surveyor, and Moodle are all open source programs that run on Unix Web platforms and make use of the php scripting language and the MySQL database system. Each of these programs can be installed using their respective Web-based installation programs. Additionally, many Web hosting providers offer one-click installation of various open source programs through a menu-driven program called Fantastico De Luxe. Fantastico generally comes bundled with phpESP, PHP Surveyor, and Moodle, along with other useful open source programs. Once installed, all three programs can be administered via a Web-based interface.

phpESP is the older of the survey programs and features a number of basic and advanced online survey functions. To create a survey, one first configures the general survey details, along with the look and feel of the survey template, and then populates the survey with individual questions. A variety of question types are supported including yes/no, multiple choice, check boxes, Likert-style scales, short answer, and even essay (although open-ended questions cannot be tallied the same way as multiple-choice and other fixed questions). Once all of the questions and answer options are entered and ordered, the user can test the survey before publishing it for real. Once activated, the survey administrator can view individual and aggregate results through the management interface, as well as download the results for analysis in Excel, SPSS, or similar programs. The strength of phpESP is its varied features, extensive development history, and simplicity of code modification. The weakness of the program is that it lacks detailed documentation and thus, can be confounding to basic computer users.

PHP Surveyor possesses many of the same survey features as phpESP, but couples them with a significantly richer administration interface that makes the program more accessible to novices. In addition to 20 different question types ranging from multiple choice and checkboxes to Likert-style scales and flexible arrays (which permit custom text descriptors in each point along the scale), PHP Surveyor supports open or closed surveys. Open surveys can be completed by anyone visiting the Web site, while closed surveys require registration or invitation to participate. Furthermore, the survey administrator can preregister selected users and send out e-mails to solicit participation in the online survey. PHP Surveyor also supports branching surveys, which enable different follow-up questions to be presented based on answers to previous questions, and uses a templating feature to change the look and feel of online surveys. Survey responses can be reviewed online or downloaded for more in-depth statistical analysis. While still under heavy development, PHP Surveyor is likely to become a leading open source survey program because of its rich feature set and relative ease of use.

Unlike phpESP and PHP Surveyor, Moodle is not strictly a survey program, but rather is an open source course management system (CMS). Moodle can be considered an open source equivalent to commercial CMSs like Blackboard or WebCT and thus, is designed for instructional purposes. It includes features to post course syllabi, documents, grades, and hold online class discussions. In addition, however, Moodle includes robust online quiz and survey modules that can be used to administer online assessments or surveys. The quiz module enables instructors to set up assessments, along with the correct answers, so students can complete these and have their scores provided instantly. The survey module includes predefined

surveys that can be used with online courses, and also supports the development of custom surveys. Results can be viewed online, or downloaded in Excel or comma-separated value format. Since Moodle is designed for online courses, complete with student accounts and registration, it is less ideal for a public survey than either phpESP or PHP Surveyor, but has many more options if one is integrating such online tests and surveys into coursework.

COST

All three programs are open source and available at no cost. In addition to the free download, the source code is also freely available, and can be changed as desired.

LOCATION

phpESP is available at http://phpesp.sourceforge.net

PHP Surveyor is available at http://www.phpsurveyor.org

Moodle is available at http://www.moodle.org

REFERENCES

Dougiamas, M., & Taylor, P. (2003). Moodle: Using learning communities to create an open source course management system. In P. Kommers & G. Richards (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003* (pp. 171-178). Chesapeake, VA: AACE.

Kaskalis, T. H. (2004). Localizing and experiencing electronic questionnaires in an educational Web site. *International Journal Of Information Technology, 1*(4), 187-190.

Rapoza, J. (2005). Open-source survey tool phpESP stands test of time. *eWeek*. Retrieved December 15, 2005, from http://www.eweek.com/article2/0,1895,1749890,00.asp

Raymond, E. S. (1998, March). The cathedral and the bazaar. *First Monday, 3*(3). Retrieved February 15, 2005, from http://www.firstmonday.org/issues/issue3_3/raymond/

KEY TERMS

Course Management System: A computer software program designed to support the delivery of online instruction. Popular CMSs include Blackboard, WebCT, and Moodle.

Database: A software package for storing information in a searchable and relational structure. Popular databases include Oracle, MySQL, SQL Server, and Access.

Open Source Software: An approach to software development where multiple individuals collaboratively write code and release both the source code and resulting program to the online community. Anyone is then permitted to freely use the software and modify the code, provided that such modifications are again made available in a public release of the code.

Unix: A popular computer operating system. Many Web servers run on Unix-based systems.

This work was previously published in Handbook of Research on Electronic Surveys and Measurements, edited by R. Reynolds; R. Woods; and J. Baker, pp. 416--418, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.9
FLOSSmole:
A Collaborative Repository for FLOSS Research Data and Analyses

James Howison
Syracuse University, USA

Megan Conklin
Elon University, USA

Kevin Crowston
Syracuse University, USA

ABSTRACT

This article introduces and expands on previous work on a collaborative project, called FLOSSmole (formerly OSSmole), designed to gather, share, and store comparable data and analyses of free, libre, and open source software (FLOSS) development for academic research. The project draws on the ongoing collection and analysis efforts of many research groups, reducing duplication, and promoting compatibility both across sources of FLOSS data and across research groups and analyses. The article outlines current difficulties with the current typical quantitative FLOSS research process and uses these to develop requirements and presents the design of the system.

INTRODUCTION

This article introduces a collaborative project called FLOSSmole,[1] designed to gather, share, and store comparable data and analyses of free and open source software development for academic research. The project draws on the ongoing collection and analysis efforts of many research groups. Our intent in developing FLOSSmole is to reduce duplication, and to promote compatibility both across sources of FLOSS data and across research groups and analyses.

Creating a collaborative data and analysis repository for research on FLOSS is important because research should be as reproducible, extendable, and comparable as possible. Research with these characteristics creates the opportunity to employ meta-analyses, exploiting the diversity of existing research by comparing and contrasting results to expand our knowledge. Unfortunately, the current typical FLOSS research project pro-

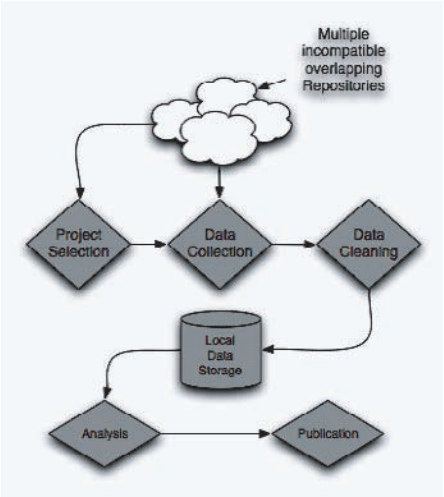
ceeds in a way that does not necessarily achieve these goals. These goals require detailed communal knowledge of the many choices made throughout a research project. Traditional publication prioritizes results, but masks or discards much of the information needed to understand and exploit the differences in our data collection and analysis methodologies. FLOSSmole was originally designed to provide resources and support to academics seeking to prepare the next generation of FLOSS research. Since its inception, FLOSSmole has also been a valuable resource for nonacademics who are also seeking good data about development practices in the open source software industry.

BACKGROUND OF PROBLEM

Obtaining data on FLOSS projects is both easy and difficult. It is easy because FLOSS development utilizes computer-mediated communications heavily for both development team interactions and for storing artifacts such as code and documentation. This way of developing software leaves a freely available and, in theory at least, highly accessible trail of data upon which many academics have built interesting analyses about optimal organization of development teams, economics of building software in the commons, and the like. Yet, despite this presumed plethora of data, researchers often face significant practical challenges in using this data to construct a collaborative and deliberative research discourse. In Figure 1, we outline the research process we believe is followed in much of the quantitative literature on FLOSS.

The first step in collecting online FLOSS data is selecting which projects and which attributes to study, two techniques often used in estimation and selection are census and sampling. (Case studies are also used but these will not be discussed in this article.)

Figure 1. The typical quantitative FLOSS research process (notice its noncyclical and noncollaborative nature)



Conducting a census means to examine all cases of a phenomena, taking the measures of interest to build up an entire accurate picture. Taking a census is difficult in FLOSS for a number of reasons. First, it is hard to know how many FLOSS projects there are "out there," and it is hard to know which projects should actually be included. For example, are corporate-sponsored projects part of the phenomenon or not? Do single-person projects count? What about school projects?

Second, the projects themselves, and the records they leave, are scattered across a surprisingly large number of locations. It is true that many are located in the major general repositories, such as Sourceforge[2] and GNU Savannah.[3] It is also true, however, that there are a number of other repositories of varying sizes and focuses (e.g., CodeHaus,[4] CPAN[5]), and that many projects, including the well-known and much-studied Apache and Linux projects, prefer to use their own repositories and their own tools. This diversity of location effectively hides significant portions of the FLOSS world from attempts at census. Even if a full listing of projects and their locations

could be collated, there is also the practical difficulty of dealing with the huge amount of data — sometimes years and years of e-mails, CVS, and bug tracker conversations — required to conduct certain comprehensive analyses.

Do the difficulties with census-taking mean that sampling would be more effective? By saying sampling we mean taking a random selection of a small (and thus more manageable) subgroup of projects that can, through careful selection, represent the group as a whole. While this will go some way toward solving the manageability problem, sampling FLOSS projects is difficult for the same reason as census-taking: the total population from which to take the sample selection is not well-defined. Perhaps more importantly, sampling open source projects is methodologically difficult because everything FLOSS research has shown so far points to massively skewed distributions across almost all points of research interest (Conklin, 2004; Xu, Yongqin, Christley, & Madey, 2004). Selecting, even at random, from highly skewed distributions does not, in general, produce a representative sample. The difficulty of sampling is demonstrated in the tendency of FLOSS studies to first limit their enquiries to projects using one repository (usually Sourceforge), and often to draw on samples created for entirely different purposes (such as top 100 lists as in Krishnamurthy, 2002), neither of which is a satisfactory general technique. Selection of projects to study is further complicated by the fact that the public repositories contain a large number of projects that are dormant, relocated, or dead.

BACKGROUND: DATA COLLECTION DIFFICULTIES

Once the projects of interest have been identified and located, the actual project data must be collected. There are two techniques that prevail in the FLOSS literature for collecting data: Web spidering and obtaining database dumps.

Spidering data is fraught with practical complexities (Howison & Crowston, 2004). Because the FLOSS repositories are usually maintained using a database back-end and a Web front-end, the data model appears straightforward to reproduce. The central limitation of spidering, however, is that the researcher is continually in a state of discovery. The data model is always open to being changed by whoever is controlling the repository, and there is usually no way that the researcher will know of changes in advance. Spidering is a time-intensive and resource-consuming process, and one that is being unnecessarily replicated throughout the world of FLOSS research.

Getting direct access to the database is clearly preferable, but not all repositories make their dumps available. (Some, such as Freshmeat, provide a nightly build containing several text files with the majority of their information included.) And understandably so because it is not a costless process to make data-dumps available. Dumps can contain personally identifiable and financial information (as with the Sourceforge linked donation system) and so the data must be anonymized or otherwise modified to protect this information. Repositories are facing an increasing number of requests for database snapshots from academics and are either seeking a scalable way to do releases or declining to release the data entirely. It is often unclear whether database dumps obtained by one research project can be shared with other academics, so rather than possibly breach confidentiality or annoy their subjects by asking for signed releases, it is understandable that academics who do get a database dump may not make those dumps easily available. Other projects, such as the Sourceforge dump available from Notre Dame[6], only provide the dumps to qualified academic researchers with editorial restrictions. It is unclear what effect this limitation will have on research efforts in the open source

community, however, since research efforts are certainly not limited to academics.

Even when dumps are available it is necessary to interpret their database schema. This is not always as straightforward as one would expect. After all, the databases were designed to be used to build Web pages quickly, not to conduct academic analyses. Furthermore, they have been built over time and face the complexity that any schema faces when stretched and scaled beyond its original intended use: labels are obscured, extra tables are used, and there are inconsistencies between old and recently added data. The interpretation and transformation of this data into information semantically interesting to researchers is not a trivial process, and there is no reason to think that researchers will do this transformation in a consistent fashion.

Even pristine and labeled data from repositories is not sufficient because different repositories store different data. Different forges can have projects with the same names, different developers can have the same name across multiple forges, and the same developer can go by multiple names. Forges have different terminology for things like developer roles, project topics, and even programming languages. They often have fields which are named the same in multiple forges but which represent different data. Another problematic area is calculated fields, such as activity or downloads, for which there is incomplete publicly available information on their formula or correctness.

BACKGROUND: DATA CLEANING DIFFICULTIES

Once projects have been selected and the available data harvested, researchers must be confident that data adequately represent the activities of a project. For example, projects use repository tools to differing degrees. For example, many projects are listed on Sourceforge, and use the mailing lists and Web hosting provided there. But some of these same projects will shun the notoriously quirky "Tracker" bug-tracking system at Sourceforge, preferring to set up their own tracking systems using, perhaps, Bugzilla or RT software. Other projects host their activities outside Sourceforge but maintain a "placeholder" registration with little used mailing lists and out of date release information. It is very difficult, short of detailed examination of each project, to know whether a project is fully using a tool. Thus, it is difficult to state with confidence that the data collected about that tool is a reasonable depiction of the project's activities.

Complete accuracy is, of course, not required because in large-scale data analysis some "dirty data" is acceptably handled through statistical techniques. At a minimum, though, researchers contemplating the accuracy of their data must have some reason to believe that there are no systematic reasons that the data collected in the name of the group would be unrepresentative. Unfortunately, given the idiosyncrasies of FLOSS projects, confidence on this point appears to require project-by-project verification, a time-consuming process for individual researchers and projects, and one that is too frequently repeated by other researchers.

The conclusion we draw from this analysis is that each step of the typical FLOSS research process introduces variability into the data that underlies any quantitative analysis of FLOSS development. Decisions about project selection, collection, and cleaning are compounded throughout the cycle of research. FLOSS researchers have not, so far, investigated the extent to which this variability affects their findings and conclusions. In addition, the demands of traditional publication also mean that the decisions are not usually fully and reproducibly reported.

Our critique is not against the existence of differences in research methods or even difference in datasets. There is, rightly, more than one way to conduct research, and indeed this richness drives discovery. Rather, our critique is that the

research community is currently unable to begin a meta-analysis phase or a reflective phase because the current process of FLOSS research introduces variability that is difficult to trace. The research process is also hampered by redundant, wasted effort in data collection and analysis. It is time to learn from the free and open source approaches we are studying and develop an open, collaborative solution to this problem.

PROPOSING A SOLUTION: FLOSSMOLE

The previous problem description motivates our attempt to build a system to support research into FLOSS projects. FLOSSmole (formerly OSSmole — the name was changed to reflect our inclusion of Free and Libre software in addition to open source software) is a central repository of data and analyses about FLOSS projects which have been collected and prepared in a decentralized manner. Data repositories have been useful in other fields; the presence of trusted datasets allows research communities to focus their efforts. For example, the TREC datasets have supported a community of information retrieval specialists facilitating performance and accuracy comparisons; the UMI machine learning repositories have been widely used in the development of new machine learning algorithms. There are numerous examples from biology and physics as well. The intention of FLOSSmole is to provide high-quality and widely used datasets, and to share standard analyses for validation, replication, and extension.

REQUIREMENTS OF THE FLOSSMOLE SYSTEM

Below we list some of our initial requirements for an optimal data and analysis clearinghouse for FLOSS data, and we note to what extent FLOSSmole has met each of these requirements. The next section expands on additional specific design attributes of the FLOSSmole system.

An optimal data and analysis repository for FLOSS data should be:

Collaborative. The system should leverage the collective effort of FLOSS researchers to reduce redundancies and to free researchers' time to pursue novel analyses. Thus, in a manner akin to the BSD rather than the GPL licensing model, FLOSSmole expects, but does not require, that those that use data contribute additional data and the analysis scripts that they obtain or use.

Available. The system should make the data and analysis scripts available without complicated usage agreements, where possible through direct unmonitored download or through interactive database queries. This should end the problem of data lockup, and will ease entry of new researchers with novel techniques. Freely available data also lowers the barriers to collegial replication and critique. FLOSSmole scripts and data are open-sourced and available to anyone via the FLOSSmole Sourceforge project page.[7]

Comprehensive and compatible. Given the fragmentation of FLOSS project storage identified previously, the system should cover more than just one repository. The system should be able to pull historical snapshots for purposes of replication or extension of earlier analyses. Compatibility requires that the system should translate across repositories allowing researchers to conduct both comprehensive and comparative analyses. (Currently FLOSSmole contains data from three repositories.) There exists the potential to develop an "interchange" format for FLOSSmole project collateral which projects themselves, which fear data and tool lock-in, might find convenient and

useful as they experiment with new tools and repositories.

Of high quality. Researchers should be confident that the data in the system is of high quality. The origins and collection techniques for individual data-points must be traceable so that errors can be identified and not repeated. Data validation performed routinely by researchers can also be shared (for example, scripts that sanity-check fields or distributions) and analyses validated against earlier analyses. By implementing these requirements, FLOSSmole is potentially a large advantage over individual research projects working with nonvalidated single datasets because it implements the "many eyeballs" FLOSS methodology for quality assurance.

Able to support reproducible and comparable analyses. It is desirable that data extracted from the database for transformation be exported with verbose comments detailing its origins and how to repeat the extraction. The best way to ensure reproducible and comparable analyses is to have as much of the process as possible be script-driven, and in this goal, FLOSSmole excels. Optimally, the system should specify a standard application programming interface (API) for inserting and accessing data via programmed scripts. That would allow analyses to specify, using the API, exactly the data used.

A system that meets these requirements, we believe, will promote the discovery of knowledge about FLOSS development by facilitating the next phase of extension through replication, apposite critique, and well-grounded comparison.

ADDITIONAL DESIGN DETAILS

The FLOSSmole data model is designed to support data collection, storage, and analysis from multiple free and open source forges in a way that meets the previously stated requirements. This section lists some additional design details we have made in implementing our FLOSSmole system.

FLOSSmole is able to take both spidered data and data inserted from a direct database dump. The raw data is time stamped and stored in the database, without overwriting any data previously collected, including data from the same project and from the same forge. Finally, periodic raw and summary reports are generated and made publicly available on the project Web site.

The type of data that is currently collected from the various open source forges includes the full HTML source of the forge data page for the project, project name, database environments, programming languages, natural languages, platforms, open source license type, operating systems, intended audiences, and the main project topics. Developer-oriented information includes number of developers, developer information (name, username, e-mail), and the developer's role on the project. We have also collected issue-tracking data (mainly bugs), such as date opened, status, date closed, and priority. Data has been collected from Sourceforge, GNU Savannah, the Apache foundation's Bugzilla and Freshmeat. We are currently creating mappings between fields from each of these repositories and assessing how comparable the fields are. The forge-mapping task is extensive and time-consuming, but the goal is to build a dataset that is more complete and is not specific to only one particular forge.

FLOSSmole is constantly growing and changing as new forges are added. And because data from multiple collectors are both expected and encouraged, it is important that the database also store information about where each data record originally came from (i.e., script name, version, command-line options used, name and contact information of person donating the data, and date of collection and donation). This process ensures accountability for problematic data, yet encourages collaboration between data collectors. The

information is stored inside the database to ensure that it does not get decoupled from the data.

Likewise, it is a general rule that data are not overwritten when project details change; rather, one of the goals of the FLOSSmole project is that a full historical record of the project be kept in the database. This will enable researchers to analyze project and developer changes over time and enable access to data that are difficult or impossible to access once they are no longer viewable from the repository's front-end interface.

Access to the FLOSSmole project is two-pronged: both data and scripts are continually made available to the public under an open source license. Anyone can download the FLOSSmole raw and summary data for use in their own research projects or just to get information about the state of the art in open source development. The raw data are provided as multiple text file "data dumps" from the FLOSSmole database. Summary files are compiled periodically, and show basic statistics. Examples of summary statistics that are commonly published would be the count of projects using a particular open source license type, or the count of new projects in a particular forge by month and year, or the number of projects that are written using each programming language. It is our hope that more sophisticated analyses will be continually be contributed by researchers, and that the system will provide dynamic and up-to-date results rather than the static pictures that traditional publication unfortunately leaves us.

The scripts that populate the FLOSSmole database are also available for download under an open source license. These scripts are given for two reasons: first, so that interested researchers can duplicate and validate our findings, and second, so that anyone can expand on our work, for example, by modifying a script to collect data from a new forge. Indeed this process has begun with the recent publication of a conference paper comparing and commenting on our spidering and summaries and beginning collaboration (Weiss, 2005). FLOSSmole expects and encourages contributions of additional forge data, and interested researchers should see the FLOSSmole project page and join the mailing list for information on how to contribute.

RESULTS

Because it is a regularly updated, publicly available data repository, FLOSSmole data have been used both for constructing basic summary reports about the state of open source, as well as for more complex social network analyses of open source development teams. For example, summary reports posted as part of the FLOSSmole project regularly report the number of open source projects, the number of projects per programming language, the number of developers per project, and so forth. These sort of descriptive data are useful for constructing "state of the industry" reports, or for compiling general statistical information about open source projects. The FLOSSmole collection methods are transparent and easily reproduced, so FLOSSmole can serve as a reliable resource for these metrics. Having a stable and consistently updated source of this information will also allow metrics to be compared over time. One of the problems with existing analyses of open source project data is that researchers will run a collection and analyze it once, publish the findings, and then never run the analysis again. The FLOSSmole data model and collection methodology was designed to support historical comparisons of this kind.

FLOSSmole data were used in a number of large-scale social network analyses of FLOSS project development. Crowston and Howison (2004) report the results of a SNA centralization analysis in which the data suggest that, contrary to the rhetoric of FLOSS practitioner-advocates, there is no reason to assume that FLOSS projects share social structures. Further FLOSSmole data were used in the preparation of Crowston,

Howison, and Annabi (2006) which, in an effort to avoid the ambiguities of relying on ratings or downloads, develops a range of quantitative measures of FLOSS project success including the half-life of bugs. FLOSSmole makes available the full data and analysis scripts, which make these analyses fully reproducible, and, we hope, extendable.

FLOSSmole data were also used in a recent exploration of whether open source development teams have characteristics typical of a complex network (Conklin, Howison, & Crowston, 2004). This research investigated whether FLOSS development networks will evolve according to "rich get richer" or "winner take all" models, as other self-organized complex networks do. Are new links (developers) in this network attracted to the largest, oldest, or fittest existing nodes (project teams)? The FLOSSmole data were used to determine that there are indeed many characteristics of a complex network present in FLOSS software development, but that there may also be a mutual selection process between developers and teams that actually stops FLOSS projects from matching the "winner take all" model seen in many other complex networks.

Projects of a nonacademic nature are making use of FLOSSmole data as well. The Swik project from SourceLabs[8] is a wiki-driven system for managing facts about other open source software projects. Swik uses FLOSSmole data to populate its initial list of projects. Working independently, the Swik team was able to download FLOSSmole data and put them to use immediately to save time and effort during their development process. By using a dataset that was freely available and for which the provenance of all data was known and validated, Swik was able to accelerate their development cycle.

LIMITATIONS AND FUTURE WORK

There are, of course, limitations in the FLOSSmole project and in our approach. First, we are limited to collecting data available online, and we are limited to collecting data gathered as a direct result of documented project activities. Of course, electronically documented project activities are not the only interactions FLOSS team members have, and even these activities are not always available for perusal by outside parties. Thus while textual data like mailing lists, CVS comments, Forums, and IRC chat logs could be included[9], FLOSSmole does not aim to capture unlogged instant messaging, IRC, Voice-over-IP, or face-to-face interactions of FLOSS developers. Nor do we intend to store interviews or transcripts conducted by researchers that would be restricted by research ethics policies.

There are also dangers in this approach that should be acknowledged. The standardization implied in an academic repository, while valuable, runs the risk of reducing the valuable diversity that has characterized academic FLOSS research. We hope to provide a solid and traceable dataset and basic analyses that will support, not inhibit, interpretative and theoretical diversity. This diversity also means that research is not rendered directly comparable simply because analyses are based on FLOSSmole data or scripts; the hard intellectual work remains and hopefully FLOSSmole, by supporting baseline activities, leaves us more time for such work.

It is quite likely that a functional hierarchy could develop between cooperating projects, something akin to the relationship between FLOSS authors and distributions, such as Debian or Red Hat and their package management systems (i.e., apt and rpm). For example, such an arrangement would allow groups to specialize in collecting and cleaning particular sources of data and others to concentrate on their compatibility. Certainly, we expect that the existing communities of academics

interested in FLOSS, such as opensource.mit.edu, will be a source of data and support.

Finally, we must also consider privacy issues. There is some discussion in the research community about breaching developer privacy in a large system of aggregated data like ours (Robles, 2005), specifically in terms of uniquely identifying developers and analyzing their work products. FLOSSmole should have the ability to hash the unique keys indicating a developer's identity. This effort will have to be researched, implemented, and documented for the benefit of our community.

CONCLUSION

Researchers study FLOSS projects in order to better understand collaborative human behavior during the process of building software. Yet it is not clear that current researchers have many common frames of reference when they write and speak about the open source phenomenon. As we study open software development, we learn the value of openness and accessibility of code and communications; FLOSSmole is a step towards applying that to academic research on FLOSS. It is our hope that by providing a repository of traceable and comparable data and analyses on FLOSS projects, FLOSSmole begins to address these difficulties and supports the development of a productive ecosystem of FLOSS research.

ACKNOWLEDGMENT

This research was partially supported by NSF Grants 03-41475, 04-14468, and 05-27457. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors thank the FLOSS developers who contributed to the research.

REFERENCES

Conklin, M. (2004, July 30). Do the rich get richer? The impact of power laws on open source development projects. In *Proceedings of the Open Source Conference (OSCON),* Portland, Oregon.

Conklin, M., Howison, J., & Crowston, K. (2005, May 17). Collaboration using OSSmole: A repository of FLOSS data and analyses. In *Proceedings of the Mining Software Repositories Workshop (MSR2005) of the 27th International Conference on Software Engineering (ICSE 2005),* St. Louis, Missouri.

Crowston, K., Howison, J., & Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and practice, 11*(2), 123148.

Crowston, K., & Howison, J. (2004). The social structure of open source software development teams. *First Monday, 10*(2). Retrieved from http://firstmonday.org/issues/issue10_2/crowston/.

Howison, J., & Crowston, K. (2004). The perils and pitfalls of mining sourceforge. In *Proceedings of the Mining Software Repositories Workshop at the International Conference on Software Engineering (ICSE 2004),* Edinburgh, Scotland.

Krishnamurthy, S. (2002). Cave or community? An empirical examination of 100 mature open source projects. *First Monday, 7*(6). Retrieved from http://www.firstmonday.org/issues/issue7_6/krishnamurthy/

Robles, G. (2005, May 17). Developer identification methods for integrated data from various sources. In *Proceedings of the International Workshop on Mining Software Repositories (MSR2005) of the 27th International Conference on Software Engineering (ICSE2005),* St. Louis, Missouri.

Weiss, D. (2005). Quantitative analysis of open source projects on SourceForge. In *Proceedings of the First International Conference on Open Source Systems (OSS 2005)*, Genova, Italy.

Xu, J., Gao, Y., Christley, S., & Madey, G. (2004). A topological analysis of the open source software development community. In *Proceedings of HICSS 2005*, Hawaii.

ENDNOTES

1. FLOSSmole: http://ossmole.sf.net
2. Sourceforge: http://www.sf.net
3. Savannah: http://savannah.gnu.org
4. CodeHaus: http://www.codehaus.org
5. Comprehensive Perl Archive Network: http://www.cpan.org
6. http://www.nd.edu/~oss/Data/data.html
7. FLOSSmole: http://ossmole.sf.net
8. Swik: http://swik.net/
9. http://libresoft.urjc.es/Activities/WoP-DaSD2006

This work was previously published inInternational Journal of Information Technology and Web Engineering, Vol. 1, Issue 3, edited by E. Damiani; and G. Succi, pp. 17-26, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 1.10
Free/Libre Open Source Software for Bridging the Digital Divide

Yu-Wei Lin
University of York, UK

INTRODUCTION: THE FORGOTTEN USERS IN SOFTWARE DESIGN

As some scholars claim, the digital divide, referring to the perceived gap between those who have access to the latest information technologies and those who do not, entails that not having access to this information is an economic and social handicap (Compaine, 2001). In software design, structured inequalities operate along the main axes of gender, race/ethnicity and class. Each of these in turn generates its own structure of unequal practices giving rise to institutionalised sexism, racism or class divisions/conflict. "Gender, race and class also crosscut each other in various complex ways, sometimes reinforcing and at other times weakening the impact of existing inequalities" (Cohen & Kennedy, 2000, p. 100). For instance, Webster's research (1996) employing feminist approaches to study computer system designs addresses the issue of a male-dominated system design field, which continuously excludes female users' needs, requirements, interests and values in the innovation process. She criticises that, "Human factors may be bolted onto existing methods of systems design, local and contingent knowledge of work and information handling processes held by users in an amorphous sense may now even be incorporated into the systems design process, but this does not create an awareness of the way in which skills and knowledge are defined in gender-divided terms" (p. 150).

In a similar course, I argue that users' experiences in developing or undeveloped countries are often ignored in mundane software designs led by developed countries. Although localisation of information infrastructure is an eminent issue emerging in current system development, profit-oriented products and services, such as Microsoft's local language program (LLP), do not really comply with local needs. Rather, this type of multi-languages software packages, a software suite fabricated universally for countries around the world, signify the phenomenon, which I term the "MacDonaldisation of Windows-Intel platforms," which in fact alienates users and the local contexts.

FLOSS IN ACTION

In recent years, free/libre open source software (FLOSS) has emerged as an important phenomenon in the information communication technology (ICT) sector as well as in the wider public domain. An increasing number of governments have endeavoured to either convert the public administration infrastructure from Windows to Linux or to adopt FLOSS for similar tasks (e.g., Munich in Germany or Zaragoza in Spain) (c.f., "Linux in Spain" on LWN.net; C|Net News.com August 29, 2001). FLOSS transparentises the often black-boxed software code and allows users to copy, distribute and modify a programme received freely. In making source code available, software technologies can be challenged, adapted, and ameliorated to satisfy diverse user needs. Apart from solving the prolonged usability1 problem in software engineering, implementing FLOSS also helps ground both social and technical knowledge in locales and bridge the digital divide. In other words, implementing FLOSS facilitates technical knowledge (e.g., programming skills and ICT expertise) and social experiences to be transported and transferred through the acclaimed practices of social networking and mutual help noted prominently in many recent community studies (e.g., Wellman, 1999; Rheingold, 2000; Hampton & Wellman, 2003; Jordan et al., 2003; Lin, 2004a).

There have been a number of tactical considerations of implementing FLOSS in countries or organisations devoid of intellectual or financial resources: for economic reasons to save software costs; for educational reasons to improve human resource; for political reasons to stop monopoly proprietary software from expanding their market share as well as to gain digital autonomy, just to name a few. Hence, it is a strategic interplay for local governmental or non-governmental organisations (NGOs), and FLOSS activists to coalesce to tackle these inequalities. Because knowledge transfer is as crucial as infrastructure implementation, hands-on training made available to the local users is essential in the execution. Projects such as the E-Riders2 and Low Income Networking and Communications3, or events such as the Summer Source Camp4 and Africa Source5, all illustrate the transfer of knowledge and technology across cultural boarders. These examples also show how the implementation of FLOSS shapes the lives and identities of local users as well as software developers around the globe (Lin, 2004b). Additionally, there is conspicuous implementation of Linux-based infrastructure in the local educational, NGO and governmental organisations in developing countries or regions (e.g., Washington Post, November 3, 2002). The advents of embedded technologies such as the "Simputer," a Linux handheld applied in India, are believed to enable affordable, sustainable village development in places without phones and power, giving more and more people a voice in the conversation about their future (Cherlin, 2002). Wireless technologies are amongst others to bring the Internet to developing countries or regions to facilitate networking at both local and global levels. Krag, a Danish expert of wireless technologies who I met at the 2003 summer source camp in Croatia, describes wireless technologies as low-cost and decentralised. Here is a quote from his talk at the O'reilly 2004 emerging technology conference6 about the advantage of wireless technologies:

> Billions of people in the world have never been online. The Internet as a technology is an elitist tool, reserved for the few and unreachable by the many. This is a problem not likely to be solved by the commercial interests of existing telecommunications companies and existing ideas about expensive, centralized infrastructure. But low-cost, decentralized wireless technologies could have an important role to play, in this respect. Their low price point and decentralized nature, and the openness of the standards, mean that these technologies are incredibly adaptable to new situations and new uses. (Krag, 2004)

Krag and his colleagues have been working in undeveloped/developing countries around the world, building up and promoting wireless technologies (mainly 802.11b standard, also known as WiFi) for the locals. They bring the Internet and intranet connectivity to those parts of the world not included in the plans of the commercial telecommunications companies. They teach and give hands-on training to the locals about how to use ICT, and at the same time build wireless networks in the countries they visit. In so doing, they hope to "not only raise awareness and heighten skill sets, but also gain the experience necessary to build a central repository of documentation and tools, targeted specifically at the developing world" (ibid.). Krag's words subtly show that working with the locals de facto benefits the legitimate knowledge system of wireless technologies by means of bringing in more empirical cases that illustrate how infrastructure can be turned into applications, and how experiments can be turned into existing proofs. That said, working in undeveloped/developing areas in fact is not a one-way giving episode, rather, it is a reciprocal process that involves mutual help and mutual learning. Sometimes, extra functions are endorsed to the original products or facilities to meet local users' needs or habits ad hoc. The local contingencies entail that products and facilities can be renovated with the new functions and features after being deployed in local environments. This close link with global and legitimate knowledge sets also suggests that local expertise is worth being documented or transcribed in order to understand the construction of a knowledge-based society from both macro and micro levels.

DEMOCRATISING SOFTWARE INNOVATION PROCESS: WHOSE DEMOCRACY?

Given the advantages of FLOSS, it is believed that FLOSS is seen as an effective tool and approach to tackle the digital divide. Although FLOSS provides more flexibility and economic good for local users, solutions are mostly identified and crafted from the point of view of the developed countries. The collaborative episodes between NGO and FLOSS sectors illustrated above show an ambition to mobilise awareness and participation, and build capacity. However, when taking the local requirements into account, the social problem of the digital divide is perceived merely in the eyes of some NGO and FLOSS workers, rather than derived from the locals. Whereas a FLOSS-based solution seems to bridge the digital divide more efficiently than proprietary software, it sometimes still ignores that the political-cultural position of the locals and does not automatically move towards the centre of the global society. The cultural differences between the outsiders and insiders at the locale influence which perspective in a FLOSS implementation (e.g., economical, educational, social, political, technical) should be prioritised. This decision is a tactic because in the decision-making process, NGO and FLOSS activists all identify and interpret the social problem from their point of view. For FLOSS and NGO activists, introducing and implementing FLOSS denotes a cultural shift of networking the local with the global. They believe that ideas and knowledge are the cosmopolitan valuables, and community building and social networking are the most effective means to engage these social capitals that can be amplifiers in an innovation system. However, the social problems have different meanings to the local. In some places, freedom of information is not the priority. Instead, to endorse the local economic purchase power is on top of any other concerns. Without taking the local interests into account, the design of the information infrastructure, which is out of context, is likely to be lost in translation. That said, the social problems addressed, presented and represented differently by NGO and FLOSS activists in each local context, are better seen as articulations of the communication and move-

ments between different forms of knowledge and cultural practice. In short, "who provides what for whom" is a crucial question when studying the implementation of FLOSS. There is a dilemma of "translation from whom?"

This dilemma has gradually been noticed by some FLOSS activists who have devoted much time to implementing FLOSS infrastructure in undeveloped/developing countries. In the case of wireless technologies, it is true that wireless technologies are relatively cheap and have all of the technical advantages of simplicity, openness, decentralisation and autonomy. However, when the digital divide is perceived as a social problem for NGO and FLOSS activists, it has actually produced a subtle prejudice invisible to the digital elite. Indeed, some FLOSS spokespersons have shown different opinions on mainstream media policy that "often take the technological configuration of the new media as a 'given' or prefigured system that needs to become more widely diffused to citizens" (Mansell, 2002, p. 408). However, as Mansell also points out, arguments such as Lessig's research (2000, 2001) "does not examine the rhetorical forms that help to sustain the configurations of the new media that are favoured by an influential minority of technology developers and producers" (op. cit.). It is likely that the implementation of FLOSS conversely deteriorates the opposition between the rich and the poor because the poor cannot perceive the digital divide as an urgent problem. It would be a failure of not being able to correspond the local's interests and encourage their motivations.

Is FLOSS a silver bullet for bridging the digital divide? It could be. Compared with other ICT based on proprietary software, a FLOSS-based implementation does provide a better approach to bridging the digital divide from many aspects, particularly in terms of mapping and incorporating local users' requirements and lowering their ICT purchase cost (Lin, 2004b; Wheeler, 2004). However, the degree of empowerment may be counteracted if local users' interests are not fully included in the implementation agenda. So-called "digital independence" and "participatory democracy" would be just a proclaimed allegation in the FLOSS hype, as they are in most ICT advocacy. Having said that, it is not my intention to deny the advantages of FLOSS and its socio-technical impacts. But I would like to point out some blind points remaining in its world wide implementation and deployment. There has been a tendency in the development of ICT to facilitate human activities with more ubiquitous and efficient infrastructure and interfaces to meet users' needs in various areas. Whilst this vision of civilisation sounds promising, the accentuation on ICT's positivity also leads to all manner of delusions and falsehoods. When a newly invented ICT product is praised for its influential potential to empower consumers, it is yet invalid without asking whether a product is usable and useful to users, or without understanding users situated in different cultural and social contexts. This focus on the infrastructure building has misled both the public and the private sectors to believe a set of dominant and monolithic values, such as "efficiency," "modernity," and "improvement" (in the context of the "state-of-the-art"). Users are imposed to accept explicitly or implicitly these values which are fabricated and fortified to meet interests of specific social groups. A solitary identity and socio-technical class is produced to categorise users: to be connected/wired or to be alienated from the digital world; to be mobile or to be outmoded; to be electronic or to be antediluvian. Diverse meanings of "innovation" thus are muted in this dominant socio-technical context. Modern users of ICT de facto have lost their socio-cultural identities, which are grounded in everyday lives. The digital divide issue is also lost in this hype that ICT is the vehicle for democratic participation.

Whilst the emergence of FLOSS broadens the digital divide issue from a level of access to information facilities (hardware) to information contents (software), this extended scope remains

the preference of a small group of digital elite. In this regard, I share the opinion of Thomas and Wyatt (2000) that access is not the only important issue for understanding inequality. Instead, it is the assumptions on which diffusion of Internet connection is seen as necessarily expanding and beneficial that should be questioned. In a sociological term, it is more essential to understand the meaning of "access" to the Internet that is interpreted and appreciated differently by different social groups across geo-cultural borders. A crucial perspective to be developed while studying the increasingly prominent FLOSS phenomenon is to examine critically the role of the intellectual and their interactions with the local. It requires a careful examination on the conceptualisation and organisation of resistance to taken-for-granted realities and dominant forms of social organisation, which often goes together with a desire for new technologies. The so-called "digital divide" should be understood as a situated digital divide.

CONCLUSION: TRANSFERRING LOCAL EXPERTISE INTO A GLOBAL KNOWLEDGE SYSTEM

This article argues that FLOSS helps developing regional communities by improving ICT capacity and empowering users. FLOSS provides local governments and organisations a shared code base with higher level of security and a dramatic reduction of development costs. FLOSS bridges the digital divide in lessening dependency on proprietary software and securing the potential for developing local relevant software. However, without adequate expertise, after opening up the black box of software technologies, users are still left in a feeble situation and strongly depedent on experts. This is why hands-on training and a well-networked community providing mutual help are necessary in the implementation process, so that participants can learn to read the code, manipulate the code, and work with the code. Here, "the code" does not only indicate software source code, but is more generically referred to the code of knowledge. Whilst open-source code is decoded, so is the formal expertise deciphered. Consequently, software technologies can be challenged, adapted, and ameliorated to satisfy diverse user needs and solve the prolonged usability problem. Nevertheless, this problem will not be solved in a short time. Whilst open source code can be exploited and employed by experts programmers for individual needs, for users, unless they own or access to the expertise as well, the knowledge does not transfer onto them straight away after source code is released. That said, the empowerment of technology is not a natural process. It requires skills and experiences dealing with and translating the rich information into various languages which facilitate wider accessibility to the core of the knowledge system. I suggest that it is crucial to coalesce the local cultures and the technologies and to improve the dialogues between FLOSS developers, implementors and the locals in order to facilitate the implementation process. In so doing, knowledge-deciphering and knowledge-transferring will less likely to get lost in translation. ICT knowledge thus can be more appropriately transferred to and relocated in locals' daily lives. The global problem of usability, hence, can be tackled through the grassroots social networking.

REFERENCES

C|Net News.com. (2001, August 29). Governments push open source software. Retrieved May 30, 2004, from http://news.com.com/2100-1001-272299.html?legacy=cnet

Cha, A.E. (2002, November 3). Europe's Microsoft alternative: Region in Spain abandons Windows, embraces Linux. Washington Post, A01.

Cherlin, E. (2002). Simputer white paper: Computers for everyone. Retrieved May 30, 2004, from

http://www.upspace.org/academy/Members/cherlin/Simputer%20White%20Paper/file_view/

Cohen, R., & Kennedy, P. (2000). Global sociology. London: MacMillan Press Ltd.

Compaine, B. M. (Ed.) (2001). The digital divide: Facing a crisis or creating a myth? Cambridge, MA: MIT Press.

Hampton, K., & Wellman, B. (2003). Neighboring in Netville: How the Internet supports community, social support and social capital in a wired suburb. City and Community, 2(4), 277-311.

Jordan, K., & Hauser, J., & Foster, F. (2003). The augmented social network: Building identity and trust into the next generation internet. First Monday, 8(8). Retrieved September 1, 2004, from http://firstmonday.org/issues/issue8_8/jordan/index.html

Krag, T. (2004). Wireless networks as a low-cost, decentralized alternative for the developing world. A talk given at the O'reilly emerging technology conference, 9-12 February 2004, San Diego, USA. Retrieved May 30, 2004, from http://conferences.oreillynet.com/cs/et2004/view/e_sess/4697

Lessig, L. (2000). Code and other laws of cyberspace. New York: Basic Books.

Lessig, L. (2001). The future of ideas: The fate of the commons in a connected world. New York: Random House.

Lin, Y.-W. (2004a). Epistemologically multiple actor-centered systems: Pr, EMACS at work! Ubiquity, 5(1). Retrieved September 1, 2004, from http://www.acm.org/ubiquity/views/v5i1_lin.html

Lin, Y.-W. (2004b). Has software development ever been lost in translation? From local epistemologies to cosmopolitan expertise. A paper presented at the 4S-EASST joint conference, 25-28 August 2004, Paris.

LWN.net. Linux in Spain. (n.d.). Retrieved September 1, 2004, from http://lwn.net/Articles/41738/

Mansell, R. (2002). From digital divides to digital entitlements in knowledge societies. Current Sociology, 50(3), 407-426.

Rheingold, H. (2000). The virtual community: Homesteading on the electronic frontier (revised edition). Cambridge, MA: MIT Press.

Thomas, G., & Wyatt, S. (2000). Access is not the only problem: Using and controlling the Internet. In S. Wyatt, F. Henwood, N. Miller, & P. Senker (Eds.), Technology and In/Equality: Questioning the information society. London: Routledge.

Webster, J. (1996). Shaping women's work: Gender, employment and information technology. London: Longman.

Wellman, B. (Ed.) (1999). Networks in the global village. Boulder, CO: Westview Press.

Wheeler, D. A. (2004). Why open source software/free software (OSS/FS)? Look at the numbers! Retrived September 1, 2004, from http://www.dwheeler.com/oss_fs_why.html

KEY TERMS

Digital Divide: The digital divide is a social issue referring to the differing amount of information between those who have access to computers and the Internet and those who do not.

FLOSS: Free/Libre Open Source Software (FLOSS), generically indicating non-proprietary software, combines the concepts of free software and open source software. It makes it easier to talk about one movement and not ignore the other, and as such, can be used as a compromise term palatable to adherents of either movement. It also emphasises the libre meaning of the word "free" rather than the "free of charge" or gratis meaning which those unfamiliar with the subject

might assume. This all-inclusive acronym has the extra advantage of being non-anglo-centric: the F stands for Frei in German while the L stands for Libre in French and Spanish, Livre in Portuguese and Libero in Italian, showing that the concepts and their implementation are not exclusive to the English-speaking world.

ICT: Information and communication technology (ICT) is the technology required for information processing. In particular the use of electronic computers and computer software to convert, store, process, transmit, and retrieve information.

NGO: A non-governmental organisation (NGO) is an organisation which is not a part of a government. Although the definition can technically include for-profit corporations, the term is generally restricted to social and cultural groups, whose primary goal is not commericial. Generally, although not always, these are non-profit organisations (NPO) that gain at least a portion of their funding from private sources.

Social Networking: Social networking describes the process of connecting individuals via friends, relatives, and acquaintances. These networks can then branch out and allow friends to connect with people inside their accepted social circle.

Usability: Usability addresses the full spectrum of impacts upon user success and satisfaction. Usability is accomplished through user-centreed (not necessarily user-driven) design. The usability engineer provides a point-of-view that is not dependent upon designers' goals because the usability engineer's role is to act as the users' advocate.

User-Centred Design: User Centered-Design (UCD) is a software process that seeks to answer questions about users and their tasks and goals, then use the findings to drive development and designs, and improve the usability and usefulness.

ENDNOTES

1 Here, "usability" serves as an umbrella term that covers generic human-machine interactions (HCI) issues as well as wider accessibility and the digital divide problems.
2 http://www.eriders.net/
3 http://www.lincproject.org
4 http://www.tacticaltech.org/summersource
5 http://www.tacticaltech.org/africasource
6 http://www.oreillynet.com/et2004/

This work was previously published in Encyclopedia of Developing Regional Communities with Information and Communication Technology, edited by S. Marshall; W. Taylor; and X. Yu, pp. 316-320, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 1.11
Community of Production

Francesco Amoretti
Università degli Studi di Salerno, Italy

Mauro Santaniello
Università degli Studi di Salerno, Italy

INTRODUCTION

There is no universal agreement regarding the meaning of the term "social software." Clay Shirky, in his classic speech "A Group is its Own Worst Enemy," defined social software as "software that supports group interaction" (Shirky, 2003). In this speech, this scholar of digital culture also observed that this was a "fundamentally unsatisfying definition in many ways, because it doesn't point to a specific class of technology."

The example offered by Shirky, illustrating the difficulties of this definition, was electronic mail, an instrument that could be used in order to build social groups on the Net, but also to implement traditional forms of communication such as broadcasting, or noncommunicative acts such as spamming. In his effort to underline the social dimension of this phenomenon, rather than its purely technological aspects, Shirky decided to maintain his original proposal, and this enables scholars engaged in the analysis of virtual communities to maintain a broad definition of social software. Heterogeneous technologies, such as instant messaging, peer-to-peer, and even online multigaming have been brought under the same conceptual umbrella of social software, exposing this to a real risk of inflation. In a debate mainly based on the Web, journalists and experts of the new media have come to define social software as software that enables group interaction, without specifying user behaviour in detail. This approach has achieved popularity at the same pace as the broader epistemological interest in so-called emergent systems, those that, from basic rules develop complex behaviours not foreseen by the source code (Johnson, 2002). This definition may be more useful in preserving the specific character of social software, on the condition that we specify this carefully. If we include emergent behaviour, regardless of which Web technologies enter into our definition of social software, we will once again arrive at a definition that includes both everything and nothing. Emergence is not to be sought in the completed product, that may be unanticipated but is at least well-defined at the end of the productive cycle, but rather resides in the relationship between the product, understood as a contingent event, and the whole process of its production and reproduction. A peculiar characteristic of social software is that, while allowing a high level of social interaction on the basis of few rules, it enables the immediate re-elaboration of products in further collective cycles of produc-

tion. In other words, social software is a means of production whose product is intrinsically a factor of production. Combining hardware structures and algorithmic routines with the labour of its users, a social software platform operates as a means of production of knowledge goods, and cognitive capital constitutes the input as well as the output of the process.

If a hardware-software system is a means of production of digital goods, social software represents the means by which those products are automatically reintroduced into indefinitely-reiterated productive cycles. This specification allows us to narrow down the area of social software to particular kinds of programmes (excluding, by definition, instant messaging, peer-to-peer, e-mail, multiplayer video games, etc.) and to focus the analysis on generative interaction processes that distinguish social software from general network software. Moreover, following this definition, it is possible to operate a deeper analysis of this phenomenon, introducing topics such as the property of hosting servers, the elaboration of rules and routines that consent reiterated cycle of production, and the relationships between actors within productive processes.

NORMATIVE EVOLUTION OF THE INTERNET FROM NET95 TO WEB 2.0

At the end of the 1990s, two particular events gained wide social significance in the evolution of global telecommunications networks. First, a deep restructuring of the fundamental architecture of the Internet radically transformed the network which had been born in the ARPA laboratories. Coming out of a rather narrow military and academic sphere, the Internet became at once easier and more complex. Graphical user interface (GUI) principles simplified computer and database management for a growing mass of individuals who were ready to get connected, giving birth to a vigorous codification of intermediate zones between man and machine. Operating systems, appliances, software, automatic updates: the popularisation of the Net has proceeded through a constant delegation of terminal management from the user to the software producer, and in the case of software distributed under the juridical instrument of the "license of use," this delegation consists in the property of parts of the "personal" computer. The American constitutionalist Lawrence Lessig, underlining the social relevance of this phenomenon, describes the original network (Net95) as being completely twisted, subject to the control of those coding authorities that, since 1995, have reconfigured the architecture of cyberspace (Lessig, 1999).

The second event that has contributed to the morphology of the transformation of new information technologies is a direct consequence of the first, and concerns contents, or products, of this new kind of network. The more Internet has been regulated by a wide group of code writers, among which a narrow circle of economic players who have assumed positions of power, the more personal relationship networks based upon it have assumed social significance and cultural reach. Ease of computer management and development of applications that allow social interaction in an intuitive and simple manner have brought blogs, wiki, syndication and file-sharing platforms to the scene. The growing use of these Internet-based applications, a result of the convergence between the regulation and socialisation of cyberspace, has slowly attracted the attention of political and economic organisations due to its capacity to allow a widespread and participative use of digital goods and knowledge. So, the first realisation of Web 2.0 (or semantic Web) was the product of a normative process operated for the most part by software engineers, a product which, arriving later, attracted the attention of the social sciences, which tended to view the means of that production as black boxes. In this period, between the last two centuries, besides more profound reflection on

emerging social systems based on social software platforms, there was scarce interest in the structures and infrastructures of telecommunications networks, understood as plentiful resources, never as a real means of production.

COMMUNITY OF PRODUCTION

The Dual Reality of Social Software

In the second half of 2006, the Web site called YouTube reached popularity and suddenly burst into the Top 10 of most visited Web sites (Alexa, 2006). Just 18 months after its launch, this platform for video-sharing reached an average of 60 million visits per day, with hundreds of thousands of clips being viewed each day in streaming and 65,000 new clips uploaded every 24 hours. YouTube's success may be explained, above all, by reference to the scarce resources that, despite the arguments of "plenty infrastructure" theorists, characterise Internet as another complex system of means of production. Having no resources to host and transmit a great quantity of video information, millions of individuals use YouTube's resources in order to reach the same goal: to share videos. Uploaded clips, moreover, are immediately available for other, undefined forms of production of meaning, that work mainly through three channels. The first involves categorising videos using labels (called tags), which are filled in by the user who uploads them. In this way, both search and correlations between videos are based on a *folksonomy* system that is able to generate bottom-up relationships. Second, users can express a judgement on the video's quality and relevance to the description, providing single clips with a "reputation." Third, a function of the platform is its capacity to supply code that, pasted in a Web page or a blog, executes the corresponding clip in a streaming player. This last function is largely used by publishers who, not having the required resources to afford streaming connections, use YouTube to retransmit their own digital productions. Another social software platform with similar characteristics is Flickr. In contrast to YouTube, this allows photo, not video, sharing. The greater number of visits for YouTube when compared to Flickr (which is estimated to be the 38th most visited site in the world, with 12 million contacts per day), is partly attributable to the greater quantity of data necessary to code videos rather than photos, that is to produce film transmissions rather than photo-galleries over the Web. The more storage and bandwidth resources required, the more users need and use the platform. This kind of service is without charge, but is not without economic relevance for the organisations that manage it. Beside traditional sponsorship, both Flickr and YouTube largely use content-targeting advertisement, a technology that allows one to scan texts and automatically associate them with ads. Advertisers buy keywords, and when a keyword is found in a text, an associated advertisement is displayed near the text. If the announcement is clicked, the advertiser pays a sum divided between the firm that manages the content-targeting technology and the owner of the Web page upon which the ad was hosted. Gathering content-targeted advertisement is substantially a duopoly, provided by Google and Overture, the latter now being fully controlled by Yahoo. Flickr and YouTube, on the other hand, are the owners of Web pages that host texts produced by their users, and receive the remaining part of the share paid by the advertiser. Thousands of dollars daily move toward the bank accounts of social software platform owners thanks to the work of millions of volunteers who produce texts and links which themselves have economic value.

The concept of the user extends from the sphere of service enjoyment to the sphere of production, to the detriment of the concept of the worker. And if industrialisation proceeded via the eradication of workers' communities (Polanyi, 1944), digitalisation today appears to produce value through the creation and management of delocalised communities of volunteers. This phe-

nomenon assumes such an economic importance in the Internet advertisement sector that not long ago Yahoo bought Flickr to deliver its ads by its own, followed by Google, that recently completed the purchase of YouTube, of which it was already a supplier of content-targeted advertisements. The same fate was reserved for del.icio.us, a social software platform for Web site addresses, taken over by Yahoo. This situation means that, rather than sharing the revenues of content-targeting with these platforms, the two key players in the market preferred to tackle multimillion dollar costs in order to implement processes of vertical organisation, in a market that looked increasingly polycentric within a few years' of its opening. This dynamic, moreover, implies that the overall value of digital commodities produced on social software platforms represents a first order economic entity in the so-called new economy.

EVOLVING SCENARIOS: FROM WEB 2.0 TO INTERNET 2

Besides *folksonomies*, social software platforms can be based on another principle: real time collaborative editing (often called *wiki*). Users of this kind of platform can produce text and modify others' texts in real time, generating processes of collective and negotiated production of meaning. The main actor in this particular segment is the Web-based encyclopaedia Wikipedia. Founded in 2001 in the United States, it quickly entered the 20 most-visited Web sites in the world, with an average of just under 60 million visits per day. At the present time, it hosts 5 million definitions in 229 different languages and idioms. In 2005, the scientific magazine Nature conducted a comparative analysis of Wikipedia and Encyclopaedia Britannica, reaching the conclusion that the two are substantially equivalent in accuracy and trustworthiness, as far as the Natural Sciences are concerned (Giles, 2005). Unlike the previously described social software platforms, Wikipedia doesn't use paid advertisements, being funded by donations gathered by the Wikimedia Foundation, a nonprofit American organisation at whose summit is a five-member Board of Trustees. With a quarterly budget that doesn't exceed $400,000, and with the explicit goal of producing independent and free knowledge, Wikipedia expressly refuses the commodification of its knowledge-based output. But rather than representing the ideal type of social software, this constitutes an exception, whose survival is threatened by evolving scenarios of telecommunication infrastructures. Parallel to the bottom-up categorisation project of digital goods in the Web, there is a categorisation project that concerns not only Web content, but even its own infrastructures. AT&T, Verizon, Comcast, and other enterprises that manage global flows of information have been complaining for several years about the lack of return on their investments in telecommunication systems.

So-called *network neutrality*, the principle that Isenberg referred to as "netstupidity" during the early 1990s, does not allow carriers to distinguish between the functions of the bits of information that they transport; that is, whether they are constitutive of video, e-mail, phone calls, Web surfing, or whatever. This prevents them from applying specific fees according to a service typology that varies on the basis of the media goods moved by these actors and the resources employed. Telco's interests, which are also connected with the traditional phone networks, are clearly damaged by the architecture of their own infrastructure, but cannot prevent users from sidestepping traditional communication fees. If we consider how these interests are converging upon those—even wider—of the main owners of intellectual properties, we can understand how the architectural reconfiguration of Net infrastructures have reached an advanced state of both planning and implementation. Attempts to protect network neutrality by legal means, through the introduction of explicit limits in the U.S. telecommunications legislation, have so

far failed, and continue to generate transversal opposition within Congress (Senate Commerce Committee, 2006. HR.5252).

The Internet 2 project, which alongside the semantic Web introduces an intelligent infrastructure that is able to categorise digital commodities in the Net in a top-down manner, evokes a scenario in which Web sites could be obliged to pay for their visitors, and where fees are proportional to bandwidth used, as well as to the typology of enlivened media commodities. This kind of evolution could have a profound impact on those rare social software platforms exclusively supported by donations from civil society, thus likely to augment the probability of a market opening alongside them in the long term.

CONCLUSION

Migration toward cyberspace presents the same dual aspect that Russel King has observed in relation to mass migration. It creates particular forms of social relationships within a space: both social relations of capitalist production and personal social relations that reproduce migration chains over time (King, 1995).

It is of particular significance that, attempting to shed light on the dual nature of migration trends, King sought to recalibrate the analysis of human movements, introducing the theme of personal relations beside the broader one involving relations of production. In the case of migration toward cyberspace, on the other hand, the area of social relationships was the first to attract the interest of the social sciences, leaving political economy themes to the mercy of the theorists of the new economy.

Social software, one of the outcomes of half-century-long innovation processes in the cybernetic area, highlights the connections between social and production relationships implied in the digitalisation of human interactions. Delocalised human resources, mostly constituted by nonworkers, hardware and software means of production, are more and more concentrated in interconnected oligopolies, cycles of production whose routines accumulate value directly from working activities not remunerated, in same cases even paid by *workuser* to means of production's owner.

Social software seems to adhere to a comprehensive process concerning both the infrastructure of the Net, with its digital commodity flows, and relations of production within a system whose peculiarity is to be equipped with a unique space where production, trade, and finance can simultaneously coexist. Bridled in such a net-space, communities producing value produce social behaviours that, beyond their anthropological significance, assume a substantial economic value. Immediate value extraction by processes generated by noneconomic motivations represent one of most important aspects of the market-based reorganisation of information technologies and, from this point of view, social software could become one of the main arenas for the new relations emerging between capital, space, and labour in the current historical conjunction.

REFERENCES

Alexa Internet Inc. (2006). *Traffic rankings*. Retrieved April 24, 2007, from http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none

Andreson, C. (2004). *The long tail*. Web site Wired Magazine. The Condé Nast Publications Inc. Retrieved April 24, 2007, from http://www.wired.com/wired/archive/12.10/tail_pr.html

Battelle, J. (2005). *The search. How Google and its rivals rewrote the rules of business and transformed our culture*. New York: Penguin Books.

Bourdieu, P. (1977). *Outline of a theory of practice*. Cambridge University Press.

Giles, J. (2005). Internet encyclopaedias go head to head. *Nature, 438*(12), 900-901.

Isenberg, D.S. (1997, August). The rise of the stupid network. *Computer Telephony,* 16-26.

Johnson, S.B. (1997). *Interface culture. How new technology transforms the way we create and communicate.* New York: Basic Books.

Johnson, S.B. (2002). *Emergence: The connected lives of ants, brains, cities, and software***.** New York: Touchstone.

Kelly, K. (1998). *New rules for the new economy: 10 radical strategies for a connected world.* New York: Penguin Books.

King, R. (1995). Migrations, globalization and place. In D. Massey & P. Jess (Eds.), *A place in the world? Places, cultures and globalization.* Oxford: The Open University.

Lessig, L. (1999). *Code and other laws of cyberspace.* New York: Basic Books.

Levine, F., Locke C., Searls, D., & Weinberger, D. (2000). *The Cluetrain Manifesto: The end of business as usual.* New York: Perseus Book.

Polanyi, K. (1944). *The great transformation: The political and economic origins of our time.* Boston: Beacon Hill.

Senate Commerce Committee (2006). Advanced Telecommunications and Opportunities Reform Act (HR.5252). Senate of the United States. 109th Congress 2D Session.

Shapiro, A.L. (2000). *The control revolution: How the Internet is putting individuals in charge and changing the world we know.* USA: Public Affairs/The Century Foundation.

Shirky, C. (2003, April 24). A group is its own worst enemy. In *Proceedings of the O'Reilly Emerging Technology Conference in Santa Clara.*

Van Couvering, E. (2004). New media? The political economy of Internet search engines. In *Paper presented to the Conference of the International Association of Media & Communications Researchers.* Porto Alegre: IAMCR.

Weinberger, D. (2002). *Small pieces loosely joined. A unified theory of the Web.* New York: Perseus Book.

Wiener, N. (1950). *Cybernetics: Or the control and communication in the animal and the machine.* Cambridge: MIT Press.

Wiener, N. (1950). *The human use of human beings.* Boston: Houghton Mifflin.

KEY TERMS

Cognitive Capital: A concept that represents knowledge as a scarce resource that can be traded with money, social influence, and political power. This concept is derived from Pierre Bourdieu's concept of "cultural capital," and it sheds light on accumulation and exchange processes regarding cognitive skills, knowledge, and information. Cognitive capital is now recognized as a key asset of institutions and economic organizations.

Cybernetics: "Control and communication in the animal and machine," as defined by its founder Norbert Wiener. This discipline studies living beings, machines, and organizations as regulatory feedback systems. The choice of term, "cybernetics," from Greek Κυβερνήτης (kybernetes, steersman, governor), shows Wiener's awareness, extensively argued in diverse works of his, about political and social relevance of interactive communication networks.

Folksnomy: System used in information organization. Rather than providing an ex ante categorizing project as in taxonomy, it exploits an open labeling system generating pertinences through its user cooperation. Assigning a label (tag) to every piece of information (photo, video, text, address, etc.) users contribute by donating a sense to the digital product universe, otherwise

unable to be tracked and used by its own community of production.

Graphical User Interface (GUI): Interface facilitating human-machine interaction, with graphic elements. Rather than inputting data and instruction in text format, GUI's user controls its elaborator directly manipulating objects: graphic images, menus, buttons, boxes, and so forth. The Interface translates user actions into machine commands, thus representing an intermediate normative zone between man and machine, a zone governed by code produced by software houses.

Instant Messaging (IM): Real-time Internet-based system allowing communication between two or more subjects. It represents an evolution of Internet Relay Chat, to whose decline it contributed. It provides a client software that allows synchronous conversations by means of interfaces supplied with many multimedia functionalities (audio, Webcam, file transfer, *animoticons*, etc.). Unlike social software, conversations produced by IM are not immediately reintroduced in cooperative processes of significance production.

Network Neutrality: Technical and political characteristic of those networks not allowing resource discrimination on the basis of their destination, content, and the applicative class of technology to which they belong. Debate on network neutrality initiated in the United States following an increasingly incisive reprojection of networks by Internet service providers (ISP) and telecommunications providers. Opposing this project reformulating Internet architectures, first of all, are content providers who would be obliged to pay, together with users, on the base of gained visits, used bandwidth, and typologies of service delivered.

Software: An unambiguous sequence of instructions that allows a machine to elaborate information. Software, also called program, defines rules and routines by which computer hardware has to act in order to perform its tasks. Combining software with physical resources, a computer can operate as a means of production, creating digital goods by manipulating a user's input.

This work was previously published in Encyclopedia of Multimedia Technology and Networking, Second Edition, edited by M. Pagani, pp. 224-229, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.12
E-Democracy:
The Social Software Perspective

Pascal Francq
Université Libre de Bruxelles, Belgium

ABSTRACT

The success of the Internet has launched McLuhan's idea of the global village. Over the years, the Internet has become a real political medium which has inspired the emergence of the concept of e-democracy. Despite some successful applications, many limitations prevent its wide expansion. Some of these limitations can be solved with social software, in particular with the emerging Web 2.0 applications. This kind of applications may contribute to a better application of e-democracy processes for local political decisions.

INSIDE CHAPTER

The Internet is today a widely used platform to exchange information and share knowledge. In this chapter, we propose a prospective study of the use of the Internet as support for e-democracy processes. The history of the Internet shows that social software has always been developed to support knowledge sharing among net surfers. Since participating in political issues implies knowledge sharing, the Internet was rapidly used as a political medium. The concept of e-democracy, i.e. the use of information and communication technologies to allow citizens to participate in the democratic process, is a natural evolution of this situation. Several examples demonstrate that e-democracy can be deployed for local decision purposes. The experiences have also shown several limitations, in particular concerning the on-line tools currently offered. We argue that solutions exist to overcome these limitations and that their integration in social software environments may enhance the concept of e-democracy in order to apply it to more complex decision-taking situations.

INTRODUCTION

With the expansion of available information and the diversity of communication and information technologies, the Internet is a medium support that cannot be ignored. Limited to scientists during its early days, the Internet became a platform for knowledge sharing and collaboration for a variety of domains due to the multitude of social software developed. In particular, the political domain was always present on the Internet and political movements have often used this medium to support their actions. Today, with one billion net surfers around the world, many people claim that the idea of "global village" popularized by

Marshall McLuhan in the late 1690s (McLuhan & Fiore, 1967) is a reality. Now that it becomes possible to engage in discussions with everybody around the world across geographical and temporal boundaries, many people believe in the Internet as a virtual place where different cultures may peacefully coexist. From this dream there emerged, in the late 1990s, the concept of **e-democracy**. The main idea is to use the information and communication technologies to make citizens participate more directly in the democratic process. Several examples of the application of this concept for local decision-taking (decisions concerning a part of a city or a town) seem to demonstrate that the idea of e-democracy is no longer utopian. Moreover, with the emergence of the "Web2.0" concept, there is an increased use of social software by net surfers. This chapter proposes a prospective view of the application of social software to support e-democracy processes. In fact, we argue that social software can already be useful in this context. Moreover, we believe that it may be enhanced by integrating existing technologies to overcome certain limitations related to the current tools supporting e-democracy processes. Concretely, the second section proposes a short historical overview to show how the Internet, initially developed to support knowledge networks, was rapidly used as a political medium. The third section presents the concept of e-democracy, some examples and the main limitations of the current applications. The fourth section proposes an overview of how modern social software can be used in the context of e-democracy, and which technologies should be integrated to propose on-line tools enhancing the application of e-democracy processes. Since these evolutions solve some of the e-democracy limitations, the fifth section briefly analyses in which context the concept of e-democracy should be deployed. Finally, the sixth section proposes some conclusions.

FROM KNOWLEDGE NETWORKS TO POLITICAL MEDIA

The history of the Internet studied in section 2.1 shows that new collaboration tools and methodologies were continuously developed to increase knowledge sharing among social networks (scientific ideas, software, electronic resources, ...). Based on these emerging social networks, communities of net surfers have collaborated through the Internet on many different projects (section 2.2). As soon as the Internet allowed net surfers to freely exchange ideas, it was rapidly used to discuss political issues. The result is the evolution of the Internet to a real political medium (section 2.3).

Internet as Knowledge Networks

The Arpanet computer network, the ancestor of today's Internet, was designed to help researchers to collaborate. Tools such as e-mail and newsgroups were developed in the late 1960s to allow researchers to discuss their ideas, present their results and take scientific decisions. Initially limited to the military research area, the access to Arpanet was rapidly extended the whole research community. In the 1980s, Arpanet became the collaboration platform for many research teams around the world. The result of this evolution was the creation in the late 1980s of the Internet as known today. The main problem with the communication tools existing that time was their asynchronous dimension which strongly limits the interactions between people. With the increased use of the Internet during the 1990s, new synchronized communication applications were developed: instant messaging tools and chat rooms. These tools provided a new level of interactions in the communication process between net surfers. As studied by Rheingold (2000), this technology has permitted the emergence of **virtual communities** of net surfers sharing similar interests. Once

communities have emerged, it is possible to make them live through all modern communication supports. With the evolution of communication tools, in particular the net meeting applications, it has become possible to efficiently organize discussions involving participants that are not presented at the same place[1]. Initially developed at the beginning of 1990s as simple diffusion tools for scientific results at CERN, Web sites have become over the years real knowledge dissemination tools. Today, the number of Web sites containing interesting information has exploded making the localization of the relevant information a crucial issue. Since several technologies have increased the interactivity between Web sites and net surfers, the Web can be considered as a social environment and many modern Web sites have become real social software.

The Internet as Collaboration Platform

With the increasing number of net surfers and the availability of synchronous communication tools, the Internet evolves to a collaboration platform. Built upon virtual communities, net surfers have started to collaborate, in particular to build freely available artefacts (software, books, music tunes, ...). Emerging in the early 1980s, the free and open source software movements are the best examples of free collaboration through the Internet. By combining social software with specific collaboration tools, such as source code managing tools, complex software projects[2], comparable in quality and features to well established commercial software, were successfully developed (Raymond, 2001). One surprising aspect of these projects is their ability to manage long-term decisions and to self-organize the work division across a dozen of individual developers. Most of the time, when conflicts appear concerning important decisions in a given project, such as the choice of a project leader, a democratic approach is privileged by the community. This new production process has been successfully applied to other domains than software development. Founded in 2001, Wikipedia[3], the free content encyclopaedia project, is the best known example of the application of the concept of open source to other content than source code. Despite some problems concerning the control of the quality of part of the content[4], everybody agrees that Wikipedia demonstrates that high quality knowledge can be built in a collaborative way. In particular, on-line writers must sometimes reach a comprise as they elaborate articles discussing controversial issues. Since the early 2000s, and the emergence of the "Web2.0" applications[5], a new dimension in on-line collaboration is proposed. In fact, the role of the net surfer has dramatically changed: he or she has been transformed from a "passive consumer of information" to an active organizer of the Internet content. The new paradigm behind this social software is that net surfers should collaborate to manage shared resources (such as bookmarks, contacts, videos, ...) they find relevant. Since current search tools have difficulties treating the amount of available information on the Internet, this approach proposes a sort of "human indexing" of its content and its users.

The Internet as Political Medium

Since it is possible to communicate and collaborate through the Internet, it seems natural that it is a place where political issues may be discussed. A close look at the history of the Internet shows that its use as a political medium is as old as the Internet itself. In fact, as pointed out by Rosenzweig (1998), it is hard to believe that the creators of the Arpanet were unaware of the political context on the late sixties in the USA, in particular the protests against the Vietnam War mobilizing the American universities campuses. Moreover, some authors claim that the success of the Internet can only be understood if we consider that the first net surfers have adopt it as a democratic and interactive communication platform (Hauben &

Hauben, 1997). This latest theory finds an echo with an analysis of the content of the messages exchanged in newsgroups carried out in 1998. This study showed that 12% of the total number of messages on Usenet were dedicated to political subjects (Hill & Hughes, 1998). The fact that political issues were discussed in newsgroups is not surprising since it was certainly the first virtual place where any kind of subjects were discussed. But, the amount dedicated to politics reveals of the political awareness of many net surfers at that time and the existence of politically-oriented virtual communities. With the growing popularity of the Web as mass communication and diffusion medium, many "politically-oriented" organizations have understood the potential of the Internet as an internal and external communication tool. Internally, it provides a collaboration platform and is a source of information for members and sympathizers. Externally, the goal is to spread their ideas on the Net and to sensitize net surfers to their points of view. With the multiplication of blogs, it becomes very easy for everybody to publish their political ideas and to create a new genre of journalism (Wall, 2005). Conscious that the Internet is becoming widely used among the population in some countries, in the late 1990s, different public authorities deployed **e-government** projects, i.e. using information and communication technologies to better communicate with their citizens. Today, in some countries, citizens can use the Internet to access the debates of parliament, to ask for administrative documents or to fill their tax returns. In the e-government approach, the Internet is only used as a communication medium for governments, but there are no real interactions between citizens and public authorities.

E-DEMOCRACY: APPLICATIONS AND LIMITS

If the Internet was used very early to discuss political issues, it took some time before these discussions were transformed into real political actions. The concept of e-democracy emerged from Internet in the late 1990s (section 3.1) and was successfully applied to several situations (section 3.2). But, as analysed in section 3.3, there are actually some important limitations.

Concept of E-Democracy

There exist several definitions in the literature for the concept of e-democracy (sometimes called cyber-democracy). Nevertheless, all authors agree that it is related to the use of information and communication technologies to involve citizens, sometimes called **e-citizens**, in the political decision-making processes. If the idea of using technologies to build a better and more humane society is not new (Wiener, 1965), the actual development of the Internet in our modern societies makes its realization partially possible. Besides the multiple theoretical definitions, it is also possible to define e-democracy as a wide range of actions that should be available for citizens through the Internet: voting for people, participating in online referendums, intervening in public debates, etc. Since we are discussing e-democracy applications rather than the concept itself, we will adopt this approach. In particular, we choose the definition proposed by Vedel (2003) suggesting that the concept of e-democracy includes three levels of interactions between e-citizens and public authorities:

1. Access to information to ensure transparency in political decisions.
2. Build debates and discussion places between citizens in order to coordinate political actions.
3. Participate in public deliberations and decision-making.

Today, the concept of e-democracy is quite successful in Northern countries. Two elements may explain this:

1. Many citizens in these countries are petitioning for a more participative democracy.
2. In the collective conscience of these countries, the Internet is widely accessible for most citizens, even through if the reality is somewhat different.

Some Examples

The Public Electronic Network (PEN), which started in Santa Monica in 1989 (Rogers, Collins-Jarvis & Schmitz, 1994) is often cited as the first example of e-democracy application. By providing a network of terminals, the American city gave the opportunity to its citizens to discuss problems together. One of the results of these discussions was a project to help the homeless people of the city. Since this experience, several other cities or regions have deployed some of the principles of e-democracy (OECD, 2003). Most of these experiences can be categorized into two types of democratic interactions:

1. The creation of Internet portals where citizens can not only consult information (already available with e-government applications), but also interact directly with politicians, for example by sending e-mails or commenting on proposals.
2. The use of electronic consultations to get the opinion of citizens. Nevertheless, in most cases, there is not always an obligation for public authorities to take the results of these consultations into account.

If most of e-democracy applications were organized by public authorities, new situations emerge today where the e-democratic initiatives are directly launched by citizens themselves. The fact that Internet is more and more used to promote e-petitions (electronic petitions) is certainly the visible part of this for most net surfers. Successful uses of e-petitions in the context of local politics were already pointed out (Macintosh, Malina & Farrell, 2002). Moreover, some e-petitions have influenced more global political decisions, such as during the debate on software patents at the level of the European Parliament where it was one of the methods used by the opponents to sensitize the parliamentarians. Despite the criticism concerning the electronic vote[6], several initiatives have been successfully organized. For example, during the primary votes of the Democrats in Arizona in 2002, nearly 50% of the votes were cast using the Internet (Solop, 2002). The important point of such initiatives is that Internet should never be the only way to interact, but one of the possible ways. In fact, since people can participate from "their homes", the Internet may support the participation of people who would not participate without it. Some initiatives were more ambitious and aim to influence global political stakes. During the 2004 US presidential campaign, a Web site allowed net surfers all over the world "to vote" for their candidate[7]. 500.000 net surfers participated in this virtual election with, of course, no influence at all on the final results. But, this can be seen as an interesting experience of e-democracy at a global level.

Main Limitations

Despite some successful e-democracy applications, there are several important limitations that impede a generalized deployment. The main limitation is, of course, the reality of the digital divide. Many people have not (or rarely) access to the Internet. This makes e-democracy, in practice, a preserve for "rich" populations. If this difference is evident between the South and the North (geographical digital divide), among the Northern countries, many people cannot use, at least regularly and correctly, the Internet (social digital divide). Besides, several authors claim that e-democracy will only be available for a small part of the population which can then control the political choices (Barber, 2004). Apart from these

problems "outside of the technological sphere", the current examples of e-democracy are always related to very specific issues due to the lack of adequate integrated solutions. Of course, everybody would agree that deploying technologies will never solve all the real problems. Our point is that some of these problems are directly related to the underlying tools. To illustrate this, we propose to study the main limitations associated to the different levels of interactions proposed by Vedel (2003). The first category of interactions is related to the access of the relevant information for citizens. If the e-government initiatives can be seen as a first attempt to offer an access to public information, most of the time, e-government portals are not well structured. The consequence is a difficulty in finding all the relevant information related to a given problem. Moreover, all necessary information to build a real political culture is not available on e-government portals. It is also useful to access studies in universities, documents of political parties, citizens' testimonies, etc. Accessing other sources of information, such as the Internet, is an important issue. But the Internet is characterized by an increased quantity of available information, and current search solutions, in particular search engines, cannot face this complexity (Fogarty, & Bahls, 2002). Beside this problem of quantity of information, the quality of the proposed information is also problematic, in particular for search engines which are the most popular search tools. Most Internet search engines rank highly the documents which are the most pointed by hyperlinks on the Web (Brin & Page, 1998). Since hyperlinks can be interpreted as human assessments on a given document, i.e. assessments on the ideas defended by a "politically-oriented" document, search engines have a tendency to always propose the documents containing ideas shared by the majority of net surfers. This problem explains the frequent criticism that the Internet acts as support for a certain form of single thought. We believe that the current search methods should be enhanced, in particular to ensure the diversity of the points of view presented to net surfers. The second category of interactions is related to the creation of places where citizens can debate over political issues. As explained earlier, newsgroups and chat rooms are powerful tools for discussions. Participating in on-line discussions can make communities emerged (Rheingold, 2000), in particular net surfers sharing similar ideas and collaborating with a unique political goal. In fact, newsgroups were already successfully used for public on-line consultations (Rosen, 2001). But, the multiple existing communication channels where such communities may exist are a brake for the emergence of large communities of organized citizens, which is the core of the e-democracy concept. Of course, if there is a fixed political reference for a group of people, such as the Web site of a party or any politically-oriented organization, a community exists *de facto*. On the other hand, limiting the communities built around existing entities is somewhat reductive. We believe that networking tools should help people build new communities and that political portals should be used to coordinate political actions around specific subjects. The third category is related to participating in public debates and decision-making. For simple decision-making, such as answer a simple question (yes/no, for/against) or choose a candidate, Internet technologies can be used without any major technical problems. But, for more complex decision-taking, such as drawing up a budget for a town or writing a bill, there is a real lack of well-established on-line tools. We believe that existing methods managing complex decision-taking should be integrated in political portals.

BUILDING AND ORGANIZING POLITICAL NETWORKS

In the previous section, the short overview of the concept of e-democracy has shown the main actual limits of its application and some possible

directions for enhancement to solve some of these limitations. These enhancements should support the building and organization of political networks not only to group net surfers sharing the same ideas, but also to provide a discussion environment where people having different points of view on a same topic can hold a debate. Today, there is a need for independent **political portals**. Such portals should be sources for information related to specific political subjects (environment, democratic participation, education, ...). They should reference information coming from e-government applications (since many official political documents are public) but also from alternative sources. They should be build upon content management systems to provide an environment for debates and collaborative writing of documents. We argue that a combination of these political portals with other social software, in particular in the context of the emerging Web2.0 applications, may contribute to solve three major problems:

1. Ensure diversity of points of view.
2. Create political networks.
3. Organize complex decision-taking.

Diversity of Points of View

One of the main stakes behind the concept of e-democracy is to access to information representing different points of view on a given problem. As already explained, the e-government portals currently developed by several countries should help to make official information more accessible. But, it is also necessary for e-citizens to read information coming from other sources and comment this information with other people, which is the role of the political portals. Since they must reference relevant information concerning a particular political subject, they should be combined with applications helping to access to interesting information available on the Internet. Several emerging Web2.0 applications propose some solutions to the access of information by providing a platform for a **social indexing** of the Web. Social bookmarking applications, such as del.icio.us, are well known examples. When net surfers find interesting documents on the Web, they tag them with keywords. It is then possible to find all the documents tagged, thus humanly assessed as relevant, with a given set of keywords. We already know that applying the social bookmarking principle can help net surfers (citizens) to access relevant (political) content. But this principle can also help citizens to have a more open mind. In fact, since political issues are characterized by a diversity of points of view, it is evident that a given document, a law or a study for example, may be interpreted in several ways depending on its readers. This diversity of points of view on a given document will probably correspond to different keywords used to tag this document. So, since net surfers will see all the keywords used by others to tag the same documents, it will be possible for them to have an overview of the different points of view by analysing the different sets of keywords used. Once these documents are tagged, they could be discussed on political portals. Everyone would agree that the major problem concerning Internet is related to the quality of the on-line content. For example, many Web sites around the world diffuse racist ideas. Currently, there is no technology that proposes a real solution to this problem, but several approaches have been proposed to limit its impact. One of them, used by several portals such as Internet marketplaces, is to propose a rating system of users (Chen & Singh, 2001). This system gives the opportunities to users to express a degree of trust to others based on their experiences of the interactions with them (for example buying something from them). This approach could be used in another way in our context: users would rate the people whom they do not trust. The idea of rating the non-trusted people rather than rating the trusted people is to avoid the limitation of the diversity of points of view. In fact, if users rated people

by trust, it is probable that they would rate well people sharing the same points of view, and the consequence would be to connect with people that reinforce their opinions rather than open them to diversity. By rating non-trusted people, it will be possible to identify groups of people who are never considered as trusted by most users. This information will be used to filter the information proposed or, at least, to inform net surfers that some political contributions were written by very untrustworthy people and should be taken with caution.

Building Political Networks

The e-democracy concept claims that e-citizens should be able to act on political decisions. It seems therefore evident that they must coordinate their actions. Such a coordination can be built upon "politically-oriented" virtual communities through political portals. If several tools exist to make these communities live (for example newsgroups) and could be integrated in portals, building these **political networks** remains a challenge. In the previous section, we have explained how social bookmarking applications may be used in the context of e-democracy. Moreover, some researchers have shown that it is possible to cluster the net surfers based on the keywords they used to tag documents (Paolillo, & Penumarthy, 2007). But, as already explained, the tags used represent a certain points of view of the tagged document. There is therefore a risk that doing a clustering on a tag-basis will lead to regroup people sharing the same point of view on a particular topic, which will reduce the access to a diversity of points of view. In fact, we need to build communities of people sharing the same interests on a given topic and not sharing the same points of view on a given topic. The clustering of net surfers into communities must therefore be based on the content of the information. The GALILEI platform is one of the solutions that proposes a solution for this problem (Francq, 2007). This platform implements an approach based on **social browsing**. In this approach, the net surfers define different interests, called profiles, and assess the documents they consider as relevant for their profiles. The system computes descriptions for the different profiles of the net surfers based on the relevance assessments on documents and a content analysis. The profiles are then clustered on the basis of their descriptions: similar profiles are grouped together in order to define a number of communities of interests. If this approach is applied with a corpus of political documents, the communities will group people sharing common interests on different political subjects.

Organize Complex Decision-Making

One of the main ideas behind the concept of e-democracy is to make citizens participate in political decisions-making. In the different examples existing today, this participation is limited to making a "simple choice", such as voting for a candidate. For more complex decision-making, there is a lack of on-line tools. A good example of complex decision-making process is to decide how to organize the allocations of a given budget. To solve this problem it is not only necessary to evaluate the priorities of each citizen, but also several constraints influence the choice such as the total budget available. In fact, within a given budget, a situation may occur where two choices can be financed: either project with priority 1 or projects with priority 2 and 3. Choosing between these two possibilities is not as easy as choosing a unique candidate for an election. Another complex problem is for several people to reach a compromise. The collaborative writing of a local policy is a typical example of this kind of problem. A first approach for dealing with complex decision-making is to gather more information from the participants and to integrate this information in the final decision. The domain of operations research has provided a huge number of methods that help take decisions (Winston, 2003), in

particular computer-aided methods for multiple criteria decision-making. But, in the context of e-democracy, these computer-aided decision systems have nevertheless several drawbacks. Since most citizens do not have the competences to understand the methods implemented in these systems, they will probably not correctly understand how their information will be used to take the final decision. This means that these systems can favour the small number of citizens that know which information they have to give to defend their points of view, which is one of the criticisms often made against e-democracy. Moreover, these systems cannot solve every form of complex decision-making, such as agreeing to a comprise. A second approach is to integrate in political portals methods developed to help a group of people dealing in complex decision-making. One of the known method is Delphi (Linstone & Turoff, 1975). This method structures the group communication process of a group of individuals in order to make them solve a complex problem as a single entity. This method was successfully applied in several contexts, including:

- building a common interpretation of historical events.
- evaluating possible budget allocations.
- delineating the pros and cons associated with potential policy options.

An on-line Web system implementing the Delphi approach was already developed (Kenis, 1995). It organizes the communication between a set of users using an interactive process:

1. Each user gives its opinion to a question asked.
2. A moderator proposes a compromise based on the different answers.
3. The compromise is submitted to every user. They can accept it or reject it (and give their comments).
4. If a given majority does not accept the proposition, the process is reiterated beginning from the second step.

Many problems remain for this type of approach, in particular the role of the moderator of the process, the authentication of the net surfers participating in the debate or the democratic control of the process. But, we believe that integrating such a system can propose a new democratic approach for solving some complex problems.

E-DEMOCRACY: WHICH CONTEXT?

It is difficult to evaluate the status of e-democracy. The paradigms of the global village (McLuhan & Fiore, 1967) and the capacity of computers to help human beings (Wiener, 1965) have created many hopes in the Internet to solve the problem of the confidence crisis of most modern democracies. Since the information and communication technologies have demonstrated their capacities to make people exchange and collaborate, many people believe that the e-democracy concept must be promoted. On the one hand, several different initiatives of this concept have been successfully applied, in particular to local political decisions. Moreover, the previous sections have shown that a combination of specific social software and political portals may solve some of the limitations of the application of e-democracy. On the other hand, many limitations cannot currently be solved. Firstly, technologies cannot best organize all forms of decision-making. Secondly, it is necessary to build new control mechanisms to ensure that the way technologies are used respects the democratic process. But, the main limitation is without doubt the (geographical and social) digital divide. To think that, from now to the middle of the twenty-first century, applications of e-democracy will be widely used seems nowadays utopian. Knowing that the geographical digital divide will not be

solved rapidly, it is yet possible to limit locally the social digital divide. Therefore, two main categories of applications of e-democracy will probably be developed in the future:

1. For political local choices related to problems concerning small socially homogeneous communities of citizens, typically the management of a town. If e-citizens have a similar level of access and mastering of the Internet, on-line collaboration tools can increase the commitment of the citizens to final political decisions.
2. For decision-making in organizations where most members have a regular access to the Internet, such as the free and open source community. For nongovernmental organizations, it may also help since most of their decisions centres are located in Northern countries.

We can therefore fear to see in the next decades an asymmetrical deployment of e-democracy applications.

CONCLUSION

The Internet was originally built as a knowledge sharing platform for scientists. Its decentralized schema and the increased facility of content creation and information access have contributed to its use as political medium. The concept of e-democracy was born with the idea of using information and communication technologies to help citizens to interact more directly in political decision-making. In fact, if some examples of e-democracy processes have successively been applied in local areas, despite some utopian attempts, the actual on-line solutions have limitations. Today, some of these limitations can be solved by using and enhancing political portals and social software. Firstly, the emerging Web2.0 applications propose collaboration solutions to manage the information on the Internet. If applied to political content, it may help people to find information on their political interests as well as ensure the diversity of points of view. Moreover, community creation approaches can be used to link people sharing similar political interests. Finally, by integrating on-line methods for complex decision-making on political portals, it is possible to organize on-line collaboration for a group of people in order to take complex decisions. The e-democracy concept has emerged because of the increased use of the Internet. With the digital divide existing today, it is impossible to apply this approach to a wide range of applications. If all democratic constraints must be respected, only specific local or social contexts exist where a real e-democracy process can be applied. Despite the unsolved limitations, this approach proposes a real participative democratic process and should be developed wherever it is possible.

REFERENCES

Barber, B.R. (2004). *Strong Democracy: Participatory Politics for a New Age*. University of California Press.

Brin, S., & Page, L. (1998). The antaomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*(33), 107-135.

Chen, M., & Singh, J.P. (2001). *Computing and using reputations for internet ratings.* Tampa, Florida, USA : ACM Press.

OECD (2003). *Promise and Problems of E-democracy: Challenges of Online Citizen Engagement.*

Fogarty, M., & Bahls, C. (2002). Information Overload : Feel the pressure? *The Scientist, 16*(16).

Francq, P. (2007). *The GALILEI Platform: Social Browsing to Build Communities of Interests and Share Relevant Information and Expertise*. In

M.D. Lytras & A. Naeve (Editors), *Open source for knowledge and learning management : strategies beyond tools.* Idea Group Publishing (319-342).

Hauben, M., & Hauben, R. (1997). *Netizens: On the History and Impact of Usenet and the Internet.* IEEE Computer Society Press.

Hill, K.A., & Hughes, J.E. (1998). *Cyberpolitics: Citizen Activism in the Age of the Internet.* Rowman & Littlefield Publishers, Inc.

Kenis, D.G.A. (1995). *Improving group decisions: designing and testing techniques for group decision support systems applying Delphi principles.* Universiteit Utrecht.

Linstone, H.A., & Turoff, M. (1975). *The Delphi Method: Techniques and Applications.* Addison-Wesley Pub. Co., Advanced Book Program.

Macintosh, A., Malina, A., & Farrell, S. (2002). Digital Democracy through Electronic Petitioning. *Digital Government. Dordrecht: Kluwer.*

McLuhan, M., & Fiore, Q. (1967). *The Medium is the Massage : An Inventory of Effects* (G. Press, Éd.). Jerome Agel.

Paolillo, J.C., & Penumarthy, S. (2007). The Social Structure of Tagging Internet Video on del. icio. us. *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 85-85.

Raymond, E.S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly & Associates.

Rheingold, H. (2000). *The Virtual Community.* MIT Press.

Rosen, T. (2001). E-Democracy in Practice: Swedish Experiences of a New Political Tool. *Stockholm, Swedish Association of Local Authorities and Swedish Federation of County Councils and Regions, Department of Democracy and Self-Government.*

Rogers, E.M., Collins-Jarvis, L., & Schmitz, J. (1994). The PEN project in Santa Monica: Interactive communication, equality, and political action. *Journal of the American Society for Information Science*, *45*(6), 401-410.

Roy Rosenzweig. (1998). Wizards, Bureaucrats, Warriors, and Hackers: Writing the History of the Internet. *The American Historical Review*, *103*(5), 1530-1552.

Solop, F.I. (2002). Digital Democracy Comes of Age: Internet Voting and the 2000 Arizona Democratic Primary Election. *PS: Political Science and Politics*, *34*(02), 289-293.

Vedel, T. (2003). L'idée de démocratie électronique: Origines, Visions, Questions. *Le désenchantement démocratique, La Tour d'Aigues: Editions de l'Aube*, 243-266.

Wall, M. (2005). 'Blogs of war': Weblogs as news. *Journalism*, *6*(2), 153.

Wiener, N. (1965). *Cybernetics:: Or Control and Communication in the Animal and the Machine.* Mit Pr.

Winston, W.L. (2003). *Operations Research.* Duxbury P.,U.S.

CASE STUDY

BetaVote.com—What if the Whole World could Vote in the U.S. Presidential Election?

During the 2004 US presidential elections, two Americans, Daniel Young and Kevin Frost, claimed that since the decisions of the United States influence the whole world, every citizen in the world should participate to the vote of its

president. They have therefore created a Web site where net surfers could choose between John Kerry and George W. Bush. Around 500,000 net surfers participated in this virtual election, and 88% of them chose senator Kerry as US president. Also, although these Internet results did not influence the real results, this initiative can yet be seen as an experience in e-democracy at a global level. Nevertheless, it illustrates the problem of the digital divide. The percentage of voters is relatively low in comparison to the total number of net surfers (evaluated to one billion). Beyond the digital divide, the low number of participants also illustrates the lack of relay for such initiatives on the Internet. Secondly, in the US, the results of the on-line vote (around 70,000 net surfers participated) gave the victory to John Kerry while the real vote gave the victory to George W. Bush. Since some studies have shown that the Americans who voted for John Kerry were mostly "highly educated", the results of the e-democracy approach seem to confirm that using the Internet as a political medium may be the preserve of a given "elite", which is one of the criticisms against e-democracy. It will be interesting to reiterate this experience during the next US presidential campaign and make comparisons with the one of 2004. In particular, the total number of net surfers participating should be analyzed as well as their geographical distribution.

USEFUL URLS

1. Communauté de communes de Parthenay: A French example of e-democracy portal, http://portail.cc-parthenay.fr/Portail2007
2. Council of Europe Forum for the Future of Democracy, http://www.coe.int/t/e/integrated_projects/democracy/
3. E-Democracy.Org/Minnesota E-Democracy, http://www.e-democracy.org
4. Villes Internet, Villes Internet – agir pour un internet citoyen, http://www.villesinternet.net
5. What if the whole world could vote in the U.S. presidential election?, http://www.betavote.com

FURTHER READINGS

Everard, J. (2001). Virtual states: the Internet and the boundaries of the nation state. Routledge.

Habermas, J. (1991). The Structural Transformation of the Public Sphere: An Inquiry Into a Category of Bourgeois Society. The MIT Press.

Rheingold, H. (2002). Smart Mobs: The Next Social Revolution. Perseus Books Group.

ENDNOTES

1 The digital divide is, of course, a problem since every participant should have an access to the Internet, a sufficient bandwidth and the corresponding hardware (microphone, webcam, etc.).

2 Linux, Mozilla, OpenOffice.org, Apache or K Desktop Environment are well known examples of software deployed on millions of computers today.

3 Wikipedia project: http://www.wikipedia.org

4 The main criticism concerning Wikipedia is the difference of quality between the articles. Some of them may be written by a team of experts of the corresponding domain, while others can be written by people without a real expertise. Many net surfers do not verify which authors have written which articles, and suppose that all articles have the same level of quality.

5 Many "Web2.0" applications appear on the Internet such as del.ico.us, LinkedIn, etc.

6 The democratic control of how technologies are deployed in the context of the electronic vote is an important issue. We believe that this is an "organisational" problem, since it is possible to control how the information is gathered and how the software manages this information. In democracies, independent commissions should organize this control and ensure the necessary transparency.

This work was previously published in Knowledge Networks: The Social Software Perspective, edited by M. Lytras; R. Tennyson; and P. Ordonez de Pablos, pp. 61-73, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.13
Software Engineering and HCI

Shawren Singh
University of South Africa, South Africa

Alan Dix
Lancaster University, UK

INTRODUCTION

Technology Affecting CBISs

As computer technology continues to leapfrog forward, CBISs are changing rapidly. These changes are having an enormous impact on the capabilities of organizational systems (Turban, Rainer, & Potter, 2001). The major ICT developments affecting CBISs can be categorized in three groupings: hardware-related, software-related, and hybrid cooperative environments.

Hardware-Related

Hardware consists of everything in the "physical layer" of the CBISs. For example, hardware can include servers, workstations, networks, telecommunication equipment, fiber-optic cables, hand-held computers, scanners, digital capture devices, and other technology-based infrastructure (Shelly, Cashman, & Rosenblatt, 2003). Hardware-related developments relate to the ongoing advances in the hardware aspects of CBISs.

Software-Related

Software refers to the programs that control the hardware and produce the desired information or results (Shelly et al., 2003). Software-related developments in CBIS are related to the ongoing advances in the software aspects of computing technology.

Hybrid Cooperative Environments

Hybrid cooperative environments developments are related to the ongoing advance in the hardware and software aspects of computing technology. These technologies create new opportunities on the Web (e.g., multimedia and virtual reality) while others fulfill specific needs on the Web (e.g., electronic commerce (EC) and integrated home computing).

These ICT developments are important components to be considered in the development of CBIS's. As new types of technology are developed, new standards are set for future development. The advent of hand-held computer devices is one such example.

BACKGROUND

A Software Engineering View

In an effort to increase the success rate of information systems implementation, the field of software engineering (SE) has developed many techniques. Despite many software success stories, a considerable amount of software is still being delivered late, over budget, and with residual faults (Schach, 2002).

The field of SE is concerned with the development of software systems using sound engineering principles for both technical and non-technical aspects. Over and above the use of specification, and design and implementation techniques, human factors and software management should also be addressed. Well-engineered software provides the service required by its users. Such software should be produced in a cost-effective way and should be appropriately functional, maintainable, reliable, efficient, and provide a relevant user interface (Pressman, 2000a; Shneiderman, 1992; Whitten, Bentley, & Dittman, 2001).

There are two major development methodologies that are used to develop IS applications: the traditional systems development methodology and the object-oriented (OO) development approach.

The traditional systems approaches have the following phases:

- Planning: this involves identifying business value, analysing feasibility, developing a work plan, staffing the project, and controlling and directing the project.
- Analysis: this involves information gathering (requirements gathering), process modeling and data modeling.
- Design: this step is comprised of physical design, architecture design, interface design, database and file design, and program design.
- Implementation: this step requires both construction and installation.

There are various OO methodologies. Although diverse in approach, most OO development methodologies follow a defined system development life cycle. The various phases are intrinsically equivalent for all of the approaches, typically proceeding as follows:

•OO Analysis Phase (determining what the product is going to do) and extracting the objects (requirements gathering), OO design phase, OO programming phase (implemented in appropriate OO programming language), integration phase, maintenance phase and retirement (Schach, 2002).

One phase of the SE life cycle that is common to both the traditional development approach and the OO approach is requirements gathering. Requirements' gathering is the process of eliciting the overall requirements of the product from the customer (user). These requirements encompass information and control need, product function and behavior, overall product performance, design and interface constraints, and other special needs. The requirements-gathering phase has the following process: requirements elicitation; requirements analysis and negotiation; requirements specification; system modeling; requirements validation; and requirements management (Pressman, 2000a).

Despite the concerted efforts to develop a successful process for developing software, Schach (2002) identifies the following pitfalls:

- Traditional engineering techniques cannot be successfully applied to software development, causing the software depression (software crisis). Mullet (1999) summarizes the software crisis by noting that software development is seen as a craft rather than an engineering discipline. The approach

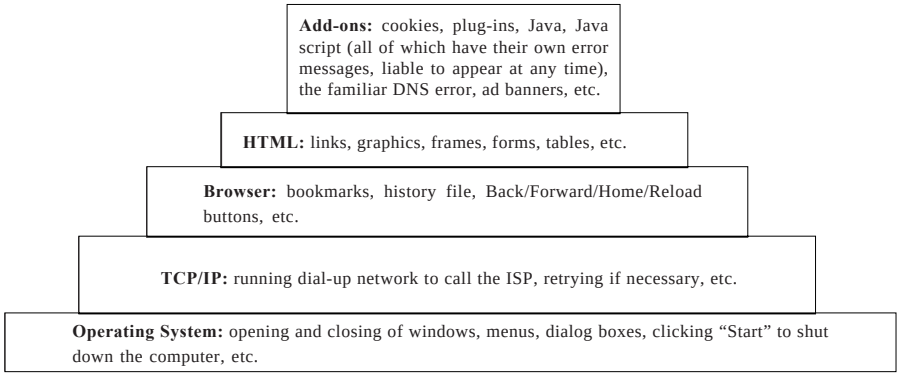
to education taken by most higher education institutions encourages that "craft" mentality; lack of professionalism within the SE world (e.g., the failure of treating an operating system's crash as seriously as a civil engineer would treat the collapse of a bridge); the high acceptance of fault tolerance by software engineers (e.g., if the operating system crashes; reboot hopefully with minimal damage); the mismatch between hardware and software developments. Hardware and software developments are both taking place at a rapid pace but independently of each other. Both hardware and software developments have a maturation time to be compatible with each other, but by that time everything has changed. The final problem for software engineers is the constant shifting of the goalposts. Customers initially think they want one thing but frequently change their requirements.

Notwithstanding these pitfalls, Pressman (2000b) argues that SE principles always work. It is never inappropriate to stress the principles of solid problem solving, good design, thorough testing, control of change, and emphasis on quality.

The Web is an intricate and complex combination of technologies (both hardware and software) that are at different levels of maturity. Engineering Web-based EC software, therefore, has its own unique challenges. In essence, the network becomes a massive computer that provides an almost unlimited software resource that can be accessed by anyone with a modem (Pressman, 2000a). We illustrate these intricacies in Figure 1, which is a representation of a home computer that is attached to the Internet. It depicts the underlying operating system (the base platform), the method of connection to the Internet (dial up, the technology that supports Web activities), browser, an example of a Web communication language (HTML), and additional technology that may be required to be Web active.

All the aspects of Figure 1 will support EC software in some way or another. An SE defect in any of these five layers would create a problem. For example, if the operating system is poorly engineered, the technology that sits on this platform will give piecemeal functionality at best. The problem is further complicated by piecemeal "patch" solutions. These piecemeal solutions can severely affect the usability of the Web, for example by giving cryptic error messages, installing add-ons that affect some unknown setting that the users do not understand, or installing add-ons that require a particular bit of hardware or software to be present.

Figure 1. EC Web application platform (adapted from Hurst & Gellady, 2000)



The View of HCI Advocates

Human-computer interaction is concerned with the way in which computers can be used to support human beings engaged in particular activities. HCI thus involves the specification, design, implementation, and evaluation of interactive software systems in the context of the user's task and work (Preece, Rogers, & Sharp, 2002; Preece, Rogers, Sharp, Benyon, Holland, & Carey, 1994; Shneiderman, 1998).

An aspect related to HCI is interaction design. Interaction design is the process of designing interactive products to support people in their everyday and work lives. In particular, it is about creating user experiences that enhance and extend the way people work, communicate, and interact (Preece et al., 2002).

As stated earlier, it is the users' experience that affects their activities on the Web. The advocates of HCI are intent on discovering the key to successful user experiences and so the concept of usability is intensively investigated in HCI. The ISO 9241-11 standard (1999) defines usability as the following: the extent to which a product can be used by a specified set of users, to achieve specified goals (tasks) with effectiveness, efficiency, and satisfaction in a specified context of use.

INTEGRATED USABILITY

Several researchers have produced sets of generic usability principles, which can be used in improving software (e.g., Mayhew, 1999; Preece et al., 1994; Shneiderman, 1998, 2000). Some of these usability principles are: learnability, visibility, consistency and standards, flexibility, robustness, responsiveness, feedback, constraints, mappings, affordances, stability, simplicity, help, and documentation. Unfortunately, the definitions of such design and usability principles are mostly too broad or general, and, in some cases, very vague. Some of these principles have been adapted for EC (see for example Badre, 2002). It has been shown repeatedly that general usability advice is not effective on its own when designing systems for a context-specific environment. Therefore, it is generally difficult for a non-usability expert or a novice to apply these principles in a particular domain and situation, taking into account the unique factors that give rise to problems in that domain.

We argue that usability advice should be linked to a context-specific environment. For example, if a designer is interested in enticing surfers to stop browsing and engage in transactions, the designer would be well advised to make different design choices for an Internet banking site than for an online library. So, the design of a site for Pick 'n Pay (supermarket chain), ABSA (commercial bank), and the University of South Africa's library should therefore be approached differently.

The HCI proponents also propose certain life cycle models. Williges, Williges, and Elkerton (1987), for example, have produced a model of development to rectify some of the problems in the "classic" life cycle model of SE. In this approach, HCI principles and interface design drive the whole process. Other such life cycle models include the Star model of Hartson and Hix (1989), the Usability Engineering life cycle of Mayhew (1999), and the Interaction Design model of Preece et al. (2002). These methods also introduce various strategies for the development of effective user interfaces. The argument for putting forward these alternative development models is that by spotting user requirements early on in the development cycle, there will be less of a demand for code generation and modification in the later stages of systems development.

FUTURE TRENDS

Standards can serve as good anchor points to focus the dialogues and collaborative activities. However, the existing standards are rather incon-

sistent and thus confusing. More efforts should be invested to render these tools more usable and useful. Specifically, it is worthy to develop implementation strategies for situating or localizing the standards so that they can be applied effectively in particular contexts (Law, 2003).

CONCLUSION

Both the SE proponents and the HCI proponents have a point with regard to their approach. SE proponents try to produce a workable solution and HCI proponents try to develop a usable solution. The two approaches are not mutually exclusive. A workable solution may not be a usable solution, and a usable solution may not be a workable solution. The problem is that the HCI advocates are isolated from their SE colleagues, who in turn ignore the HCI advocates. The HCI advocates use a "blinder approach" in their attempt to develop software by only focusing on the HCI aspects of the design of software, while the SE developers are concerned with a satisfactory solution. The aspects of Figure 1 will in effect influence the HCI advocates' approaches as well as the SE advocates' approaches for designing software for the Web. The uncertainty aspect has to be factored into the design process.

REFERENCES

Badre, N. A. (2002). Shaping Web usability: Interaction design in context. Boston: Addison-Wesley.

Hartson, H. R., & Hix, D. (1989). Human-computer interface development: Concepts and systems for its management. ACM Computing Surveys, 21, 5-92.

Hurst, M., & Gellady, E. (2000). White paper one: Building a great customer experience to develop brand, increase loyalty and grow revenues. Creativegood. Retrieved September 14, 2003, from http://www.creativegood.com

ISO 9241-11 (1999). Ergonomic requirements for office work with visual display terminals. Part 11 Guidance on usability. Retrieved from http://www.iso.org./iso/en/catalogue DetailPage. catalogueDetail?csnumber =16883&icsi=13&lcsz=180&lc53=

Law, E. L. C. (2003). Bridging the HCI-SE gap: Historical and epistemological perspectives. Paper presented at the INTERACT Workshop, Zürich, Switzerland.

Mayhew, D. J. (1999). The usability engineering lifecycle: A practitioner's handbook for user interface design. San Francisco: Morgan Kaufmann.

Mullet, D. (1999). The software crisis. University of North Texas. Retrieved October 20, 2003, from http://www.unt.edu/benchmarks/archives/1999/july99/crisis.htm

Preece, J., Rogers, Y., & Sharp, H. (2002). Interaction design: Beyond human-computer interaction. New York: John Wiley & Sons.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). Human-computer interaction. Harlow, UK: Addison-Wesley.

Pressman, R. S. (2000a). Software engineering: A practitioner's approach (5th ed.). London: McGraw Hill.

Pressman, R. S. (2000b). What a tangled Web we weave. IEEE Software, (January/February), 18-21.

Schach, S. R. (2002). Object-oriented and classical software engineering (5th ed.). Boston: McGraw Hill.

Shelly, B. G., Cashman, J. T., & Rosenblatt, J. H. (2003). Systems analysis and design (5th ed.). Australia: Thomson: Course Technology.

Shneiderman, B. (1992). Designing the user interface: Strategies for effective human-computer interaction. Reading, MA: Addison Wesley.

Shneiderman, B. (1998). Design the user interface: Strategies for effective human-computer interaction (3rd ed.). Reading, MA: Addison-Wesley.

Shneiderman, B. (2000). Universal usability. Communication of the ACM, 43(5), 84-91.

Stair, R. M. (1992). Principles of information systems: A managerial approach. Boston: Boyd & Fraser.

Turban, E., Rainer, R. K., & Potter, E. R. (2001). Introduction to information technology. New York: John Wiley & Sons.

Vanderdonckt, J., & Harning, M. B. (2003, September 1-2). Closing the gaps: Software engineering and human-computer interaction. Interact 2003 Workshop, Zurich, Switzerland. Retrieved in July 2004, from http://www.interact2003.org/workshops/ws9-description.html

Whitten, L. J., Bentley, D. L., & Dittman, C. K. (2001). Systems analysis and design methods (5th ed.). Boston: McGraw-Hill Irwin.

Williges, R. C., Williges, B. H., & Elkerton, J. (1987). Software interface design. In G. Salvendy (Ed.), Handbook of human factors (pp. 1414-1449). New York: John Wiley & Sons.

KEY TERMS

Information Systems: First known as business data processing (BDP) and later as management information systems (MIS). The operative word is "system" because it combines technology, people, processes, and organizational mechanisms for the purpose of improving organizational performance.

Interaction Design: The process of designing interactive products to support people in their everyday and work lives.

ISO 9241-11: This part of ISO 9241 introduces the concept of usability but does not make specific recommendations in terms of product attributes. Instead, it defines usability as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

Requirements' Gathering: The process of eliciting the overall requirements of a product from the customer.

Software Engineering: Concerned with the development of software systems using sound engineering principles for both technical and non-technical aspects. Over and above the use of specification, design and implementation techniques, human factors and software management should also be addressed.

Usability: The ISO 9241-11 standard definition of usability identifies three different aspects: (1) a specified set of users, (2) specified goals (asks) which have to be measurable in terms of effectiveness, efficiency, and satisfaction, and (3) the context in which the activity is carried out.

This work was previously published in Encyclopedia of Human Computer Interaction, edited by C. Ghaoui, pp. 548-552, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 1.14
Software Agent Technology: An Overview

Chrysanthi E. Georgakarakou
University of Macedonia, Greece

Anastasios A. Economides
University of Macedonia, Greece

ABSTRACT

This chapter provides an overview of the rapidly evolving area of software agents and presents the basic aspects of applying the agent technology to virtual enterprises (VE). As the field of software agents can appear chaotic, this chapter briefly introduces the key issues rather than present an in-depth analysis and critique of the field. In addition to, this chapter investigates the application of agent technology to virtual enterprises and presents current research activity that focuses on this field serving as an introductory step. Furthermore, this chapter makes a list of the most important themes concerning software agents and the application of agent technology to virtual enterprises apposing some order and consistency and serve as a reference point to a large body of literature.

INTRODUCTION

The aim of this chapter is to survey some key research issues in the software agents' area. It annotates several researchers' opinions on many areas concerning software agents trying to give a more documentary point of view of each argued subject. Its main goal is to provide an overview of the rapidly evolving area of software agents serving as a reference point to a large body of literature and outlining the key aspects of software agent technology. While this chapter does not act as an introduction to all the issues in the software agents' field, it intends to point the reader at the primary areas of interest. In addition to, this chapter investigates the application of agent technology to virtual enterprises. It presents basic aspects of applying agent technology to virtual enterprises serving as an introductory step.

First of all, this overview chapter attempts to answer the question of what a software agent is. Secondly, it analyzes the three technologies that distributed artificial intelligence (DAI) has evolved: (1) multi-agent system (MAS), (2) distributed problem solving (DPS), and (3) parallel AI (PAI). Thereinafter, it makes the distinction between single agent and multi-agent systems analyzing their dimensions. In addition to, it goes through the broad spectrum of agent properties.

Furthermore, it discusses the most acknowledged classification schemes or taxonomies (typologies) of software agents proposed in the agent research community. Moreover, it presents the most well known agent architecture classification schemes arguing about each distinct architecture. Besides, it explores the two most important agent communication approaches: (1) communication protocols, and (2) evolving languages. It also discusses about a number of languages for coordination and communication that have been proposed. It argues about possible implementations of agent transportation mechanisms as well. Further, it annotates prominent ontology specification languages and editors for ontology creation and maintenance. Then, it lists and argues standard languages and several prototype languages for implementing agent-based systems that have been proposed for constructing agent-based systems. Afterwards, it presents a number of tools and platforms that are available and support activities or phases of the process of agent-oriented software development. Next, it examines several agent oriented software engineering (AOSE) methodologies that have been proposed to assist engineers to create agent-based systems. At the end, it investigates the application of the agent technology to virtual enterprises, answering the question of why to use agents in virtual enterprises and presenting the current research activity that focuses on the agent technology applied to virtual enterprises.

BACKGROUND

As software agents comprise a prominent scientific area of research activity, a plethora of researchers have investigated them and stated their own point of view. Nwana and Ndumu (1996) mention that software agent technology is a rapidly developing area of research. According to Wooldridge and Jennings (1995), the concept of an agent has become important in both artificial intelligence (AI) and mainstream computer science. Oliveira, Fischer, and Stepankova (1999) observe that for some time now agent-based and multi-agent systems (MASs) have attracted the interest of researchers far beyond traditional computer science and artificial intelligence (AI).

Although software agent technology demonstrates expeditious advancement, there is a truly heterogeneous body of work being carried out under the "agents" banner (Nwana & Ndumu, 1996). Nwana and Ndumu (1996) introduce software agent technology by overviewing the various agent types currently under investigation by researchers. Nwana (1996) largely reviews software agents, and makes some strong statements that are not necessarily widely accepted by the agent community. Nwana (1996) presents a typology of agents, next places agents in context, defines them and overviews critically the rationales, hypotheses, goals, challenges, and state-of-the-art demonstrators of the various agent types of the proposed typology. Besides, Nwana (1996) attempts to make explicit much of what is usually implicit in the agents' literature and proceeds to overview some other general issues which pertain to all the types of agents in the typology.

Agent-based and multi-agent systems (MASs) have attracted the researchers' interest to great extents. Oliveira et al. (1999) try to identify focal points of interest for researchers working in the area of distributed AI (DAI) and MAS as well as application oriented researchers coming from related disciplines, for example, electrical and mechanical engineering. They do this by presenting key research topics in DAI and MAS research and by identifying application domains in which the DAI and MAS technologies are most suitable.

Sycara (1998) presents some of the critical notions in MASs and the research work that has addressed them and organizes these notions around the concept of problem-solving coherence. Sycara (1998) believes that problem-solving coherence is one of the most critical overall characteristics that an MAS should exhibit.

Jennings et al. (1998) provide an overview of research and development activities in the field of autonomous agents and multi-agent systems. They aim to identify key concepts and applications, and to indicate how they relate to one another. Some historical context to the field of agent-based computing is given, and contemporary research directions are presented. Finally, a range of open issues and future challenges are highlighted.

Wooldridge and Jennings (1995) aim to point the reader at what they perceive to be the most important theoretical and practical issues associated with the design and construction of intelligent agents. For convenience, they divide these issues into three areas (agent theory, agent architectures, and agent languages). Their paper is not intended to serve as a tutorial introduction to all the issues mentioned and includes a short review of current and potential applications of agent technology.

Wooldridge (1998) provides an introductory survey of agent-based computing. The article begins with an overview of micro-level issues in agent-based systems: issues related to the design and construction of individual intelligent agents. The article then goes on to discuss some macrolevel issues; issues related to the design and construction of agent societies. Finally, the key application areas for agent technology are surveyed.

An article that should not be omitted at this point is Weiß's (2002) paper. Weiß (2002) offers a guide to the broad body of literature of agent-oriented software engineering (AOSE). The guide, which is intended to be of value to both researchers and practitioners, is structured according to key issues and key topics that arise when dealing with AOSE: methods and frameworks for requirements engineering, analysis, design, and implementation; languages for programming, communication and coordination, and ontology specification; and development tools and platforms.

On the other hand, considering the agent technology application to virtual enterprises, Jennings, Norman and Faratin (1998) exhibit considerable concepts. They argue the case of the agent-based approach showing how agent technology can improve efficiency by ensuring that business activities are better scheduled, executed, monitored, and coordinated.

According to Camarinha-Matos (2002), multi-agent systems represent a promising approach to both model and implement the complex supporting infrastructures required for virtual enterprises and related emerging organizations. The current status of application of this approach to industrial virtual enterprises, virtual communities, and remote supervision in the context of networked collaborative organizations is presented (Camarinha-Matos, 2002). Examples of relevant projects are provided and major challenges and open issues identified as well (Camarinha-Matos, 2002).

Petersen, Divitini, and Matskin (2001) describe how virtual enterprises can be modeled using the AGORA multi-agent architecture, designed for modelling and supporting cooperative work among distributed entities. They underline that the distributed and goal-oriented nature of the virtual enterprise provides a strong motivation for the use of agents to model virtual enterprises. They also mention the main advantages of their approach.

This chapter provides an overview of research activity regarding the scientific domain of software agents. As the field of software agents can appear chaotic, this chapter briefly introduces the key issues rather than present an in-depth analysis and critique of the field. References to more detailed treatments are provided. The purpose of this chapter is to make a list of the most important themes concerning software agents, apposing some order and consistency and serve as a reference point to a large body of literature. In addition to, this chapter makes an introduction of applying agent technology to virtual enterprises and describes current research activity that addresses the above-mentioned issue.

A BRIEF OVERVIEW OF SOFTWARE AGENT TECHNOLOGY

What is a Software Agent?

Software agent technology is a rapidly developing area of research and probably the fastest growing area of information technology (IT) (Jennings & Wooldridge, 1996; Nwana & Ndumu, 1996). Application domains in which agent solutions are being applied or researched into include workflow management, telecommunications network management, air-traffic control, business process reengineering, data mining, information retrieval/management, electronic commerce, education, personal digital assistants (PDAs), e-mail filtering, digital libraries, command and control, smart databases, and scheduling/diary management (Nwana & Ndumu, 1996).

Over the last years, many researchers in the area of agents have proposed a large variety of definitions for the "agent" term. It is stated that it is difficult to give a full definition for the note of agency. Nwana (1996) predicates there are at least two reasons why it is so difficult to define precisely what agents are. Firstly, agent researchers do not "own" this term in the same way as fuzzy logicians/AI researchers, for example, own the term "fuzzy logic"—it is one that is used widely in everyday parlance as in travel agents, estate agents, and so forth. Secondly, even within the software fraternity, the word agent is really an umbrella term for a heterogeneous body of research and development (Nwana, 1996). Concerning the agent definition, Nwana (1996) states:

When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. Given a choice, we would rather say it is an umbrella term, meta-term or class, which covers a range of other more specific agent types, and then go on to list and define what these other agent types are. This way, we reduce the chances of getting into the usual prolonged philosophical and sterile arguments which usually proceed the former definition, when any old software is conceivably recastable as agent-based software. (p. 6)

Bradshow (1997) identifies two approaches to the definition of an agent as follows: (1) agent as an ascription—this approach is based on the concept that "agency cannot ultimately be characterized by listing a collection of attributes but rather consists fundamentally as an attribution on the part of some person," and (2) agent as a description: agents are defined by describing the attributes they should exhibit.

Jennings and Wooldridge (1996) offer a relatively loose notion of an agent as a self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives will be used here.

Wooldridge (1998) defines an intelligent agent as a system that enjoys the following four properties: autonomy (agents operate without the direct intervention of humans or others, and have control over their actions and internal state), social ability (agents are able to cooperate with humans or other agents in order to achieve their tasks), reactivity (agents perceive their environment, and respond in a timely fashion to changes that occur in it), and pro-activeness (agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative).

According to Hayes (1999), an agent is an entity (either computer or human) that is capable of carrying out goals, and is part of a larger community of agents that have mutual influence on each other. Agents may co-exist on a single processor, or they may be constructed from physically, but intercommunicating processors (such as a community of robots) (Hayes, 1999). The key concepts in this definition are that agents can act autonomously to some degree, and they are

part of a community in which mutual influence occurs (Hayes, 1999).

Distributed Artificial Intelligence (DAI) Technologies

Distributed artificial intelligence (DAI) is a subfield of artificial intelligence (AI) which is concerned with a society of problem solvers or agents interacting in order to solve a common problem: computers and persons, sensors, aircraft, robots, and so forth (Green, Hurst, Nangle, Cunningham, Somers, & Evans, 1997). Such a society is termed a multi-agent system, namely, a network of problem solvers that work together to solve problems that are beyond their individual capabilities (Green et al., 1997). Software agents have evolved from multi-agent systems (MAS), which in turn form one of three broad areas which fall under DAI, the other two being distributed problem solving (DPS) and parallel AI (PAI) (Nwana, 1996). Therefore, agents inherit potential benefits from both DAI, for example modularity, speed, reliability and AI (e.g., operation at knowledge level, easier maintenance, reusability, platform independence) (Nwana, 1996).

Agent Systems

Jennings et al. (1998) state that an agent-based system is a system in which the key abstraction used is that of an agent. In principle, an agent-based system might be conceptualized in terms of agents, but implemented without any software structures corresponding to agents at all (Jennings et al., 1998). A parallel with object-oriented software can be drawn, where it is entirely possible to design a system in terms of objects, but to implement it without the use of an object-oriented software environment (Jennings et al., 1998). But this would at best be unusual, and at worst, counter-productive (Jennings et al., 1998). According to Jennings et al. (1998), a similar situation exists with agent technology and they therefore expect an agent-based system to be both designed and implemented in terms of agents.

An agent-based system may contain one or more agents (Jennings et al., 1998). There are cases in which a single agent solution is appropriate (Jennings et al., 1998). However, the multi-agent case—where the system is designed and implemented as several interacting agents—is arguably more general and more interesting from a software engineering standpoint (Jennings et al., 1998). Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives and/or multiple problem solving entities (Jennings et al., 1998). Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions (Jennings et al., 1998). Examples of common types of interactions include: cooperation (working together towards a common aim), coordination (organizing problem solving activity so that harmful interactions are avoided or beneficial interactions are exploited), and negotiation (coming to an agreement which is acceptable to all the parties involved) (Jennings et al., 1998).

As the technology matures and endeavors to attack more complex, realistic, and large-scale problems, the need for systems that consist of multiple agents that communicate in a peer-to-peer fashion is becoming apparent (Sycara, 1998). The most powerful tools for handling complexity are modularity and abstraction (Sycara, 1998). Multi-agent systems (MASs) offer modularity (Sycara, 1998). If a problem domain is particularly a complex, large, or unpredictable, then the only way it can reasonably be addressed is to develop a number of functionally specific and (nearly) modular components (agents) that are specialized at solving a particular problem aspect (Sycara, 1998). In MASs, applications are designed and developed in terms of autonomous software entities (agents) that can flexibly achieve their objectives by interacting with one another

in terms of high-level protocols and languages (Zambonelli, Jennings, & Wooldridge, 2003). An MAS can be defined as a collection of, possibly heterogeneous, computational entities, having their own problem solving capabilities and which are able to interact among them in order to reach an overall goal (Oliveira et al., 1999).

Agent Properties

A software agent is a computer system situated in an environment that acts on behalf of its user and is characterised by a number of properties (Chira, 2003). Most researchers agree that autonomy is a crucial property of an agent. Alonso (2002) states about agents that it is precisely their autonomy that defines them. Furthermore, cooperation among different software agents may be very useful in achieving the objectives an agent has (Chira, 2003). According to the weak notion of agency given by Wooldridge and Jennings (1995) the most general way in which the term agent is used is to denote hardware or (more usually) software-based computer system that enjoys the following properties: autonomy, social ability, reactivity and pro-activeness (Wooldridge & Jennings, 1995). Jennings et al. (1998) identify three key concepts in their definition that they adapt from (Wooldridge & Jennings, 1995): situatedness, autonomy, and flexibility (by the term flexible they mean that the system is responsive, pro-active and social). For Wooldridge (1998), an intelligent agent is a system that enjoys autonomy, social ability, reactivity, and pro-activeness. He also refers to the fact that other researchers argue that different properties, such as mobility, veracity, benevolence, rationality and learning, should receive greater emphasis.

An agent may possess many properties in various combinations. In continuance, we enumerate and define all the properties that we adopt for the purposes of this research:

1. **Autonomy:** It means that the agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state (Sycara, 1998).
2. **Reactivity or situatedness or sensing and acting:** It means that the agent receives some form of sensory input from its environment, and it performs some action that changes its environment in some way (Chira, 2003; Sycara, 1998).
3. **Proactiveness or goal directed behavior:** It means that the agent does not simply act in response to its environment; it is able to exhibit goal-directed behavior by taking the initiative (Chira, 2003; Odell, 2000; Wooldridge & Jennings, 1995).
4. **Social ability:** It means that the agent interacts and friendliness or pleasant social relations mark this interaction; that is, the agent is affable, companionable or friendly (Odell, 2000).
5. **Coordination:** It means that the agent is able to perform some activity in a shared environment with other agents (Odell, 2000). Activities are often coordinated via plans, workflows or some other process management mechanism (Odell, 2000).
6. **Cooperation or collaboration:** It means that the agent is able to coordinate with other agents to achieve a common purpose; non-antagonistic agents that succeed or fail together (Odell, 2000).
7. **Flexibility:** It means that the system is responsive (the agents should perceive their environment and respond in a timely fashion to changes that occur in it), pro-active and social (Jennings et al., 1998).
8. **Learning or adaptivity:** It means that an agent is capable of (1) reacting flexibly to changes in its environment; (2) taking goal-directed initiative, when appropriate; and (3) learning from its own experience, its

environment, and interactions with others (Chira, 2003; Sycara, 1998).

9. **Mobility:** It means that the agent is able to transport itself from one machine to another and across different system architectures and platforms (Etzioni & Weld, 1995).
10. **Temporal continuity:** It means that the agent is a continuously running process, not a "one-shot" computation that maps a single input to a single output, then terminates (Etzioni & Weld, 1995).
11. **Personality or character:** An agent has a well-defined, believable personality and emotional state (Etzioni & Weld, 1995).
12. **Reusability:** Processes or subsequent instances can require keeping instances of the class agent for an information handover or to check and to analyze them according to their results (Horn, Kupries, & Reinke, 1999).
13. **Resource limitation:** An agent can only act as long as it has resources at its disposal (Horn et al., 1999). These resources are changed by its acting and possibly also by delegating (Horn et al., 1999).
14. **Veracity:** It is the assumption that an agent will not knowingly communicate false information (Wooldridge, 1998; Wooldridge & Jennings, 1995).
15. **Benevolence:** It is the assumption that agents do not have conflicting goals and that every agent will therefore always try to do what is asked of it (Wooldridge, 1998; Wooldridge & Jennings, 1995).
16. **Rationality:** It is the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved—at least insofar as its beliefs permit (Wooldridge, 1998; Wooldridge & Jennings, 1995).
17. **Inferential capability:** An agent can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation and/or other agents (Bradshow, 1997).
18. **"Knowledge-level" communication ability:** The ability to communicate with persons and other agents with language more resembling humanlike "speech acts" than typical symbol-level program-to-program protocols (Bradshow, 1997).
19. **Prediction ability:** An agent is predictive if its model of how the world works is sufficiently accurate to allow it to correctly predict how it can achieve the task (Goodwin, 1993).
20. **Interpretation ability:** An agent is interpretive if it can correctly interpret its sensor readings (Goodwin, 1993).
21. **Sound:** An agent is sound if it is predictive, interpretive and rational (Goodwin, 1993).
22. **Proxy ability:** An agent can act on behalf of someone or something that is, acting in the interest of, as a representative of, or for the benefit of, some entity (Odell, 2000).
23. **Intelligence:** The agent's state is formalized by knowledge and interacts with other agents using symbolic language (Odell, 2000).
24. **Unpredictability:** An agent is able to act in ways that are not fully predictable, even if all the initial conditions are known (Odell, 2000). It is capable of nondeterministic behavior (Odell, 2000).
25. **Credibility:** An agent has a believable personality and emotional state (Odell, 2000).
26. **Transparency and accountability:** An agent must be transparent when required, but must provide a log of its activities upon demand (Odell, 2000).
27. **Competitiveness:** An agent is able to coordinate with other agents except that the success of one agent implies the failure of others (Odell, 2000).
28. **Ruggedization:** An agent is able to deal with errors and incomplete data robustly (Odell, 2000).

29. **Trustworthiness:** An agent adheres to laws of robotics and is truthful (Odell, 2000).

Agent Typology

Agents may be usefully classified according to the subset of these properties that they enjoy (Franklin & Graesser, 1996). There are, of course, other possible classifying schemes (Franklin & Graesser, 1996). For example, software agents might be classified according to the tasks they perform, for example, information gathering agents or email filtering agents (Franklin & Graesser, 1996). Or, they might be classified according to their control architecture (Franklin & Graesser, 1996). Agents may also be classified by the range and sensitivity of their senses, or by the range and effectiveness of their actions, or by how much internal state they possess (Franklin & Graesser, 1996).

There are several classification schemes or taxonomies proposed in the agent research community from which the following three are well acknowledged (Chira, 2003): (1) Gilbert's scope of intelligent agents (Bradshow, 1997), (2) Nwana's (1996) primary attributes dimension typology, and (3) Franklin's and Graesser's (1996) agent taxonomy.

A typology refers to the study of types of entities and there are several dimensions to classify existing software agents (Nwana, 1996). Agents may be classified according to (Bradshow, 1997): (1) mobility, as static or mobile, (2) presence of a symbolic reasoning model, as deliberative or reactive, (3) exhibition of ideal and primary attributes, such as autonomy, cooperation and learning, (4) roles, as information or Internet, (5) hybrid philosophies, which combine two or more approaches in a single agent, and (6) secondary attributes, such as versatility, benevolence, veracity, trustworthiness, temporal continuity, ability to fail gracefully and mentalistic and emotional qualities (Nwana, 1996).

Nwana (1996) identifies seven types of agents (Chira, 2003). Next, we enumerate and describe each agent type:

1. **Collaborative agents:** They are "able to act rationally and autonomously in open and time-constrained multi-agent environments" (Chira, 2003; Nwana, 1996). Key characteristics: autonomy, social ability, responsiveness, and pro-activeness (Chira, 2003; Nwana, 1996).
2. **Interface agents:** They support and assist the user when interacting with one or more computer applications by learning during the collaboration process with the user and with other software agents (Chira, 2003; Nwana, 1996). Key characteristics: autonomy, learning (mainly from the user but also from other agents), and cooperation with the user and/or other agents (Chira, 2003; Nwana, 1996).
3. **Mobile agents:** They are autonomous software programs capable of roaming wide area networks (such as WWW) and cooperation while performing duties (e.g., flight reservation, managing a telecommunications' network) on behalf of its user (Chira, 2003; Nwana, 1996). Key characteristics: mobility, autonomy, and cooperation (with other agents—e.g., to exchange data or information) (Chira, 2003; Nwana, 1996).
4. **Information/internet agents:** They are designed to manage, manipulate or collate the vast amount of information available from many distributed sources (information explosion) (Chira, 2003; Nwana, 1996). These agents "have varying characteristics: they may be static or mobile; they may be non-cooperative or social; and they may or may not learn" (Chira, 2003; Nwana, 1996).
5. **Reactive agents:** They act/respond to the current state of their environment based on a stimulus-response scheme (Chira, 2003; Nwana, 1996). These agents are relatively simple and interact with other agents in basic

ways but they have the potential to form more robust and fault tolerant agent-based systems (Chira, 2003; Nwana, 1996). Key characteristics: autonomy and reactivity (Chira, 2003; Nwana, 1996).

6. **Hybrid agents:** They combine two or more agent philosophies into a single agent in order to maximize the strengths and minimize the deficiencies of the most relevant techniques (for a particular purpose) (Chira, 2003; Nwana, 1996).
7. **Smart agents:** They are equally characterised by autonomy, cooperation, and learning (Chira, 2003; Nwana, 1996).

According to Nwana (1996), there are some applications that combine agents from two or more of the above types. Nwana (1996) refers to these as heterogeneous agent systems. This category of agent systems is generally referred to (by most researchers) as multi-agent systems (Chira, 2003).

Agent Architectures

Researchers working in the area of agents' architectures are concerned with the design and construction of agents that enjoy the properties of autonomy, reactivity, pro-activeness, and social ability (Wooldridge, 1998). Wooldridge (1999) states that agent architecture is essentially a map of the internals of an agent—its data structures, the operations that may be performed on these data structures, and the control flow between these data structures**.** Three classes of agent architectures can be identified (Wooldridge & Jennings, 1995): (1) deliberative or symbolic architectures are those designed along the lines proposed by traditional, symbolic AI, (2) reactive architectures are those that eschew central symbolic representations of the agent's environment, and do not rely on symbolic reasoning, and (3) hybrid architectures are those that try to marry the deliberative and reactive approaches (Wooldridge, 1998). Wooldridge and Jennings (1995) indicate that agent architectures can be viewed as software engineering models of agents and identify the above-mentioned classes of agent architectures.

Wooldridge (1999) considers four classes of agents. In our opinion most agents follow one of these four architectural classes. To continue, we enumerate and give a short description of each class:

1. **Logic based agents:** In which decision-making is realized through logical deduction (Wooldridge, 1999).
2. **Reactive agents:** In which decision-making is implemented in some form of direct mapping from situation to action (Wooldridge, 1999).
3. **Belief-desire-intention (BDI) agents:** In which decision-making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent (Wooldridge, 1999).
4. **Layered architectures:** In which decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction (Wooldridge, 1999).

Agent Communication Approaches

One of the most important features of an agent is interaction. In other words, agents recurrently interact to share information and to perform tasks to achieve their goals (Kostakos & Taraschi, 2001). Without communication, different agents cannot know from each other who is doing what and how they can cooperate (Bussink, 2004). Therefore communication is a must if we want to set up a useful multi-agent system (Bussink, 2004).

There are several approaches to how this communication can take shape (Bussink, 2004). The two most important approaches are communication using communication protocols, and communication using an evolving language

(Bussink, 2004). Both techniques have their advantages and disadvantages (Bussink, 2004). In industrial applications communication protocols will be the best practice, but in systems where homogeneous agents can work together language evolution is a good option (Bussink, 2004). The basis for language evolution is in human communication (Bussink, 2004). The agent languages consist of grammars and vocabularies, just like any human language (Bussink, 2004). Some researchers even do research in the area of language evolution using agents in order to get more understanding of how human communication has evolved (Bussink, 2004). For a long time, the only way agents communicated was using communication protocols (Bussink, 2004). Therefore research often focussed on this area and a lot of specifications have been written (Bussink, 2004). Because of the formal nature of protocols, there are a quite a few widely known and used standards (Bussink, 2004).

Agent Communication Languages (ACLs)

The difficulty to precisely handle coordination and communication increases with the size of the agent-based software to be developed. A number of languages for coordination and communication have been proposed. Weiβ (2002) enumerates a list of such languages. Next, we enumerate and describe the most prominent examples of agent communication languages (ACLs) according to Weiβ (2002):

1. **Knowledge query and manipulation language (KQML):** It is perhaps the most widely used agent communication language (Weiβ, 2002).
2. **ARCOL ("ARTIMIS communication language"):** It is the communication language used in the ARTIMIS system (Weiβ, 2002). ARCOL has a smaller set of communication primitives than KQML, but these can be composed (Weiβ, 2002).
3. **FIPA agent communication language (FIPA-ACL):** It is an agent communication language that is largely inuenced by ARCOL (Weiβ, 2002). Together FIPA-ACL, ARCOL, and KQML establish a quasi standard for agent communication languages (Weiβ, 2002).
4. **Knowledge interchange format (KIF):** It is a logic-based language that has been designed to express any kind of knowledge and metaknowledge (Weiβ, 2002). KIF is a language for content communication, whereas languages like KQML, ARCOL, and FIPA-ACL are for intention communication (Weiβ, 2002).
5. **Domain independent COOrdination language (COOL):** It aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management (Weiβ, 2002). Languages like COOL can be thought of as supporting a coordination/communication (or "protocol-sensitive") layer above intention communication (Weiβ, 2002).

Apart from these most prominent languages, several others showing unique properties have been proposed (Weiβ, 2002). Some of the above-mentioned languages follow:

1. Interagent communication language (ICL) (Weiβ, 2002)
2. AgentTalk (Weiβ, 2002)
3. Communication and coordination language (CoLa) (Weiβ, 2002)
4. Tuple centres spread over networks (TuCSoN) (Weiβ, 2002)
5. LuCe (Weiβ, 2002)
6. Simple thread language ++ (STL++) (Weiβ, 2002)
7. Strictly declarative modelling language (SDML) (Weiβ, 2002)

Agent Transportation Mechanisms

In agent environments, messages should be schedulable, as well as event driven (OMG Agent Working Group, 2000). They can be sent in synchronous or asynchronous modes (OMG Agent Working Group, 2000). The transportation mechanism should support unique addressing as well as role-based addresses (OMG Agent Working Group, 2000). Lastly, the transportation mechanism must support unicast, multicast, and broadcast modes and such services as broadcast behavior, nonrepudiation of messages, and logging (OMG Agent Working Group, 2000). Next, we enumerate and describe possible implementations of the agent transportation mechanism:

1. **Common object request broker architecture (COBRA):** It is the acronym for common object request broker architecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks (The Object Management Group). Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network (The Object Management Group).
2. **Object management group (OMG) messaging services:** OMG is an international trade association incorporated as a nonprofit in the United States (The Object Management Group). The OMG is currently specifying a new messaging service (The CORBA Object Group Service).
3. **JAVA messaging service:** It is the standard API for sending and receiving messages (Creative Science Systems).
4. **Remote method invocation (RMI):** It defines and supports a distributed object model for the Java language hiding the ORB from the programmer and providing an API for the development of distributed applications (Bracho, Matteo, & Metzner, 1999). Java remote method invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts (Java.sun.com).
5. **Distributed component object model (DCOM):** Microsoft® Distributed COM (DCOM) extends the component object model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet (DCOM Technical Overview). With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application (DCOM technical overview).
6. **Enterprise Java Beans events:** The newest Java component model is Enterprise Java Beans (Tuukka Vartiainen, Java Beans, and Enterprise Java Beans). Besides its name, the Java language and the idea of component based software re-use; it has little or no similarities with the Java Beans standard (Tuukka Vartiainen, Java Beans, and Enterprise Java Beans). Enterprise Java Beans are located on the server and they support a distributed programming model that could be described as a flexible, two-way, object-oriented version of traditional client-server programming (Tuukka Vartiainen, Java Beans, and Enterprise Java Beans).

Ontology Languages and Editors

Besides an ACL, a common understanding of the concepts used among agents is necessary for

a meaningful agent communication. A common ontology is required for representing the knowledge from various domains of discourse (OMG Agent Working Group, 2000). The ACL remains just syntax without a shared common ontology containing the terms used in agent communication and the knowledge associated with them (Nwana & Wooldridge, 1996). Next, we enumerate and describe the most elaborated examples of such languages according to Weíβ (2002):

1. **Ontolingua and frame logic:** They are frame-based languages (Weíβ, 2002). Both of them extend first-order predicate logics (Weíβ, 2002). The key modeling primitive of these languages are frames as known from artificial intelligence (Weíβ, 2002).
2. **CLASSIC and LOOM:** They are description logics that allow an intentional definition of concepts (Weíβ, 2002).
3. **CycL:** It extends first-order predicate logic and was developed to enable the specification of large common-sense ontologies (Weíβ, 2002).

In addition, we enumerate and describe the most prominent ontology specification languages that are conform to syntactic and semantic Web standards according to Weíβ (2002):

1. **Simple HTML ontology extension (SHOE):** It is a language that slightly extends HTML and enables a hierarchical classification of HTML documents and the specification of relationships among them (Weíβ, 2002).
2. **Ontology exchange language (XOL):** It is an XML- and frame-based language for the exchange of ontologies (Weíβ, 2002).
3. **Ontology inference layer (OIL):** It aims at unifying formal semantics as offered by description logics, rich modelling primitives as offered by frame-based languages, and the XML and RDF Web standards (Weíβ, 2002). OIL can be seen as an extension of XOL offering both an XML-based and an RDF-based syntax (Weíβ, 2002).
4. **DAML-ONT and DAML-OIL:** They are the DAML (DARPA Agent Markup Language) languages (Weíβ, 2002). DAML-OIL, which replaces DAML-ONT and represents the state of the art in the field, has well-defined model-theoretic and axiomatic semantics (Weíβ, 2002).

Furthermore, we enumerate and describe three good examples of editors for ontology creation and maintenance according to Weíβ (2002):

1. **Protégé:** It supports single-user ontology acquisition (Weíβ, 2002).
2. **Webonto:** It supports multiple-user ontology acquisition over the Web (Weíβ, 2002).
3. **OntoEdit:** It supports multilingual development of ontologies and multiple inheritances (Weíβ, 2002).

Languages for Constructing Agent-Based Systems

Most agent systems are probably written in Java and C/C++ (Weíβ, 2002). Apart from these standard languages, several prototype languages for implementing agent-based systems have been proposed that all aim at enabling a programmer to better realize agent-specific conceptions (Weiß, 2002). Three paradigms for implementing agent systems have been proposed: agent-oriented programming, market-oriented programming and interaction-oriented programming (Weiß, 2002). Weíβ (2002) lists some of most prominent and best understood prototype languages following the agent-oriented paradigm (references to these languages are provided in Weiß, 2002). Next, we enumerate and describe the above mentioned prototype languages:

1. **AGENT-0, PLACA and AGENT-K:** AGENT-0 realizes the basic ideas of the agent-oriented programming paradigm as formulated by Shoham (Weíβ, 2002). A language that extends AGENT-0 toward planning is PLACA, and a language that aims at integrating AGENT-0 and KQML is AGENT-K (Weíβ, 2002).
2. **Concurrent MetateM:** It allows specifying the intended behavior of an agent based on temporal logics (Weíβ, 2002).
3. **AgentSpeak (L):** It is a rule-based language that has a formal operational semantics and that assumes agents to consist of intentions, beliefs, recorded events, and plan rules (Weíβ, 2002). AgentSpeak (L) is based on an abstraction of the PRS architecture (Weíβ, 2002).
4. **3APL:** It incorporates features from imperative and logic programming (Weíβ, 2002). 3APL has well defined operational semantics and supports monitoring and revising of agent goals (Weíβ, 2002).
5. **ConGolog:** It is a concurrent logic-based language initially designed for high-level robot programming (Weíβ, 2002).
6. **Agent process interaction language (APRIL), multiagent interaction and implementation language (MAIL/MAI2L), and VIVA:** Other examples of languages following the agent-oriented programming paradigm (Weíβ, 2002).

Nwana and Wooldridge (1996) classify constructing agent application languages according to a typology. Next, we present the above-mentioned classification (Nwana & Wooldridge, 1996):

1. **Collaborative agents:** The actor language actors and the agent-oriented programming languages Agent-0 and Placa.
2. **Interface, information and mobile agents:** The scripting languages TCL/Tk, Safe-TCL, Safe-Tk, Java, Telescript, Active Web tools, Python, Obliq, April and Scheme-48.
3. **Reactive agents:** The reactive language RTA/ABLE.

However traditional languages are still used to construct agent applications (Nwana & Wooldridge, 1996). It is possible to implement agent-based systems in languages like Pascal, C, Lisp, or Prolog (Nwana & Wooldridge, 1996). But as a rule, one would not choose to do so because such languages are not particularly well suited to the job (Nwana & Wooldridge, 1996). Typically, object-oriented languages such as Smalltalk, Java, or C++ lend themselves more easily for the construction of agent systems (Nwana & Wooldridge, 1996). This is because the concept of an "agent" is not too distant from that of an "object": Agents share some properties with objects such as encapsulation, and frequently, inheritance and message passing (Nwana & Wooldridge, 1996). However, agents differ distinctly from objects vis-à-vis polymorphism (Nwana & Wooldridge, 1996).

Tools and Platforms

A number of tools and platforms are available that support activities or phases of the process of agent-oriented software development (Weiβ, 2002). Most of them are built on top of and integrated with Java (Weiβ, 2002). While almost all available tools and platforms have their focus on implementation support, some of them do also support analysis, design, and test/debugging activities (Weiβ, 2002).

Weíβ (2002) makes a list of such tools and platforms separating them into often-sited academic and research prototypes and into commercial products for development support. References to the following tools and a brief description as well can be found in (Weiβ, 2002). Next, we present the above-mentioned classification:

1. **Academic and research activity:** ZEUS, Java Agent DEvelopment framework (JADE), Lightweight extensible agent platform (LEAP), agenTool, RETSINA, Java agent template, lite (JATLite), FIPA-OS, MADKIT, SIM_AGENT, Java-based agent framework for multi-agent systems (JAFMAS), agent building shell (ABS), open agent architecture (OAA), and Agentis
2. **Commercial activity:** AgentBuilder, JACK, intelligent agent factory and grasshopper

Serenko and Detlor (2002) state about the term agent toolkit that each vendor uses its own explanation of the term and for the needs of their report define an agent toolkit as any software package, application or development environment that provides agent builders with a sufficient level of abstraction to allow them to implement intelligent agents with desired attributes, features and rules. Some toolkits may offer only a platform for agent development, whereas others may provide features for visual programming (Serenko & Detlor, 2002). Serenko and Detlor (2002) categorize the available agent toolkits on the market into four major categories. Next, we present the four categories and the representative toolkits of each category:

1. **Mobile agent toolkits:** Concordia, Gossip, FarGo, and IBM Aglets
2. **Multi-agent toolkits:** MadKit, ZEUS, JADE, JATLite, and MAST
3. **General purpose toolkits:** FIPA-OS and Ascape
4. **Internet agent toolkits:** Microsoft Agent, Voyager, and NetStepper

Agent-Oriented Software Engineering (AOSE) Methodologies

Agent researchers have produced methodologies to assist engineers to create agent-based systems (Agent-Oriented Software Engineering). Some researchers have taken agent theory as their starting point and have produced methodologies that are rooted in that theory (Agent-Oriented Software Engineering). Other researchers have taken object techniques as their point of departure and have enriched them to be suitable for agents (Agent-Oriented Software Engineering). Others have taken knowledge engineering concepts and have extended them (Agent-Oriented Software Engineering). Researchers also have tried to assemble methodologies by combining features from different methodologies (Agent-Oriented Software Engineering). Yet other researchers have produced methodologies based on both agent and object technologies (Agent-Oriented Software Engineering).

Methodologies having as background the agent and multi-agent technology are characterized by a clear focus on capturing social-level abstractions such as agent, group, or organization, that is, on abstractions that are above the conventional object level (Weiβ, 2002). Methodologies having as background the object orientation are characterized by the attempt to appropriately extend existing object-oriented techniques such that they also capture the notion of agency (Weiβ, 2002). Methodologies having engineering background knowledge are characterized by an emphasis on the identification, acquisition and modeling of knowledge to be used by the agent components of a software system (Weiβ, 2002).

The most popular approaches based on agent and multi-agent technology are the following:

1. **Generic architecture for information availability (GAIA):** This is a method that distinguishes between analysis and design and associates different models with these two phases (Weiβ, 2002). Gaia focuses on organizational aspects in terms of concepts such as roles, interactions and acquaintances (Weiβ, 2002).
2. **Societies in open and distributed agent spaces (SODA):** This is another good

example of an analysis and design method that concentrates on the social (interagent) aspects of agent systems and that employs the concept of coordination models (Weiβ, 2002).

3. **Cassiopeia:** This is a design method that distinguishes three levels of behavior—elementary, relational, and organizational—and aims at capturing both structural and dynamic aspects of the target system (Weiβ, 2002).
4. **Aalaadin:** This is a general analysis and design framework that has its focus on the organizational level of multi-agent systems and is built on the three core concepts of agents, groups, and roles (Weiβ, 2002).

The most popular approaches based on object-oriented technology are the following:

1. **KGR:** This is a design and specification method for a particular class of agents, namely, BDI agents (Weiβ, 2002).
2. **Multiagent systems engineering (MaSE):** This method covers design and initial implementation through two languages called agent modeling language (AgML) and agent definition language (AgDL) and builds upon OMT and UML (Weiβ, 2002).
3. **Multiagent systems iterative view engineering (MASSIVE):** This method covers analysis, design and code generation, and combines standard software engineering techniques such as multi-view modeling, round-trip engineering, and iterative enhancement (Weiβ, 2002).
4. **Agent-oriented analysis and design (AOAD):** This analysis and design method proposes the use of extended class responsibility cards (CRCs) and the use of both the object modelling technique (OMT) and the responsibility driven design (RDD) method known from object-oriented development (Weiβ, 2002).
5. **Multi-agent scenario-based (MASB):** MASB is an analysis and design method that covers issues of both objects and agents via behavior diagrams, data models, transition diagrams, and object life cycles (Weiβ, 2002).

The most popular approaches based on knowledge engineering technology are the following:

1. **Conceptual modelling of multi-agent systems (CoMoMAS):** This is an elaborated extension of the CommonKADS methodology, supporting analysis, design, and automated code generation (Weiβ, 2002).
2. **Multi-agent system commonKADS (MAS-CommonKADS):** This is another extension of CommonKADS that supports analysis and design of agent-oriented systems (Weiβ, 2002).

Other agent-oriented software engineering methodologies (AOSE) are tropos, agent-oriented analysis and design, agent modelling technique for systems of BDI agents, agent oriented methodology for enterprise modelling, a process for agent societies specification and implementation (PASSI), prometheus, AOR, ROADMAP, OPM /MAS, ingenias, DESIRE, AAII methodology, cooperative information agents design, adept, AUML, ADELFE, MESSAGE /UML, the styx agent methology, SABPO, expectation-oriented analysis and design (EXPAND) and ODAC (Cernuzzi, Cossentino, & Zambonelli, 2004; Iglesias, Garijo, & Gonzalez, 1999; Agent-Oriented Software Engineering; Faculty Science Unitn; Weiβ, 2002; Wooldridge & Ciancarini, 2000).

AGENTS IN VIRTUAL ENTERPRISES (VES)

What is a Virtual Enterprise?

The term, and the concept, "virtual enterprise" (VE) emerged already in the beginning of the 1990s and could be seen as the further optimization and perfection of the basic ideas about dynamic networking (Putnik, 2004). Although the virtual enterprise research represents a growing and multidisciplinary area it still lacks a precise definition of the concepts and an agreement on the used terminology (Camarinha-Matos, 2002). So far, there is no unified definition for this paradigm and a number of terms are even competing in the literature while referring to different aspects and scopes of virtual enterprises (Camarinha-Matos, 2002). Akin concepts are supported by Gijsen, Szirbik, and Wagner (2002); Freitas Mundim, Rossi, and Stocchetti (2000); Putnik (2004) and Petersen et al. (2001).

The definitions range from the virtual enterprise as a simple subcontracting network to the virtual enterprise as a dynamic network, in which the partners share that share resources, risks and even markets, and which operates in a virtual environment or with virtual agents (Putnik, 2004). According to Do, Halatchev, and Neumann (2000) a virtual enterprise is a form of cooperation of independent market players (enterprises, freelancers authorities, etc.) which combine their core competencies in order to manufacture a product or to provide a service. Marík and McFarlane (2005) conclude that a virtual enterprise represents a cluster of organizations collaborating to achieve one or more goals. Katz and Schuh (1999) define that the virtual enterprise is based on the ability to create temporary co-operations and to realize the value of a short business opportunity that the partners cannot (or can, but only to lesser extent) capture on their own. Other attempts at defining virtual enterprises are listed in (Petersen et al., 2001).

In our opinion, an interesting definition that we adopt is the following:

A goal-oriented constellation of (semi)autonomous distributed entities. Each entity, which can be an organization and/or an individual, attempts to maximize its own profits as well as contribute to defining and achieving the overall goals of the virtual enterprise. Virtual enterprises are not rigid organizational structures within rigid frameworks, but rather (heterogeneous) ensembles, continuously evolving over time. (Petersen et al., 2001, p. 2)

Why to Use Agents in Virtual Enterprises?

Marík and McFarlane (2005) state that a virtual enterprise might address problems ranging from simple membership to distributed inventory management and synchronization of supply, production, and distribution schedules. They also support that these problems are inherently distributed, with each organization willing to share only limited information and having its own business goals in conjunction with the overall goal. All the above statements orientate to an agent technology solution.

According to Jennings et al. (1998), considering a virtual enterprise, the domain involves an inherent distribution of data, problem solving capabilities and responsibilities. In addition, the integrity of the existing organizational structure and the autonomy of its sub-parts need to be maintained (Jennings et al., 1998). Moreover, interactions are fairly sophisticated, including negotiation, information sharing, and coordination (Jennings et al., 1998). Besides, the problem solution cannot be entirely prescribed (Jennings et al., 1998). According to Jennings et al. (1998), all the above observations motivate the choice of agents as a technology solution as well.

According to Fox and Gruninger (1998), the entrepreneurial and virtual nature of the agile

enterprise coupled with the need for people and information to have a strategic impact entails a greater degree of communication, coordination and cooperation within and among enterprises. In other words, the agile organization must be integrated (meaning by the term integrated the structural, behavioural and information integration of the enterprise) (Fox & Gruninger, 1998). Petersen et al. (2001) support that cooperation is required both to perform work and to adapt the constellation to the varying needs of the environment. They state that goal-oriented and distributed nature of virtual enterprises implies that there is no central control; rather, the control is decentralized. According to their opinion, the distributed and goal-oriented nature of the virtual enterprise provides a strong motivation for the use of agents to model virtual enterprises.

The following parallelism demonstrates remarkable interest. According to Rahwan, Kowalczyk, and Yang (2001) the virtual enterprise creation could be viewed as a Cooperative System design problem. A Cooperative System is a system in which a set of autonomous agents (computational and human) interact with each other through sharing their information, decision making capabilities and other resources, and distributing the corresponding workload among themselves, in order to achieve common and complementary goals (Camarinha-Matos & Afsarmanesh, 1998). The above parallelism motivates as well the agents as a technology solution.

The nature of agents, by definition, enables decentralized control of the enterprise, which is desirable in a dynamic and flexible environment, and the behavior of the complete enterprise emerges as a result of the behaviors of the individual agents (Petersen et al., 2001).

Another strong point in favor of the adoption of agents is their versatility (Petersen et al., 2001). They can play two main roles (Petersen et al., 2001). First, they provide a flexible means of modeling the virtual enterprise in terms of cooperative work among the agents (Petersen et al., 2001). Second, they can be used to provide active support to the members of the virtual enterprise (Petersen et al., 2001). Thus, agents being computational entities, the resulting model provides an easy and efficient passage to the computational support that is required by virtual enterprises (Petersen et al., 2001).

According to Marík and McFarlane (2005), MASs and relevant technologies consider each company as an agent able to carry out specific (usually quite complex) functions. The agents are registered with a certain platform and communicate in a standard agent communication language (Marík & McFarlane, 2005). Virtual enterprise formation, as well as the joint planning and scheduling activities, is based on jointly known negotiation rules and scenarios (Marík & McFarlane, 2005). These are very similar (or identical) to protocols or auctions in the MAS domain (Marík & McFarlane, 2005). The highly specialized members of a virtual enterprise, such as brokers or professional network organizers, can easily find their counterparts in the MAS community—for example, various middle agents and brokers (Marík & McFarlane, 2005). The negotiation and brokering algorithms that have proven useful for the MAS domain can serve to formalize (and later automate) the corresponding virtual enterprise processes (Marík & McFarlane, 2005). Specialized agents called meta-agents could also serve as tools both to help detect the network's less efficient parts or bottlenecks and to provide advice supporting the virtual enterprise's self-evolution in the desired direction (Marík & McFarlane, 2005). Virtual enterprise creation is analogous to coalition formation in the MAS domain (Marík & McFarlane, 2005).

They also support that MAS concept of knowledge sharing, which classifies knowledge as public, private, and semiprivate, has high potential for virtual enterprises. Requirements for keeping agents' knowledge confidential and preventing knowledge disclosure, as well as specific security principles used with MASs,

can be reused for virtual enterprises (Marík & McFarlane, 2005).

Current Research Activity Focusing on Agents in Virtual Enterprises (VE)

Virtual enterprises have recently received increasing attention. Due to the advancement of distributed information technology and the changing needs of the business community, enterprises are expected to be more agile and responsive (Petersen et al., 2001). Many current developments in multi-agent systems (MAS) are more and more focused on the production of robust development environments (Camarinha-Matos, 2002). Considerable efforts are also being put on standardization of architectures and communication languages, which are important requirements for the industrial application of the paradigm (Camarinha-Matos, 2002). We have observed that there is a remarkable body of literature that studies the application of agent technology to virtual enterprises as researchers pay enough attention to this scientific area of activity. In continuance of the study, some prominent research efforts follow.

According to Yonghe and Biqing (1999), decision and control processes within the domain of virtual enterprises have not received deserved attention till now. Based on agent technology, they bought up architecture for control and decision-making during the dynamic creation and operation of a virtual enterprise. An approach for integrating different business units is presented (Yonghe & Biqing, 1999). Prototype software simulating the design of a new product in a virtual enterprise is developed (Yonghe & Biqing, 1999).

Petersen, Jinghai, and Matskin (2003) present the virtual enterprise formation process as an agent interaction protocol and an approach to its implementation. They have focused on the selection of partners within the formation process in order to understand these interactions and the contents of the messages that are exchanged between the agents. Based on this, they describe how the AGORA multi-agent architecture can be used to support the formation of a virtual enterprise.

Gou, Huang, Liu, and Li (2001) propose an agent-based virtual enterprise model and provide the agent collaboration mechanisms under the model, thereby achieving the agent based virtual enterprise modeling and operation control. Their agent-based approach achieves distributed control over the whole business process execution of the virtual enterprise.

According to Fankhauser and Tesch (1999), negotiations encourage agents to reason about the interests of their opponents. Thus, negotiations suffer from counter speculations (Fankhauser & Tesch, 1999). Auctions apply to asymmetric trading only; they either favor the auctioneer or the bidders (Fankhauser & Tesch, 1999). Both mechanisms do not promote agents to tell the truth (Fankhauser & Tesch, 1999). Therefore, they propose to use a trust-broker to mediate between the agents. They introduce three symmetric, negotiation free one-step protocols to carry out a sequence of decisions for agents with possibly conflicting interests. The protocols achieve substantially better overall benefit than random or hostile selection, and they avoid lies (Fankhauser & Tesch, 1999). They analyze the protocols with respect to informed vs. uninformed lies, and with respect to beneficial vs. malevolent lies, and show that agents are best off to know and announce their true interests.

Gong and Wang's (2000) research is a contribution to the model of multi-agent system (MAS) for supporting the dynamic enterprise model (DEM). It separates the business process from the organizational structure (organizational structure tier and business process tier), models each of them as MAS, and coordinates agents by the "yellow page" mechanism (Gong & Wang, 2000). This model not only can regulate itself in terms of DEM, but also is centered on the coordination strategies between agents composing it (Gong

& Wang, 2000). It is believed that the model of MAS is a practical way to build flexible enterprise information system (Gong & Wang, 2000).

Chrysanthis, Znati, Banerjee, and Shi-Kuo (1999) view the establishment of a virtual enterprise as a problem of dynamically expanding and integrating workflows in decentralized, autonomous and interacting workflow management systems. They focus on the idea of mobile agents called adlets and their use in establishing virtual enterprises that involves advertising, negotiating and exchanging control information and data as well as its management.

Szirbik, Aerts, Wortmann, Hammer, and Goossenaerts (2000) propose a systematisation of the monitoring and control aspects in a virtual enterprise. As an instrument, they use the mobile agent paradigm, defining the concept of a mobile agent web (MA-web). According to them, one of the roles of the agents in this environment is to mediate negotiations between the parties of the virtual enterprise. They make some assumptions about the new behavior and code of conduct in the MA-web, such as the willingness to share data and knowledge.

Based on the analysis of why agent-based mechanism is suitable and only suitable for cross-domain cooperation of virtual enterprise, Zhang et al. (2004) propose a framework to implement it. In their framework, there is a service information supply-demand center that is in charge of service information management, and agent is responsible for cooperative partner selecting before cooperation and interaction during cooperation. The relevant key strategies and basic interaction models are also described (Zhang et al., 2004).

Ouzounis and Tschammer (2001) discuss concepts and technologies that are considered to satisfy key requirements of dynamic virtual enterprises, and propose DIVE, a framework for the specification, execution and management of shared business processes in dynamic virtual enterprises.

Suh et al. (2005) describe an open and flexible infrastructure to support dynamic collaboration among companies through the entire lifecycle of the virtual enterprise. The proposed approach is an agent-enhanced architecture on which the conversation model is grafted (Suh et al., 2005). The collaboration among enterprises is modelled by a collaboration policy, which is a machine-readable specification of a pattern of message exchange among agents participating in the collaboration (Suh et al., 2005).

FUTURE TRENDS

Luck, McBurney, and Gonzalez-Palacios (2006) stated a thorough and outstanding approach about the future of multi-agent systems. As we consider their point of view extremely prominent, we appose at this point of the chapter some parts of their findings (for a more complete investigation consult (Luck et al., 2006)). Luck et al. (2006) extrapolated future trends in multi-agent systems by classifying them into four broad phases (current, short-term future, medium-term future and long-term future) of development of multi-agent system technology over the next decade.

At first phase, multi-agent systems are typically designed by one design team for one corporate environment, with participating agents sharing common high-level goals in a single domain (Luck et al., 2006). These systems may be characterized as closed (Luck et al., 2006). The communication languages and interaction protocols are typically in-house protocols, defined by the design team prior to any agent interactions (Luck et al., 2006). Design approaches, as well as development platforms, tend to be ad hoc, inspired by the agent paradigm (Luck et al., 2006). There is also an increased focus on taking methodologies out of the laboratory and into development environments, with commercial work being done on establishing industrial-strength development techniques and notations (Luck et al., 2006).

In the short-term future, multi-agent systems will increasingly be designed to cross corporate boundaries, so that the participating agents have fewer goals in common, although their interactions will still concern a common domain, and the agents will be designed by the same team, and will share common domain knowledge (Luck et al., 2006). Standard agent communication languages will be used, but interaction protocols will be mixed between standard and nonstandard ones (Luck et al., 2006). Development methodologies, languages and tools will have reached a degree of maturity, and systems will be designed on top of standard infrastructures such as Web services or Grid services, for example (Luck et al., 2006).

In the medium term future, multi-agent systems will permit participation by heterogeneous agents, designed by different designers or teams (Luck et al., 2006). Any agent will be able to participate in these systems, provided their (observable) behavior conforms to publicly stated requirements and standards (Luck et al., 2006). However, these open systems will typically be specific to particular application domains (Luck et al., 2006). The languages and protocols used in these systems will be agreed and standardized (Luck et al., 2006).

In the long-term future, we will see the development of open multi-agent systems spanning multiple application domains, and involving heterogeneous participants developed by diverse design teams (Luck et al., 2006). Agents seeking to participate in these systems will be able to learn the appropriate behavior for participation in the course of interacting, rather than having to prove adherence before entry (Luck et al., 2006). Selection of communications protocols and mechanisms, and of participant strategies, will be undertaken automatically, without human intervention (Luck et al., 2006).

The above-mentioned aspect about future is enhanced with the AOSE Technical Forum Group's (2004) perception of the future trends in the area of agent-oriented software engineering. According to AOSE Technical Forum Group (2004), the research in the area of agent-oriented software engineering is still in its early stages, and several challenges need to be faced before agent-oriented software engineering becoming a widely accepted and a practically usable paradigm for the development of complex software systems. One possible way to identify and frame the key research challenges in the area of agent-oriented software engineering is to recognize that such challenges may be very different depending on the "scale of observation" adopted to model and build a software system (AOSE Technical Forum Group, 2004).

At one extreme, the micro scale of observation is that where the system to be engineered has to rely on the controllable and predictable behavior of (a typically limited number) individual agents, as well as on their mutual interactions (AOSE Technical Forum Group, 2004). There, the key engineering challenges are related to extending traditional software engineering approaches toward agent-oriented abstractions (AOSE Technical Forum Group, 2004). Brand new modeling and notational tools, as well as possibly brand new software process models may be needed (AOSE Technical Forum Group, 2004).

At the other extreme, the macro scale of observation is the one where a multi-agent system is conceived as a multitude of interacting agents, for which the overall behavior of the system, rather than the mere behavior of individuals, is the key of interest (AOSE Technical Forum Group, 2004). In this case, a discipline of agent-oriented software engineering should focus on totally different problems, and should be able to develop novel "systemic" approaches to software engineering, possibly getting inspiration from areas such as complex systems sciences and systemic biology (AOSE Technical Forum Group, 2004).

In between, the meso scale of observation is where the need of predictability and control, typical of the micro scale, clashes with the emergence of phenomena typical of the macro scale (AOSE

Technical Forum Group, 2004). Therefore, any engineering approach at the meso scale requires accounting for problems that are typical of both the micro and the macro scale, and possibly for new problems specific to the meso scale (AOSE Technical Forum Group, 2004). These include: identifying the boundaries of a systems—which may be challenging in the case of open multi-agent systems; electing trust as a primary design issue; identifying suitable infrastructures for multi-agent systems support (AOSE Technical Forum Group, 2004).

As concerns the virtual enterprises' scientific domain, we believe that agent technology has much to offer with respect to the formation and the operation of a virtual enterprise. According to Camarinha-Matos (2002), several challenges remain open for MAS requiring further research, such as support for the full life cycle of the virtual enterprise, adoption of contract-based coordination models, necessary integration of MAS with several other paradigms, interoperation with legacy systems and enterprise applications, inclusion of specialized protocols and standards, and support of robust safety mechanisms.

There is a need to integrate ACL with mechanisms for safe communications (cryptography, digital signature, certification, etc.) that have been developed for virtual enterprises and e-commerce (Camarinha-Matos, 2002). The development of advanced simulation tools to support planning, optimization, and assessment of operation of virtual enterprises and distributed business processes is another open challenge that can benefit from a MAS approach (Camarinha-Matos, 2002). Finally it is important to stress that in order to be accepted by the industrial community, MAS applications need to be successfully demonstrated in complex real world pilot systems (Camarinha-Matos, 2002).

CONCLUSION

The area of software agents is vibrant and rapidly developing. A number of fundamental advances have been made in the design and the implementation of software agents as well as in the interaction between software agents. In this brief chapter, we have tried to convey some of the key concepts of the active field of software agents and make a reference point to a large body of literature outlining essential issues. We were limited to enumerate our findings of our survey regarding software agent technology, instead of judging them, aiming to provide a synoptic review of the basic aspects. It is up to the reader to judge how successful we have been in meeting our goal in this chapter. In addition, we have argued the issue of applying the agent technology to virtual enterprise. Our purpose was to offer a brief introduction of the application of agent technology to virtual enterprises and to provide some useful hints for further studying concerning the above-mentioned theme.

REFERENCES

Agent-Oriented Software Engineering. *Research Area Examination* (2005). Retrieved from http://www.deg.byu.edu/proposals/ResearchAreaExamMuhammedJM.pdf

Alonso, E. (2002). AI and agents: State of the art. *AI Magazine, 23*(3), 25-29.

AOSE Technical Forum Group (2004). *AL3-TF1 Report*. Retrieved January 31, 2007, from http://www.pa.icar.cnr.it/~cossentino/al3tf1/docs/aose_tfg1_report.pdf

Bracho, A., Matteo, A., & Metzner, C. (1999). A taxonomy for comparing distributed objects technologies. *CLEI Electronic Journal, 2*(2).

Bradshow, J. M. (1997). An introduction to software agents. In J. M. Bradshow (Ed.), *Software agents*. Cambridge: MIT Press.

Bussink, D. (2004). A comparison of language evolution and communication protocols in multi-agent systems. In *Proceedings of the 1st Twente Student Conference on IT, Track C—Intelligent_Interaction*. Retrieved January 31, 2007, from http://referaat.ewi.utwente.nl/

Camarinha-Matos, L. M. (2002). Multi-agent systems in virtual enterprises. In *Proceedings of AIS2002—International Conference on AI, Simulation and Planning in High Autonomy Systems*, Lisbon, Portugal (pp. 27-36).

Camarinha-Matos, L. M., & Afsarmanesh, H. (1998). Cooperative systems challenges in virtual enterprises. In *Proceedings of CESA'98—IMCAS Multiconference on Computational Engineering in Systems Applications*, Nabeu—Hammamet, Tunisia.

Chira, C. (2003). *Software agents*. (IDIMS Report). Retrieved January 31, 2007, from http://pan.nuigalway.ie/code/docs/agents.pdf

Chrysanthis, P. K., Znati, T., Banerjee, S., & Shi-Kuo, C. (1999). Establishing virtual enterprises by means of mobile agents. In *Proceedings of the Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises* RIDE-VE '99 (pp.116-123).

Cernuzzi, L., Cossentino, M., & Zambonelli, F. (2004). *Process models for agent-based development*. http://www.pa.icar.cnr.it/~cossentino/paper/eaai_zambonelli_draft.pdf

Creative Science Systems (CSS). Retrieved from http://www.creativescience.com/Products/soa.shtml

DCOM Technical Overview, Microsoft Corporation. (1996). Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

Do, V., Halatchev, M., & Neumann, D. (2000). A context-based approach to support virtual enterprises. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*.

Etzioni, O., & Weld, D. S. (1995). Intelligent agents on the Internet: Fact, fiction and forecast. *IEEE Expert, 10*(4), 44-49.

Faculty Science Unitn. University of Trento. *AOSE Methologies*. (n.d.) Retrieved from http://www.science.unitn.it/~recla/aose/

Fankhauser, P., & Tesch, T. (1999). Agents, a broker, and lies. In *Proceedings of the Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises* RIDE-VE '99 (pp.56-63).

Fox, M. S., & Gruninger, M. (1998). Enterprise modelling. *AI Magazine*, 109-121.

Franklin, S., & Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*.

Freitas Mundim, A. P., Rossi, A., & Stocchetti, A. (2000). SME in global markets: Challenges, opportunities and threats. *Brasilian Electronic Journal of Economics, 3*(1).

Gijsen, J. W. J., Szirbik, N. B., & Wagner, G. (2002). Agent technologies for virtual enterprises in the one-of-a-kind-production industry. *International Journal of Electronic Commerce, 7*(1), 9-26.

Goodwin, R. (1993). *Formalizing properties of agents* (Tech. Rep.) Pittsburgh, PA: Carnegie-Mellon University, School of Computer Science.

Gong, B., & Wang, S. (2000). Model of MAS: Supporting dynamic enterprise model. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation* (pp. 2042-2046).

Gou, H., Huang, B., Liu, W., & Li, Y. (2001). Agent-based virtual enterprise modeling and

operation control. In *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 2058-2063).

Green, S., Hurst, L., Nangle, B., Cunningham, D. P., Somers, F., & Evans, D. R. (1997). *Software agents: A review* (Tech. Rep. No.TCS-CS-1997-06). Trinity College Dublin, Broadcom Éireann Research Ltd.

Hayes, C. C. (1999). Agents in a nutshell: A very brief introduction. *IEEE Transactions on Knowledge and Data Engineering, 11*(1), 127-132.

Horn, E., Kupries, M., & Reinke, T. (1999). Properties and models of software agents and prefabrication for agent application systems. In *Proceedings of the International Conference on System Sciences (HICSS-32), Software Technology Track*

Iglesias, C. A., Garijo, M., & Gonzalez, J. C. (1999). A survey of agent-oriented methologies. In *Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)* (pp. 317-330).

Java.sun.com. The Source for Java Developers. Retrieved from http://java.sun.com/products/jdk/rmi/

Jennings, N. R., Norman, T. J., & Faratin, P. (1998). ADEPT: An agent-based approach to business process management. *ACM SIGMOD Record, 27*(4), 32-39.

Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems Journal, *1*(1), 7-38.

Jennings, N., & Wooldridge, M. (1996). Software agents. *IEE Review*, 17-20.

Katz, B. R., & Schuh, G. (1999). The virtual enterprise. Retrieved January 31, 2007, from http://portal.cetim.org/file/1/68/KatzySchuh-1999-The_virtual_enterprise.pdf

Kostakos, V., & Taraschi, C. (2001). Agents. Retrieved January 31, 2007, from http://www.cs.bath.ac.uk/~vk/files/agents.pdf

Luck, M., McBurney, P., & Gonzalez-Palacios, J. (2006). *Agent-based computing and programming of agent systems* (LNCS 3862, pp. 23-37). Springer.

Marík, V., & McFarlane, D. C. (2005). Industrial adoption of agent-based technologies. *IEEE Intelligent Systems, 20*(1), 27-35

Nwana, H. S. (1996). Software agents: An overview. *Knowledge Engineering Review, 11*(3), 1-40.

Nwana, H., & Ndumu, D. (1996). A brief introduction to software agent technology. In *Proceedings of the Unicom Seminar on "Real-World Applications of Intelligent Agent Technology"* (pp. 278-292).

Nwana, H., & Wooldridge, M. (1996). Software agent technologies. *BT Technology Journal, 14*(4), 68-78.

Odell, J. (2000). Agents: Technology and usage (Part 1). *Executive Report, 3*(4).

Oliveira, E., Fischer, K., & Stepankova, O. (1999). Multi-agent systems: Which research for which applications. *Robotics and Autonomous Systems, 27*, 91-106.

OMG Agent Working Group (2000). *Agent technology.* Green paper, OMG Document ec/8/1/00, (1).

Ouzounis, V. K., & Tschammer, V. (2001). An agent-based life cycle management for dynamic virtual enterprises. In *Proceedings of the Sixth International Conference on Computer Supported Cooperative Work in Design* (pp. 451-459).

Petersen, S. A., Divitini, M., & Matskin, M. (2001). An agent-based approach to modeling virtual enterprises. *Production Planning & Control, 12*(3), 224-233.

Petersen, S. A., Jinghai, R., & Matskin, M. (2003). Virtual enterprise formation with agents: An approach to implementation. In *Proceedings of the IAT 2003, IEEE/VVIC International Conference on Intelligent Agent Technology* (pp. 527-530).

Putnik, G. D. (2004). Virtual Enterprise as a "flow" enterprise. In *Proceedings of the Fourth Annual Meeting of the European Chaos and Complexity in Organisations Network ECCON - ECCON 2005.*

Rahwan, I., Kowalczyk, R., & Yang, Y. (2001). Virtual enterprise design: BDI agents vs. objects. *Advances in artificial intelligence* (LNCS 2112, pp. 147-157).

Serenko, & Deltor, B. (2002). *Agent toolkits: A general overview of the market and an assessment of instructor satisfaction with utilizing in the classroom* (Working Paper No. 455). Retrieved January 31, 2007, from http://www.business.mcmaster.ca/msis/profs/detlorb/nserc/McMaster_Working_Paper_455.pdf

Sycara, K. P. (1998). Multi-agent systems. *AI Magazine, 19*(2), 79-92.

Sycara, K. P. (1998). The many faces of agents. *AI Magazine, 19*(2), 11-12.

Szirbik, N., Aerts, A., Wortmann, H., Hammer, D., & Goossenaerts, J. (2000). Mediating negotiations in a virtual enterprise via mobile agents.In *Proceedings of the Academia/Industry Working Conference on Research Challenges* (pp. 237-242).

The Object Management Group. (n.d.). *CORBA® BASICS*. Retrieved from http://www.omg.org/gettingstarted/corbafaq.htm

The Object Management Group. (n.d.). Retrieved from http://www.omg.org

The CORBA Object Group Service. *A Service Approach to Object Groups in CORBA, Nº 1867.* (1998). Retrieved from http://lsrwww.epfl.ch/OGS/thesis/

Tuukka Vartiainen, Java Beans, and Enterprise Java Beans. (n.d.). Retrieved from http://www.cs.helsinki.fi/u/campa/teaching/tukka-final.pdf

Weiß, G. (2002). Agent orientation in software engineering. *Knowledge Engineering Review, 16*(4), 349-373.

Wooldridge, M. (1998). Agent-based computing. *Interoperable Communication Networks, 1*(1), 71-97.

Wooldridge, M. (1999). Intelligent agents. In G. Weiss (Ed.), *Multi-agent systems.* MIT Press.

Wooldridge, M., & Ciancarini, P. (2000). Agent-oriented software engineering: The state of the art. In P. Ciancarini & M. Wooldridge (Eds.), *Proceedings of the First International Workshop on Agent-Oriented Software Engineering* (pp. 1-28).

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review, 10*(2), 115-152.

Yan, Z., Meilin, S., & Shaohua, Z. (2004). An agent-based framework for cross-domain cooperation of virtual enterprise. In *Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design* (pp. 291-296).

Yonghe, L., & Biqing, H. (1999). Virtual enterprise: An agent-based approach for decision and control. In *Proceedings of the IEEE SMC '99 International Conference on Systems, Man, and Cybernetics* (pp. 451-456).

Zambonelli, F., Jennings, N., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methology. *ACM Transactions on Software Engineering and Methodology, 12*(3).

This work was previously published in Agent and Web Service Technologies in Virtual Enterprises, edited by N. Protogeros, pp. 1-24, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 1.15
Automated Software Testing

Paula Donegan
Instituto Atlântico, Brazil

Liane Bandeira
Instituto Atlântico, Brazil

Cristina Matos
Instituto Atlântico, Brazil

Paula Luciana da Cunha
Instituto Atlântico, Brazil

Camila Maia
Instituto Atlântico, Brazil

ABSTRACT

This chapter approaches paramount aspects related to test automation, introducing the importance of implementation in the software market and essential bases, such as adjustment to the organizational reality and establishment of an efficient strategy. Types of tools and directives for a successful implantation are presented. Test automation has been considered the main measure taken to enhance test efficiency — fundamental in the software-development process. Responsible for verifying and/or validating the quality of the executable product compared to performed documentation and client requirements. Therefore, with the chapter content here provided, we aim to provide the reader with an understanding of test automation and grant relevant orientations to assist implementing it.

INTRODUCTION

Given the growing complexity of applications and new technologies, such as the advent of the client/server environment (in particular Web applications), the effort necessary for application testing has increased.

To assure that software conforms to requirements, various test stages may be identified: unit, integration, system, and acceptance. Bugs' impact increases with the evolution of the test stage in

which they are found, in other words, the cost of detecting errors during unit test is less than integration and system tests.

Each use case has test objects that may need to be retested several times during the project, demanding resources. These retests normally are required when a new functionality is added or when a bug is corrected, because there is no guarantee that the changes made will impact negatively on other parts already constructed. Therefore, the assistance of a tool capable of repeating a test already executed in the past is quite interesting.

Besides, multiple execution paths and diversity of possible inputs and outputs of an application complicate the execution of manual tests, which may be simplified by automation. In addition, performance, load and volume tests are examples of tests that are difficult to be accomplished without the help of automated testing tools. There are also some types of tests that are almost impossible to be executed manually, for example, a test to verify a system's performance with thousands or millions of simultaneous accesses or having to use an enormous amount of data.

Automating software tests speeds development and reduces retesting effort spent in each stage, thus reducing time and cost. However, this reduction is normally noticed only after a while, because there are high investments in the implantation stage, such as organizational needs, training, and tools acquisition. Automation allows increase of amplitude and depth of developed tests.

Testing automation might or might not be helpful. It allows one to take advantage of idle machine time (i.e., the period in which the developer is not working) to execute tests. Therefore, test execution can be more effective and waste less resources.

BACKGROUND

Automated software testing is an activity that seems to have obvious benefits: tests may be executed swiftly, are more consistent, and may be repeated various times without increasing cost. However, it is not a trivial activity and requires effective planning and elaborate test-case definition, as well as other characteristics, which will be explained in more detail later in this chapter. Benefits and risks, possible tools, an implantation methodology and directives for script generation are also described.

An automated test between different phases of the development process has the purpose of verifying if what was constructed from that stage backwards is correct and is adequate as an input for the next stage. An example would be a programmer testing a software component before doing the integration of components.

The generated test process is automated and capable of ensuring that the system logically operates according to the code by executing the functions through a series of test cases.

With a tool, you can expect the test script to conduct the verification processes and return results that verify whether the product under test meets code logic.

A test engineer usually follows a procedure to decide whether a problem found is a defect. However, an automated test tool makes decisions based on methods invocation, during which it detects errors and defects. Thus, the tool makes an effort to remind the developers of the importance of adopting a good error-handling technique.

But, can a tool verify that all test tasks have been performed? The answer is based on the requirements of your organization and on the architecture of the software project (Li & Wu, 2004).

Tests to validate a product against specified client requirements are more subjective and cannot always be automated. An example is a test executed by a user in a beta-program.

MANUAL TESTING VS. AUTOMATED TESTING

Software testing is necessary to guarantee the quality of a software product, may be performed during the whole software development process, in a manual, automated or hybrid manner, using different types of tests and tools.

Automated software testing simulates system behavior using tools. The test actions performed on the application are specified in code (scripts and test classes). In a context where required tests are not possible or viable to be executed manually, automated software testing is very important. As examples in which automated tests are indicated, one has Automated tests are used in regression, load, performance, and endurance tests and tests involving a vast amount of data.

Regression tests require testing all constructed functionalities after one of them is changed or a new one is inserted. Automation helps in this process, providing repetition of tasks already executed. Thus, functionality tests already recorded may be executed again, decreasing retest efforts.

For load tests, a high amount of users simultaneously accessing the application is necessary (e.g., in the order of 100; 1,000; and 10,000). How can such a test be performed? Automation is a resource that can easily simulate this scenario.

When using test performance, an automation test tool is used to capture time measurements each time they are executed. Performance degradations can be detected by collecting these measurements and reviewing them as time series (Karner, 1993).

Endurance testing requires automation. In this type of test, a specific application behavior is observed with the execution of its functionalities for a certain amount of time — weeks or months. This way, it is possible to detect problems like memory leaks, stack corruption and wild pointers.

Some specific tests — such as installation, configuration, and usability tests and specific hardware operations tests — require a strong human intervention, as well as a specific human evaluation and validation. Manual tests are recommended for these kinds of tests, being executed by at least one person, following a test procedure with application input and output data.

Test automation has a high cost. Training and scripts maintenance are necessary and, in case the tools used are not free, licenses are needed according to the number of users. However, automation has become essential given to systems' high complexity, need of performance and stress testing, resulting in increase of testing time and cost, reduction of software quality and, after the recognition of importance of software tests, pressure over project development teams has increased.

Test execution may also be performed with hybrid methods, characterized by applying jointly manual and automated techniques in the same project. The choice will depend on a thorough analysis of time and resources available and test complexity. The degree of maintenance and human intervention must be taken into consideration given to application modifications.

Many tests are not executed many times during a project life cycle. In these cases it is more advantageous to execute the tests manually, because efforts employed in their automation do not provide large returns. Therefore, it is not recommended to automate 100% of the tests.

Manual and automated tests should not be compared only in terms of time, effort, and cost, because the value of a test is especially in the

information it provides. In the same way, manual and automated tests are not allowed to be compared in many cases, because they provide distinctive information, as seen before, for every type of test, one of them is normally indicated.

However, in certain situations it might be a reasonable goal to create the most extensive automated test suite, such as contractual or regulatory reasons, when it is necessary to prove that the final version passed a strong battery of tests; when tests will be delivered with the product so that customers can also run the tests themselves; when the product has stringent requirements for backwards compatibility, remaining for many versions, or when the only way to test some products is by writing programs to exercise them (Kaner, Bach, & Pettichord, 2002).

OVERVIEW OF AUTOMATED TESTING

Nowadays automated testing is considered the main resource to improve the efficiency of a test process (Rios & Moreira, 2003), using tools so that the computer is responsible for assistance, design achievement, execution or tests control. Information collection and their quick dissemination are assisted by automated tests to provide a faster feedback to the development team (Kaner, 2002).

Tools to execute automated tests normally simulate the use of an application by several users simultaneously, a high load of data processing, as well as repetition of tasks previously executed. Besides, automating regression tests is of great relevance for the maintenance phase, since tests executed during development may be repeated in new tests, when the system is already in production.

When the automation does not apply to the entire test process, it can be used to execute punctual tests. This occurs normally with some types of tests — such as load and performance tests — because they are truly difficult to be performed without the help of a tool, requiring big effort and many computational resources.

The preparation of automated tests takes more time than manual tests. This is mainly a consequence of high effort and cost necessary to generate and maintain automation code. Therefore, so that the investment may bring satisfactory return on investment, automation must help to achieve the specific tests mission (Kaner, 2002), providing a supply of functional and nonfunctional tests.

Automation must be introduced in a context where the test process is well performed, with well-defined activities and with a mature and experienced test team. Otherwise, automation will not assist the achievement of test objectives. For this reason, test processes must be fixed before automating them.

Specific management treatment is required, in other words, planning, execution, and control, because normally automation is characterized by its innovation and high investments, complexity, and risks. Without this management, the expected benefits may not be achieved, besides consuming resources and dispersing the team from their test objectives, consequently interfering on the quality of tested products. Moreover, previous strategies and planning definitions are necessary, so that risks are controlled, possibility of scripts reuse is increased and automation optimized.

Automated testing tools do not replace testers, nor will their work be simpler. Actually their effort will be redirected to essential activities of the test automation process. For this reason, it is of great relevance to prepare the group, because the success of automated testing depends predominantly on their skills.

TEST-STAGES AUTOMATION

Test automation may be applied to diverse stages: unit, integration, and systemic. Each one has peculiar characteristics and is implemented in a

different way. Next, these aspects will be shown for each test stage.

Automated Unit Testing

Unit tests consist of testing components individually. The tester verifies the input domain only for the unit in question, ignoring the rest of the system. These tests require code construction executed during depuration mode.

The developer has the responsibility to ascertain that units are correctly constructed, analyzing code, comparing it with the system's specification and checking if any functionality is missing or was not completely developed.

Tests and error corrections may be made at any stage of the development cycle. Nevertheless, some authors like Boehm (1981), have shown that the cost of searching and finding errors increases while the development advances. It is cheaper to fix defects during the codification stage.

Procedure to implement unit testing:

1. Prepare test environment.
2. Define input domain based on requirements and use cases.
3. Define, for every input, expected output based on requirements and use cases.
4. Implement components to be tested.
5. Group unit tests in collections of components.
6. Implement unit tests.
7. Execute unit testing.
8. Fix component tested, if there is an error.
9. Execute step 8 while any error remains.

Unit testing consists basically of:

1. Variables initiation, including database population.
2. Business rules or input functions are applied.
3. Destruction of variables, including the cleaning up of data input to data base.
4. Comparison between results of applied function with expected results, failing in case they differ.

Grouping unit tests forms a tree, where the leaves consist of the unit tests and in the other nodes are the groupings of unit tests. This technique allows automating and executing all tests, a subset of them, or each one individually.

Unit test automation enables dynamic problem detection, generating automated reports, and facilitates execution of regression tests, which are necessary for every meaningful code update.

Automated Integrated Testing

Integrated tests are performed to test various parts of the system (components, modules, applications, etc.) that were separately developed in a set. Integration tests are executed after each of the system's parts has been tested individually, using unit tests or systemic tests, in case of applications that communicate (with each other).

Analyzing the time line, you can notice that integrated tests are mainly performed after a unit test and before systemic tests, normally are executed by developers that create their own builds and test integrates units or by a specialized testing team.

Procedure to implement integrated tests:

1. Prepare test environment, using test data and test server, which are configured to simulate the production environment.
2. Identify test cases based on requirements and architecture.
3. Detail procedures for each test case.
4. Implement integrated tests.
5. Execute integrated tests.
6. Analyze results. If errors are found, they must be registered in the problem reports tool and associated to responsibles for the corresponding corrections. If none are found codification may stop.

7. Fix problems encountered.
8. Execute tests again. After ending it, return to step 6.

Automated System Testing

System testing is the most misunderstood and most difficult testing process. System testing is not a process of testing the functions of the complete system or program, because this would be redundant with the process of function testing (Myers, 2004). System testing compares the system or program with its original objectives (requirements).

In this stage a test environment, compatible with the one in which the system or program will be used, is necessary.

Various test types are performed during this stage: usability, functionality, performance, stress, and so on. Some of these may be automated to make execution more agile, especially in case of regression tests.

Systemic tests may be automated in several ways, but a list follows with their basic activities:

1. Prepare test environment, installing and configuring necessary hardware and software.
2. Design test cases.
3. Define automation strategy.
4. Select project scope for automation.
5. Implement systemic test scripts for each test case.
6. Execute systemic test.
7. Analyze results. If errors are found, they must be registered in the problems report tool and associated to responsibles for the corresponding corrections.
8. Fix problems found.
9. Generate reports with tests status.
10. Execute tests again, returning to step 6 after finishing them.

Automation Techniques

Automated functional test tools make use of some techniques that differentiate themselves basically because of the contents of the generated scripts: record & playback, scripts programming, data-driven and keyword-driven. These techniques will be explained as follows:

- **Record and playback:** Technique that consists of recording a test execution made on the application's interface and playing back this execution later. The generated scripts contain unalterable data, test procedures, and expected results. The advantage of this technique is the simplicity of generating scripts. However, there is a high sensibility to changes, which restricts a script's lifetime. For example, a simple change of an interface may lead to the necessity of recording again all the scripts, implying a high script maintenance cost.
- **Scripts programming:** Considering the components of the generated scripts, this technique is similar to the one above. However, this one allows updating generated scripts. With this resource, script programming has a higher rate of reuse, longer lifetime, and less difficulties to be maintained, compared to the technique of Record & Playback. Nevertheless, there is still a big volume of scripts and high-maintenance cost (Fantinato, Cunha, Dias, Mizuno, & Cunha, 2004).
- **Data-driven:** This is a technique that approaches test-data extraction through scripts, storing them in separate files. Therefore, scripts will only contain test procedures and actions for the application. This technique is useful for tests using different inputs and combinations of scripts with common test procedures (Kaner, 2002). This way, maintenance requires a reduced effort when it is

necessary to include, update, or delete any test data. However, this technique depends on the test-execution logic, in other words, if any step is added to the procedure (or removed) the script needs to be generated again.

Nevertheless, it allows automating in parallel: test data can be created while test procedures are being generated. The extraction of test data from scripts provides tests that are easy to understand and revise. Many test tools include direct support for the technique *data-driven* (Kaner, 2002).

- **Keyword-driven:** Based on retrieving test procedures from scripts, remaining only test data and specific test actions, which are identified by keywords. This resource's operation, being invoked by keywords and even receiving parameters, is similar to that of structured-program functions. An important enhancement of this technique is the reduction of script-maintenance effort if there is incorporation, removal, or modification of any step of a test-procedure execution (Fantinato et al., 2004).

  In spite of tools existent in the market recommending a specific testing technique, the organization needs to establish the most adequate technique according to their own test context after analyzing all possibilities.

Good Practices for Automation

A list of relevant practices to improve systemic test-script implementation and reuse follows, which may be used according to the adopted technique:

- When recording a script for a test case, some necessary initialization may be necessary — such as database connection, variables, and functions — which can be shared between test cases of the same use case or even between many use cases.
- Each script may be recorded independently from the others, having its own variables, initialization, and so on, although it would be better to share common objects for test scripts, in other words, use the concept of modularity. For example, if it is necessary to clean up a database for a specific use case before executing any test script, or if there are common script steps, a text file to store these steps may be used and later on can be included at scripts execution time. If these steps change, only one place will have to be modified.
- Another practice is to define functions executing a common activity for many scripts, and only call it from those scripts.
- A clean and robust code must be implemented. Comments should be made for important lines or code blocks. While implementing, possible maintenance may not be discarded.
- When a flow alteration or verification point in a script is recorded again, there must be some care to update only script sections where there really must be a change.
- To delete any garbage shown in a form, before inserting any data, a combination of keys should be used ("HOME", "SHIFT+END" and "DELETE") when recording an action of data input.
- If test scripts are recorded initially from a prototype (in case the application is still not available), it is necessary to certify that names of fields and IDs of clickable icons on screen will not be altered. If they are changed, the script will need an update later.
- Dynamic data for test scripts should be used (test data separate of scripts). The immediate benefit is that, if additional data is needed, more records can simply be added; there is

no need to modify the test script (Mosley & Posey, 2003).

BENEFITS AND RISKS

The benefits of automated tests are innumerable. Some of them will be described here, as well as risks that were considered most important.

A growing number of software versions to be tested exhausts the capacity of manual tests in terms of time, patience, input variation, attributes combination and, as consequence, cost. With advent of increasingly complex, integrated, and critical software, automated tests have become a necessity.

Benefits of test automation may be observed in any test stage, such as unit tests, integrated tests, and systemic tests.

Unit testing is a way of efficiently granting an applications quality. In this stage, a big quantity of errors can be detected and easier corrected. The manual execution of unit tests is a difficult process that consumes time and resources, being practically unviable since almost all system units must be tested, depending on the context, such as functions, subroutines, components, or classes.

Benefits of automated unit tests are easily noticed. During the project's life cycle unit tests, automation can decrease the costs of rewriting code, because more errors are found still in the codification stage. This test stage also grants more security when doing updates, since the entire set of tests are executed automatically. It is easier for programmers to write test routines that retain their value in time and assure the correct functioning of units. Besides, automated unit testing provides test results immediately after their execution.

In *integrated tests*, where the focus is on systems architecture and integration between different units developed, it is also indispensable to execute them automatically. The integration between system units may occur daily during the development process and it is important to always test the relationships between those components using automatic mechanisms, like continuous integration.

Automation of *systemic tests* requires attention and planning, because in this stage the functionalities' behavior is verified according to specified requirements. System testing is conducted at a higher level. During systemic tests, an examination is made of integration of parts that make up the entire system (Dustin, Rashka, & Paul, 1999).

For each test stage, automation must take into consideration possible software changes. However, systemic test scripts have their execution recorded in an application interface, therefore, there must be a special care avoiding costs of rework. Flexible scripts (dependencies with interface are minimized) can assist scripts maintenance.

In general, test automation in any stage allows:

- **Test effort reduction:** Introduction of automated test tools will not immediately reduce the test effort. Experience has shown that a learning curve is associated with attempts to apply automated testing to a new project and to achieve effective use of automated testing.
  While the testing effort will likely increase at first, a playback on test tool investment will appear after the first iteration of the tools' implementation, due to improved productivity of the test team.
  Manual tests must still be performed on the project, while a learning curve may exist due to effort with familiarization and efficiency in the use of the tool (Dustin et al., 1999).
- **Schedule reduction:** As the testing effort may actually increase, the testing schedule will not experience an initial decrease but may instead become extended. Therefore, permission to increase the schedule is required when initially introducing an automated test tool. Once an automatic testing

process is established and effectively implemented, the project can expect to experience gains in productivity and turnaround time, having a positive effect on schedule and cost (Dustin et al.,1999).

- **Improved regression testing:** An automated test tool provides simplified regression testing. Automated regression testing can verify in an expedient manner that no new bugs were introduced into a new build (Dustin et al., 1999).
- **Improved focus on advanced test issues:** Automated testing allows simple repeatable tests. A significant amount of tests are conducted on the basic user-interface operations of an application.

  Besides delaying other tests, the tedium of these tests exacts a very high toll on manual testers. Manual testing can become stalled due to the repetition of these tests, at the expense of progress on other required tests. Automated testing presents the opportunity to move on more quickly and to perform a more comprehensive test within the schedule allowed. That is, automatic creation of user interface operability tests gets these tests out of the way rapidly and releases test resources, allowing test teams to turn their creativity and effort to more complicated problems and concerns.
- **Increased test coverage:** Automated testing may increase breadth and depth of test coverage, yet there will still not be enough time or resources to perform a 100% exhaustive test. Even with automation, testing every combination of a system exhaustively is impossible. Therefore, there must be strategies (equivalence partitioning) to select relevant test data.
- **Productivity increase:** Having increased test coverage, automation also increases productivity. Testers may verify more test cases in less time than when using manual tests.
- **Improved performance testing:** Many load-testing tools are available that allow one to automatically test the performance of a system's functionalities, producing timing numbers and graphs and thresholds of the system. There is no longer the necessity to sit with a stopwatch in hand. The objective of performance testing is to demonstrate that a system functions in accordance with its performance requirement specifications regarding acceptable response time, while processing the required transaction volumes on a production-size database (Dustin et al., 1999).

From these benefits, we can observe that automation is essential, because, even with an extremely experienced tester, it is almost impossible to know all the relations and combinations between attributes of medium- and large-sized applications.

Even so, the implementation of a solution for automated tests, even with excellent test tools available, is not trivial and can fail mainly due to:

- **Nonqualified test team:** An automated tool requires new skills for test analysts, therefore additional training is required. An efficient automation is not that simple. Test scripts automatically generated by the tool during recording must be modified manually, which requires scripting knowledge, so as to make the scripts robust, reusable, and maintainable. To be able to modify the scripts, the test engineer must be trained on the tool and the tool's built-in scripting language (Dustin et al., 1999).
- **Inadequate tools:** The automation tool must be in accordance to the company's business needs with the test process introduced. There are many tools that automate tests and a correct choice cooperates to fully use features offered. There is not a tool clearly

superior to other tools for every situation. The choice of the most adequate tool depends on characteristics of applications, test stages to be automated, as well as the adopted organizational test policy.

Simply using tools, by themselves, will not promote a significant quality enhancement of developed products, if not accompanied by an adoption of a work methodology.

An incorrect choice may lead to automation failure, once the organizational testing needs will not be completely suppressed and resources and efforts addressed to manual tests deviated. Besides, given to the affected credibility, it may be difficult to obtain further investments of higher management to proceed with automation or even to reinforce manual tests.

- **Elevated cost with systemic test scripts maintenance:** This aspect may not compensate automation costs. Systemic test scripts have to be flexible enough to easily support various changes and insertion of new functionalities that appear during a project's life cycle.

  To create scripts easy maintainable, professionals with experience in software development are important. They are supposed to be capable of designing a modular script structure, in the same way as when doing a normal software design.
- **Automation of every test type:** As explained in this chapter, when tests require high human intervention it is not recommended to focus on automated tests.
- **Lack of planning and control of the automation process:** Test automation must be considered a separate project from the software development process. Cautions related to project management have to be taken, such as concerns with scope, time, cost, quality, integration, resources, and risks.

Execution of automated tests helps to provide reliable and stable applications. However, to reduce associated costs and risks, it is important to analyze factors mentioned earlier on. Thus automation benefits can be more evident in the organization and risks can be controlled by mitigation and/or contingency.

AUTOMATED TEST TOOLS

The tools for automated testing are instruments to make the testing process easier, replacing and/or assisting manual tests performed by testers. Various tools exist to be used in the various test stages.

Many organizations have successfully chosen and purchased a test tool, but around half of those organizations have not achieved any benefit from their investment because their tools have ended up not being used (*shelfware*) (Fewster & Graham, 1999). A way of easing this risk is by knowing better the available market categories and types of automated tools.

Categories and Types

While GUI-testing tools (also called "capture/playback" tools or functional-testing tools) are much hyped in the testing industry, it is important to be aware of the other types of tools available to support the testing life cycle. This item provides an overview of the available tools, listed in Table 1, supporting the various test activities.

The existent divisions of types of test tools are not uniform. Therefore, to be more didactic and easier to understand the purpose of each test tool, they may be divided into categories, and those categories have many types of tools associated with them, according to their objectives:

- **Test-development tools:** Assist test plan elaboration and generation of test cases and

Table 1. Tools by category and type

Tool Category	Type of Tool	Description	Example
Test Development	Test-procedure generator	Generates test procedures from requirements/design/object models	▪ TYX — Test Procedure Generation Wizards
	Test-data generator	Generates test data	▪ Tnsgen ▪ DTM Data Generator
	Test-data extraction	Extraction and verification of test data	▪ Kapow Web Collector
Test Execution	Unit testing	Unit API testing and assertions verification	▪ PyUnit ▪ JUnit ▪ HttpUnit
	Memory-leak detection	Verify that an application is properly managing its memory resources	▪ JProbe ▪ Pruify
	GUI-testing (capture/playback)	Automate GUI tests by recording user interactions with systems, so they may be replayed automatically	▪ Robot ▪ WinRunner ▪ QARun ▪ Functional Tester
	Load, performance, and stress testing	Load/performance and stress testing	▪ Robot ▪ LoadRunner ▪ JMeter ▪ JProbe
	Network testing	Monitoring, measuring, testing, and diagnosing performance across entire network	▪ CSSCheck ▪ Bobby
Test Support	Test management	Provide such test-management functions as test-procedure documentation, storage, and traceability	▪ TestManager ▪ TestDirector ▪ QADirector
	Code-coverage analyzers and code instrumentors	Identify untested code and support dynamic testing	▪ Clover ▪ NoUnit ▪ Jtest
	Metrics reporting	Read source code and display metrics information, such as complexity of data flow, data structure, and control flow. Can provide metrics about code size in terms of numbers of modules, operands, operators, and lines of code.	▪ McCabe Visual Quality
	Usability measurement	User profiling, task analysis, prototyping, and user walkthroughs	▪ SUMI
	Defect tracking (bug tracking)	Manage data base with found defects	▪ ClearQuest ▪ Bugzilla ▪ Jira

input data. Test development is responsible for designing tests and test data, being the most important set of activities in a testing process. Tests must enclose all requirements, otherwise, they will be invalid.

- **Test execution tools:** Assist test case execution and results evaluation; include also the capture/programming of test scripts.
- **Test support tools:** Assists test-specific activities, not being considered particular for tests. Include tools to support revisions, inspections, walkthroughs, project management, as well as tools for defect tracking and database managing.

Following are some key points regarding some of the types of test tools, according to their categories.

Test-Development Tools

- **Test-procedure generators:** A requirements-management tool may be coupled with a specification-based test-procedure (case) generator. The requirements-management tool is used to capture requirements information, which is then processed by the test-procedure generator. The generator creates test procedures by statistical, algorithmic, or heuristic means. In statistical test-procedure generation, the tool chooses input structures and values in a statistically random distribution, or a distribution that matches the usage profile of the software under test (Dustin, 2002).

  Most often, test-procedure generators employ action-, data-, logic-, event-, and state-driven strategies. Each of these strategies is employed to probe for a different kind of software defect. When generating test procedures by heuristic- or failure-directed means, the tool uses information provided by the test engineer. Failures discovered frequently in the past by the test engineer are entered into the tool. The tool then becomes knowledge-based, using the knowledge of historical failures to generate test procedures.
- **Test-data generators:** Test-data generators aid the testing process by automatically generating test data. Many tools on the market support the generation of test data and populating databases. Test-data generators can quickly populate a database based on a set of rules, whether data is needed for functional testing, data-driven load testing, or performance and stress testing.
- **Test-data extraction tools:** Using production data for testing purposes increases the integrity of testing by allowing testing teams to establish test scenarios using real test cases rather than relying on fabricated testing environments.

  A test-data extraction tool minimizes time to create test data and maximizes integrity and usability of data.

Test-Execution Tools

- **Unit-testing tools:** This type of tool is used to program and execute tests in units of the developed application. The units are normally classes, methods, or flows. Generally a unit-testing tool supports only one development language.
- **Memory-leak detection tools:** These tools are used for a specific purpose: to verify that an application is properly using its memory resources. These tools ascertain whether an application is failing to release memory allocated to it, and provide runtime error detection. Since memory issues are involved in many program defects, including performance problems, it is worthwhile to test an application's memory usage frequently.
- **GUI-testing tools (capture/playback tools):** Many automated GUI testing tools are on the market. These tools usually in-

clude a record-and-playback feature, which allows the test engineer to create (record), modify, and run (playback) automated tests across many environments. Tools that record the GUI components at the user-interface control are most useful. The record activity captures the keystrokes entered by the test engineer, automatically creating a script in a high-level language in the background. This recording is a computer program, referred to as a test script. Using only the capture and playback features of such a tool uses only about one-tenth of its capacity, however. To get the best value from a capture/playback tool, engineers should take advantage of the tool's built-in scripting language.

To create a reusable and maintainable test procedure, the test engineer must modify the recorded script. The outcome of the script becomes the baseline test. The script can then be played back on a new software build to compare its results to the baseline. The results can be compared pixel-by-pixel, character-by-character, or property-by-property, depending on the test-comparison type, and the tool automatically pinpoints the difference between the expected and actual result (Dustin, 2002).

- **Load, performance, and stress testing tools:** Performance-testing tools allow the tester to examine the response time and load capabilities of a system or application. The tools can be programmed to run on a number of client machines simultaneously to measure a client-server system's response times when accessed by many users at once. Stress testing involves running the client machines in high-stress scenarios to determine whether and when they break.
- **Network-testing tools:** The popularity of applications operating in client/server or Web environments introduces new complexity to the testing effort. The test engineer no longer exercises a single, closed application operating on a single system, as in the past. Client-server architecture involves three separate components: the server, the client, and the network. Interplatform connectivity increases potential for errors. As a result, the testing process must cover the performance of the server and the network, the overall system performance, and functionality across the three components (Dustin, 2002). Many network test tools allow the test engineer to monitor, measure, test, and diagnose performance across an entire network.

Test-Support Tools

- **Test-management tools:** Test-management tools support the planning, management, and analysis of all aspects of the testing life cycle. Some test-management tools, such as IBM's Rational TestStudio, are integrated with requirement- and configuration-management and defect-tracking tools, in order to simplify the entire testing life cycle.
- **Code-coverage analyzers and code-instrumentors tools:** Measuring structural coverage enables the development and test teams to gain insight into the effectiveness of tests and test suites. They are able to quantify the design complexities, help produce the integration tests, and measure the number of integration tests that have not been executed. Some of them measure multiple levels of test coverage, including segment, branch, and conditional coverage.
- **Metrics reporting tools:** Tools responsible for reporting metrics analyze code complexity and identify application passages that offer a higher fault risk, reporting quality metrics. With these tools it is possible to know parts of code that have or have not been tested.

  The usage of metrics-reporting tools assures higher product quality, because it is possible to have metrics showing the quality of what

is being developed.

- **Usability-measurement tools:** Usability engineering is a discipline that includes user-interface design, ergonomic concerns, human factors, graphics design, and industrial and cognitive psychology. Usability testing is largely a manual process of determining the ease of use and other characteristics of a system's interface (Dustin, 2002). However, some automated tools can assist with this process, although they should never replace human verification of the interface.
- **Defect tracking (bug tracking):** Most of the existing commercial tools present a conjunction of tool types or even categories. It may be a big advantage to obtain these integrated tools. However in some cases not all functionalities are needed and the tools cost may be an inhibitor. Therefore, it is very important to analyze test tools' costs and benefits thoroughly.

  Some test tools are intrusive, because it may be necessary to insert special code into the application program so that the automated tool may work correctly, interacting with the testing tool. Development engineers may be reluctant to incorporate this extra code. They may fear it will cause the system to operate improperly or require complicated adjustments to make it work properly. To avoid such conflicts, test engineers should involve the development staff in selecting an automated tool and if the tool requires code additions (not all tools do), developers need to know that well in advance.

  Intrusive tools pose the risk that defects introduced by testing hooks (code inserted specifically to facilitate testing) and instrumentation could interfere with normal functioning of the system.

  Besides, as with all technologies, test tools can be unpredictable. For example, repositories may become corrupt; baselines may not be restored, or may not always behave as expected. Often, much time must be spent tracking down a problem or restoring a back-up of a corrupted repository. Test tools are also complex applications in themselves, so they may have defects that interfere with the testing effort.

Programming Languages of Automated Test Tools

It has already been mentioned that there are a lot of test tools available on the market, and testing teams have used the tools and achieved great success. These test automation tools also include writing (i.e., coding) test scripts, which are software programs (Mosley & Posey, 2003). As such, they have their own programming languages and/or language extensions that are required to accommodate software-testing events, such as Java (JUnit and JProbe) or Visual Basic; in one of the standard scripting languages such as Perl, CGI, or VB Script; or in the operating system's command procedure language (Unix).

The scripting language is usually embedded in a capture/playback tool that has an accompanying source code editor. Some popular test tools write test scripts in Visual Basic 6.0 and execute the script in Visual Studio IDE. There is also literature that introduces how to write test scripts in Visual Basic 6.0 by hand. Some other tools are written in Java, such as JUnit, and JProbe and there are tools to write test scripts for software products run on Unix, Linux, and other platforms.

Test Tools Development

The usage of an automated tool will not always be adequate or provide necessary benefits. Diversified factors may lead to the necessity of building a tool:

- **Application incompatibility:** The application may contain an element that will cause compatibility problems with any capture/

playback tool on the market. If a work-around cannot be created, it may not be possible to use the capture/playback tool. Even if a work-around can be created, it may not be worth the effort, being more beneficial to create a customized tool.

- **Operating system incompatibility:** If there is a market absence of a tool, compatible with the various operation systems in use, there is no other choice than to consider building a customized tool that will work in the specific environment under test.
- **Specialized testing needs:** In case the test tool cannot reach a critical/complex component, an enhancement to the tool may be developed.

In case the organization chooses to develop a testing tool, resources, cost and time need to be determined, as well as management permission. The tools development must be treated as development effort of the project that demanded the use of the tool (Fewster & Graham, 1999). Many books provide detailed explanations of how to proceed to develop a test tool inside an organization, such as Li and Wu (2004) and Fewster and Graham (1999).

IMPLANTATION OF TEST AUTOMATION

Many companies, certain of the advantages of test automation, want to define a process of test automation to enhance the software-testing process, but do not know how to do it. Next, some phases are described which may be used for the test-automation implantation.

Introduce Automated Test Ideas

In this stage, the expectations management related to test automation and explanation of its benefits are performed, introducing automated test ideas to the organization.

Some imagine that a lot of time is spent with automation and that it has high costs. Therefore it is important to study its applicability (Donegan et al., 2005), verifying management, financial and structural benefits and risks involved, because the advantages and disadvantages of automation are to be made clear for everyone. At this moment, those involved with the implantation, including internal investors, are supposed to be conscious that normally the return on investment will be obtained in a medium or long term, since test automation does not reduce testing time as soon as it is introduced. This occurs given to time spent with tools initial learning, adaptation to the new testing process, among other factors mentioned before.

Generally, introducing tests automation requires a lot of investment. That does not happen only with the acquisition of a tool, but also with professional preparation of stakeholders and with maintenance of existent hardware to support the one being installed. Those variables must be considered in the projects' budget.

In this stage, an open communication with high management is fundamental, being constantly informed of organizational risks already detected, as well as direct benefits of automation, so that they may support the entire automation process even being aware of the existent challenges.

It is important to form a team with specialized professionals to begin the implantation of tests automation. The team must have at least one representative with knowledge of the organizational culture. These professionals will also be responsible for conducting the next stage.

After taking into account all the factors mentioned, the organization might take the decision of beginning test automation, or may abort the idea, being aware of risks and benefits involved. In case the decision is positive, some of the initial automated test ideas above have to be detailed in

a tests automation plan, defining tasks that will be executed.

Choose Test Tool

This stage guides the choice and evaluation of necessary tools for test automation.

Initially, the requirements of the tool are researched, according to organizational necessities: available resources, architecture of developed applications, platforms used, automation tool language, type and stage of desired test, and others. If possible, requirements should be prioritized conforming to the level of importance, making it possible to focus on key requirements in the evaluation process.

According to the requirements of highest importance, available tools in the market are selected. In case there is not any tool accessible to provide the organizational needs, the possibility of developing a homemade tool must be taken into account, which was seen earlier on in the chapter.

However, generally, a single test tool will not fulfill all the automated testing requirements for an organization, unless that organization works only with one type of operating system and one type of application. Expectations must be managed. It must be made clear that currently there is not a single tool on the market that is compatible with all operating systems and programming languages. This is only one reason why several tools are required to test the various technologies.

After preselecting the tools, there must be an assurance that the tools will fulfill the requirements and other points, like usability and performance. A way of doing this is using demo versions or doing hands-on. As a consequence, one or more tools are selected, according to company needs.

Refine Testing Process

This stage sketches the refinement of the organizational test process, including necessary activities for automated tests.

As an aftermath of the test tools choice, the process of implantation of automation begins. Now it is necessary to lay out objectives and strategies, by means of common architectures used in organizational projects. The strategy for automation in a company must contemplate directives that increase codification use and decrease their maintenance. See the "Good Practices" section.

After establishing these aspects, automation activities are defined. These activities are incorporated into the organizational testing process and stakeholders need to be properly trained. The mainly affected stakeholders are investigated, and a definition of the course goals (such as time and scope) is made.

A good way to consolidate the test process and chosen tools is to define a project to be automated and tested, called pilot project, applying the new test process and tools. It is necessary to monitor process planning, test-case elaboration, code creation, and tests execution and, especially, the results must be evaluated (Molinari, 2003).

All those involved in the test automation must participate, so that pertinent directives to the organizational context may be outlined and so that their commitment with the "Automation Product" is assured.

Develop Test Automation

Test Automation Planning

Conducting a common process of a Test Plan and designing test automation characterize the planning stage.

For the Test Plan, the scope of automation is defined and involved risks of the project in question are investigated, analyzed, and qualified. Besides, a test-automation strategy must be defined to orientate execution and monitoring, based on factors such as test scope, reuse of code, maintenance effort, and availability of time and resources.

Conventions of suites and code nomination may be established, as well as logical grouping relevant to worked scope. The most complex groups have to be identified and described in detail.

Test cases must have a well-defined design as they provide relevant information about the system to be tested without considering if tests are manual or automated. These test cases have to be analyzed, in order to identify the best automation possibilities. At this point, the test analyst must have a good understanding of how to raise valuable automation opportunities, in other words, opportunities that bring great aggregate value.

In case of automation of functional tests, another step consists of planning and designing useful test data, identifying the best way to communicate with code. See "Automation Techniques" section.

Test Automation Execution

Test execution approaches installation and configuration of tool and environment to initiate tests. This environment must also consider performance tests, which use various stations or virtual users. All necessary information concerning the environment must be contained in the test plan.

The preparation of a manual with instructions concerning chosen tools, test design information and adopted conventions is relevant.

The test team must develop test code based on the premises established during the planning stage and on the test projects already developed. The codes must be simple, robust, maintainable and, if possible, reusable.

After the test codes have been developed, unit, integrated and systemic tests can be performed. Generally, a different team of the one that executes systemic tests executes unit and integrated tests.

Each test team must report detected errors during test execution (integrated and systemic tests). A tool to report bugs facilitates this step.

Test Automation Management

Now that the tests are already planned and executed, they need to be managed. Control is essential for the test automation evolution analysis, with the perspective of analyzing if the new form of performing tests is acceptable and adaptable. Automation management is also fundamental to identify some points adopted during the planning stage that need to be adjusted, or even to reevaluate the tool with the purpose of certifying usability in the organization's context.

Tests codification must be constantly monitored, so that it can be a reflex of the application in such a way that modifications made in the product during its life cycle can also be contemplated in test code. This activity prevents reporting errors to the development team, caused by failure of test codification.

For systemic tests automation, when errors are found, it is necessary to do an analysis of them before sending the report to the development team, filtering duplicities and fake bugs, which are bugs caused by errors made during codification of tests.

Evaluate and Improve Test Automation

Automated test activities evaluation must be made during the whole test life cycle, so that improvements may be implanted. These enhancements can be observed along with execution of automated tests in organizational projects. Many of them are incorporated punctually into projects. The

company is, therefore, responsible for collecting, evaluating, and incorporating the improvements to the organizational process, when it is the case. Afterwards, stakeholders must be informed of changes made to the process and, if necessary, be trained.

Another important point is the test tool performance evaluation and determining what can be done to enhance it in the next project or test cycle.

RELATED WORKS

There are many works related to test automation: reports and case studies of organizations that have implemented automated software tests; research projects published in diverse events and magazines, creating models for test automation, elaborating efficient techniques for different types of automated tests, as well as others.

Organizations that develop software are constantly automating their tests to improve process and product quality, there are many examples to be found nowadays, here are two specific organizations using automated tests: Instituto Atlântico and BMC Software.

Instituto Atlântico implanted automation of functional system tests initially in a pilot project and, nowadays, automated tests are being used in other projects of the organization. Time spent to develop automated use case tests sometimes is superior to the time necessary to test it manually. However, Instituto Atlântico uses regression tests and this economizes time to execute tests, recovering the amount of time already used with codification after some increments. The tools used are those from Rational IBM: Robot, Test Manager and, more recently, Functional Tester. More details can be read in Donegan et. al (2005).

BMC Software developed a system to automate tests of a products suite called BMC's MetaSuite — family of applications client/server. They formulated orientations to choose people to automate tests and what should be automated.

BMC uses the tool QA Partner. One of the main gains using this tool is the easy maintenance of test scripts. Test cases are implemented before the interface is completely ready, because the test cases are independent of implementation details. In a week, BMC executed around three times each test case of three suite products. And those tests could be amply repeated in a more consistent manner. More information can be found in Pettichord (2001).

There are some interesting publications following, related to test automation, providing new approaches to automation. They may be adopted by organizations, if they are in conformity with objectives and purposes of the organization or specific project using automated tests.

Interface-Driven Model-Based Test Automation

The paper describes an interface-driven approach that combines requirement modeling to support automated test case and test driver generation. It focuses on how test engineers can develop more reusable models by clarifying textual requirements as models in terms of component or system interfaces. The focus of interface-driven modeling has been recognized as an improvement over the process of requirement-driven model-based test automation (Blackburn, Chandramouli, Busser, & Nauman, 2002).

Model-Based Approach to Security Test Automation

The paper summarizes the results of applying a model-based approach to automate security functional testing. The approach involves developing models of security function specifications (SFS) as the basis for automatic test-vector and test-driver generation (Blackburn, Busser, & Nauman, 2002).

FURTHER READING

A very good place to do some further reading is in the Web site http://www.stickyminds.com. You can find every kind of papers related to software tests, including some very interesting related to automated software testing

In terms of tools to automate tests, if you are interested in more examples of existent tools, we advise you to access Brian Marick's website http://www.testingfaqs.org (Marick, 2005). He presents an extensive list of tools for some types of automated tests. To learn more about the development of a tool to automate tests, it is interesting to read "Effective Software Test Automation — Developing an Automated Software Testing Tool" (Li, 2004), a book that teaches how to build a fully automated testing tool and provides expert guidance on deploying it in ways reaching high benefits.

In the section "Implantation of Test Automation" we showed a brief methodology to automate tests in an organization, however if you desire some further readings, it is interesting to read about the automate testing life-cycle methodology (ATLM) by Dustin, Rashka, and Paul (1999).

CONCLUSION

Good planning, clear definition of roles of those involved with automated tests, adequate tools and implantation with strong management to control risks and costs are essential so that the automated tests are successful in an organization.

Risks and benefits involved with automation need to be researched and afterwards informed to all stakeholders involved, especially to the high management, so that there is an effective compromise. Nevertheless, results obtained during test automation also have to be disclosed.

Automated tests are capable of increasing test coverage; because of time gain, cost subsided by future problems, and increase of the application's stability and investors trust.

A methodology for implantation of test automation was explained in this chapter, suggesting test automation to be implanted in stages, beginning with the decision of automation until process improvement, facilitating an efficient management and control of the automation process. An important stage for test automation is the choice of tools to be used. To make the right choice it is important to know the organization's real needs, testing-process maturity, test-team experience, automation techniques, and advantages and disadvantages of available commercial tools.

When initiating test automation it is important to customize the organizational test process according to required activities necessary for automated tests, training professionals, preparing environment, among other aspects given along the chapter. Thus it is possible to perceive that test automation must be treated as a project, requiring a plan, activities execution, and results monitoring. Therefore, it is more likely to obtain a positive return for the organization.

REFERENCES

Blackburn, M., Busser, R., & Nauman, A. (2002). *Interface-driven model-based test automation — Starwest 2002.* Retrieved September 2, 2005, from http://www.software.org/pub/externalpapers/starwest_2002.pdf

Blackburn, M., Chandramouli, R., Busser, R., & Nauman, A. (2002). *Interface-driven model-based test automation — Quality Week 2002.* Retrieved September 2, 2005, from http://www.software.org/pub/externalpapers/blackburn_issre_2002.pdf

Boehm, B. (1981). *Software engineering economics.* Englewood Cliffs, NJ: Prentice Hall.

Donegan, P., Bandeira, L., Matos, A., Cunha, P., Maia, C., & Pires, G. (2005) Aplicabilidade da automação de testes funcionais — a experiência no instituto atlântico. In *Simpósio brasileiro de qualidade de software* (pp. 447-454). Porto Alegre: Pontifícia Universidade Católica do Rio Grande do Sul.

Dustin, E. (2002). *Effective software testing: 50 specific ways to improve your testing.* Boston: Pearson Education.

Dustin, E., Rashka, J., & Paul, J. (1999). *Automated software testing: Introduction, management, and performance.* Boston: Addison Wesley.

Fantinato, M., Cunha, A., Dias, S., Mizuno, S., & Cunha, C. (2004). AutoTest — Um framework reutilizável para a automação de teste funcional de software. In Anais SBQS — *Simpósio brasileiro de qualidade de software* (pp. 286-300). Brasília, Brazil: Universidade católica de Brasília.

Fewster, M., & Graham, D. (1999). *Software test automation: Effective use of test execution tools.* Boston: Addison Wesley.

Kaner, C., Bach, J., & Pettichord, B. (2002). *lessons learned in software testing: A context-driven approach.* New York: John Wiley & Sons.

Karner, G. (1993). *Metrics for objectory.* Unpublished diploma thesis, University of Linköping, Sweden.

Li, K., & Wu, M. (2004). *Effective software test automation: Developing an Automated software testing tool.* Alamdea, CA: Sybex.

Marick, B. (2005). *Testingfaqs.org — An information resource for software testers.* Retrieved September 2, 2005, from http://testingfaqs.org

McGraw, G., & Michael, C. (1996, July). Automatic generation of test-cases for software testing. In CogSci 1996, *Proceedings of the 18th Annual Conference of the Cognitive Science Society* (pp. 370-375). Mahwah, NJ: Lawrence Erlbaum.

Molinari, L. (2003). *Testes de software — produzindo sistemas melhores e mais confiáveis.* São Paulo, Brazil: Ed. Erica.

Mosley, D., & Posey, B. (2003). *Just enough software test automation.* New York: Yourdon Press Series, Prentice Hall.

Myers, G. (2004). *The art of software testing.* New York: John Wiley & Sons.

Pettichord, B. (2001). *Success with test automation.* Revised version of a Quality Week paper. San Francisco. Retrieved September 02, 2005, from http://www.io.com/~wazmo/succpap.htm

Rios, E., & Moreira, T. (2003). *Projeto & engenharia de software — testes de software.* Rio de Janeiro, Brazil: Atlas Book.

This work was previously published in Verification, Validation and Testing in Software Engineering, edited by A. Dasso, pp. 82-110, copyright 2007 by Idea Group Publishing (an imprint of IGI Global).

Chapter 1.16
Software Metrics and Measurements

Michalis Xenos
Hellenic Open University, Greece

INTRODUCTION

In the past few years, a large number of e-government and e-commerce systems have been developed, thus resulting to a constantly increasing number of software developers involved in software development for such systems. To ensure the production of high quality e-government and e-commerce systems, it is important for developers to collect and analyze measurable data that guide estimation, decision making, and assessment. It is common sense that one can control and manage better what he is able to measure.

Although there are major differences between e-commerce and e-government (e.g., access, structure and accountability; Jorgenson & Cable, 2002) there are no significant differences in terms of software metrics that can be applied to both. Metrics are used in e-government and e-commerce software development to measure various factors related to software quality and can be classified as product metrics, process metrics and recourse metrics. *Product metrics* are also called software metrics. These are metrics that are directly related to the product itself, such as code statements, delivered executables, manuals, and strive to measure product quality, or attributes of the product that can be related to product quality. *Process metrics* focus on the process of software development and measure process characteristics, aiming to detect problems or to push forward successful practices. *Resource metrics* are related to the resources required for software development and their performance.

This article focuses on product metrics and on how such metrics can aid in design, prediction and assessment of the final product quality, provide data used for decision making, cost and effort estimation, fault prevention, testing time reduction, and, consequently, aid in producing better software for e-government and e-commerce systems.

BACKGROUND

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world so as to describe such entities according to clearly defined rules (Fenton & Pfleeger, 2004). In software development, measurements are conducted by using metrics. A *metric* is an empirical assignment of a value to an entity aiming to describe a specific characteristic of this entity. Measurements have been introduced into the e-government and e-commerce software development process in order to satisfy the need to control software development and produce higher quality results.

Since the mid 1970s when the first software metrics were proposed, a large number of metrics have been proposed in the following years. The proliferation of metrics was followed by more practical proposals on how to interpret results from metrics (see Shepperd & Ince, 1990) and methods combining metrics into measurement methodologies (see Xenos, 2003).

Public or private entities involved in software development for e-gsovernment and e-commerce applications can select from a variety of applied metrics those that are more suitable to be included in the development process (e.g., see Goodman, 2004; Kan, 2003). Therefore, taking into account the volume of literature that exists about software metrics, it is no more a question of finding metrics for an e-government or e-commerce project, rather than selecting the appropriate ones and extensively training engineering teams to utilize them (Hirsh, 2005). Given the large number of metrics (measuring almost everything), any attempt to select a metric without basing the selection on a detailed breakdown of the development needs and an extensive investigation of the proposed metric's applicability would result in minor benefits from its use or no benefits at all. To benefit from the use of metrics, apart from fully understanding the various existing metrics, one should also define well *why* he wants to measure, *what* to measure and *when* is the right time to measure it.

So the first question is: *Why use metrics?* The answer to this question is that metrics are needed to provide understanding of different elements of e-government and e-commerce software projects. Because it is not always clear what causes a project to fail, it is essential to measure and record characteristics of good projects as well as bad ones. Metrics provide indicators for the developed software. As Ragland (1995) stated, indicators are metrics or combinations of metrics that provide insights of the software development process, the software project, or the product itself. Measurements aim at the assessment of the status of the development process and the developed product. Therefore, metrics can be used for performance evaluation, cost estimation as Stamelos and Angelis (2001) have proposed, effort estimation, improving productivity, selecting best practices and—in general—for improving the quality of e-government and e-commerce systems.

This discussion leads to the next question: *What to measure?* As previously mentioned, process and product are what we need to measure. One may argue that, since the result of e-government and e-commerce projects is software, what we need to measure is only software. This is not true. According to Deming (1986), if the product you have developed is erroneous, do not just fix the errors, but also fix the process that allowed the errors into the product. This way you will not have to keep fixing the error in subsequent productions. Therefore, both process and product metrics and measurements are important in e-government and e-commerce software development.

It must be noted that, before selecting the appropriate metrics, it is very important to define the desired product quality characteristics. The selection of quality characteristics aids in defining what needs to be measured and what needs not, depending on the particular needs of the e-government and e-commerce application. In the early 1970s,

McCall, Richards, and Walters (1977) defined a framework for measuring such characteristics and proposed the Factors Criteria Metrics model—also known as FCM model—defining what is software quality in terms of subcharacteristics. Incorporating FCM and experience from similar proposals, years later, the ISO standard ISO/IEC 9126 (2001) standardized what product quality is in terms of subcharacteristics. Therefore the definition of product quality is important, as product metrics are used in the software development procedure to measure those product characteristics that are related to product quality.

Having defined the goals and reasons for measuring, the next question is: *When to measure?* Although measurements should be conducted throughout the entire e-government and e-commerce software development life cycle, their scope varies depending on the development phase. Different measurement goals are defined at different development phases, thus resulting into different kinds of metrics. In the early phases of e-government and e-commerce software development, metrics are used mainly for estimation purposes. It is useful to collect metrics relating to different projects as these can serve as historical data for future projects, aiding in better results.

In the intermediate phases of the e-government and e-commerce development process, metrics are used for project monitoring purposes and, in the meantime, code metrics are used to prevent errors. Furthermore, defect reports during testing are used for evaluating product quality and calibrating the measurement methods of the early phases. This purpose is also served by collecting external measurement data following project delivery, namely during the beta testing or maintenance phases of an e-government or e-commerce project. So the time to measure is determined by the requirements and the aims of the measurement program and can vary from a project to another.

Summarizing, using an oversimplifying statement, it could be said that metrics are an important instrument for the development of software to be integrated into e-government and e-commerce systems; metrics aid in making estimations in the early phases of a project, preventing problems in intermediate phases and evaluating quality in the late project phases.

USING METRICS IN SOFTWARE DEVELOPMENT FOR E-GOVERNMENT AND E-COMMERCE SYSTEMS

This section classifies product metrics in two categories—internal and external—provides a short definition and examples of each category, and discusses their advantages and disadvantages. The section concludes by presenting how these metrics can be combined and used in software development for e-government and e-commerce systems.

Product metrics can be categorized (Fenton & Pfleeger, 2004) as internal product metrics and external product metrics. *Internal product metrics* are those used to measure attributes of the product that can be measured directly by examining the product on its own irrespectively of its behavior. *External product metrics* are those used to measure attributes of the product that can be measured only with respect to how the product relates to its environment.

Internal Metrics

Internal metrics can be classified in three categories based on the product attributes they measure. These categories are size, complexity, and data metrics. As far as internal product metrics in general are concerned, it is important to mention that one of their major *advantages* is that they are easy to automate and therefore data collection can be made in an easy, automated, and cost-effective way. Furthermore, the measurement results can also be analyzed in an automated way using statis-

tical techniques and thus conclusions can be drawn rapidly. Tools such as QSUP (Xenos, Thanos, & Christodoulakis, 1996), Emerald (Hudepohl et al., 1996), GQM automation (Lavazza, 2000), and so forth have rendered internal measurements very easy to conduct. The screenshot from the metrics results of QSUP shown in Figure 1 is an example of the simple and automated way in which such measurements can be conducted. For further examples regarding metrics application, see Xenos (2003).

On the other hand, it should be mentioned that a major among the *disadvantages* of internal product measurements is the fact that they are often difficult to interpret. In other cases, the internal quantities measured are not clearly related to the external quality characteristics that one wants to assess. Such problems can only be solved in the framework of a well-defined measurement method that combines internal and external metrics, as will be discussed next.

External Metrics

Based on the ISO/IEC 9126 (2001) standard and on similar works such as Jung and Kim (2004), the *external factors* affecting software quality are Functionality, Usability, Efficiency and Reliability. For their definitions see "Key Terms," as defined by Kitchenham and Pfleeger (1996).

Figure 1. The results presentation window from QSUP Internal metrics

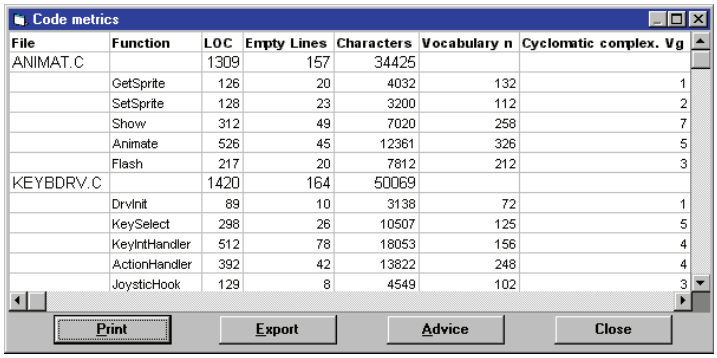


Table 1. High-level characteristics of e-commerce systems

Characteristics of e-commerce systems	Related quality factors
Easy access to the Web pages of the e-commerce system	Functionality, Usability, Efficiency
Easy navigation	Functionality, Usability
Adaptation to user profile	Functionality, Usability, Efficiency
Search engine service	Functionality, Usability, Reliability
Easy exit—undo functions	Functionality
Useful help service	Functionality, Usability, Efficiency
Electronic shopping cart	Functionality, Usability
Electronic shopping list	Functionality, Usability
Secure and reliable transactions	Functionality, Reliability
Security protocols SET, SSL	Reliability
Correct and accurate information about the products	Reliability
Direct delivery of the products	Usability, Efficiency
Indisputable financial transactions	Reliability
Recoverability of products and services	Usability, Functionality
Legitimate Web site	Reliability

External metrics are used to measure directly these four factors or the characteristics of which these factors are composed. For example, a set of high-level quality characteristics of e-commerce systems is presented in Table 1 (Stefani & Xenos 2001). This is important for the distinction between generic metrics and metrics defined especially for e-commerce systems.

Unlike internal metrics (measuring software internal characteristics and aiming at relating measurements of such characteristics to these factors), external metrics measure directly these factors or their characteristics. Such metrics can be based on subjective estimates. Among the means employed by external metrics are surveys on user opinion providing valuable measurements for software functionality or usability. Measures like defect reports or mean time between failures are used to determine product reliability, whereas measures like memory usage are used to determine efficiency.

As already mentioned, the application of external metrics implies that a certain extent of subjectivity is involved; even metrics that appear to be objective are often characterized by some degree of subjectivity. For example, defect reports seem to be a solid metric that can be used to objectively measure reliability. But the number of defect reports submitted by a user is influenced by issues such as the time and the extent of product usage, the user experience and even the user's motivation to edit and submit a defect report. Therefore, such metrics must be analyzed very carefully and under a framework that will take under consideration such issues.

One of the major *advantages* of external metrics is that they measure directly the desired external product quality characteristics, thus no further analysis or interpretation is needed. Additionally, external metrics contribute to a great extent to what is considered to be one of the main goals of e-government and e-commerce quality: user satisfaction. On the other hand, *disadvantages* and problems should be seriously taken under consideration when deciding to use external metrics, the most important of which being that such metrics are not objective and as a result additional effort is required to ensure their objectivity. Furthermore, they are not as cost effective as internal measurements and in many cases it is difficult to conduct measurements due to high error rates especially in cases that error detection techniques have not been used during measurements.

Combining Internal and External Metrics into a Measurement Method

As already mentioned, internal and external measurements must be conducted under a well-defined framework with precise goals. Before selecting the appropriate metrics for any project, a *quality manual* should be established collecting and documenting all metrics available for use in the software developing entity. This manual is a basic component of the metrics application process and includes the metrics, the measurement techniques as well as the guidelines for the application of metrics, the data analysis and the corrective actions required for improving the developing process of e-government and e-commerce systems. It should also be mentioned that the quality manual includes all metrics that are available regardless of how many times they have been used, or the availability of measurements data from past software development projects.

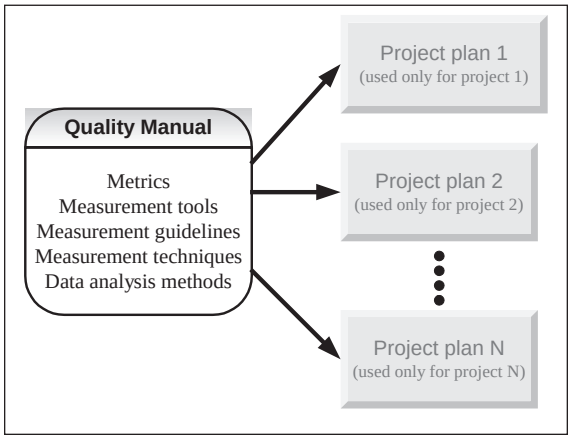
Then, for each e-government or e-commerce project, a set of metrics appropriate for this particular project is selected from the quality manual. The criteria on which the selection of metrics is based are the particular quality factors that the project places emphasis on. This set of metrics is documented—using the guidelines available in the quality manual—and consists the *quality plan* of the particular project. Thus, an e-government or e-commerce project quality plan should include all the metrics, measurement guidelines and goals applicable for the project. It is self-

evident that the project plan of a specific project may be entirely different from another project's plan and may use a completely different set of metrics. Figure 2 presents an illustration of the above procedure.

The quality plan of each e-government or e-commerce project should include internal metrics so as to provide an easy and inexpensive way to detect possible causes for low product quality, as this might be perceived by the end-users, and take early corrective action. It should also include external metrics—applied during alpha or beta testing and post shipment—so as to measure external quality factors, as well as the soundness of the internal metrics and measurements results or even calibrate internal metrics.

It should be noted that the successful selection of metrics and measurement techniques to be included in an entity's quality manual is heavily dependent on the entity's maturity. The adoption of sophisticated techniques and complex metrics by a company may prove to be ineffective, if it is not supported by years of experience with metrics and measurements and large volumes of data from past project measurements. Software developing companies should always keep this fact in mind and set feasible measurement goals not aiming too high at the early stages of metrics application.

Figure 2. Selection of metrics for each project



FUTURE TRENDS

For about 3 decades now, metrics have been used for the estimation of product related issues (such as product size, required effort, time required for testing, etc.) for early detection and prevention of problems during development and for product assessment after product release. Although in both cases metrics have proved to be successful in practice and have aided significantly towards developing higher quality e-government and e-commerce applications, the benefits from the use of metrics are not commonly recognized. This is partly due to the lack of awareness of metrics in small- and medium-size software developing companies. Although metrics are extensively used in large companies, in many cases, small- and medium-size enterprises are not even aware of the prospect and benefits of using metrics. However, this is constantly changing. More and more small- and medium-size e-government and e-commerce software-developing companies become aware of product metrics and measurements. Besides, the adoption of standards such as the ISO, or assessment in CMM higher levels, has contributed to this change since both standards are encouraging the use of metrics.

Another issue that is expected to change in the near future is the availability of more sophisticated tools. Although many measurement tools are available, using a number of metrics, there are not many tools available yet combining past projects' measurement data with current project data in order to aid in decision making. Combining metrics with decision support techniques, or methods for resolving uncertainty will lead to the development of valuable tools, which can aid towards higher quality software for e-government and e-commerce systems. A recent approach towards this direction (Fenton, Krause, & Neil,

2002) is using metrics and Bayesian networks for controlling software development, by automatically predicting defects in the released product.

CONCLUSION

This article introduced the reader to software metrics that are used to provide insight about different elements of e-government or e-commerce systems software. It presented internal metrics that can be applied prior to the release of the product to provide indications relating to quality characteristics, and external metrics applied after product delivery to give information about user perception of product quality.

Software metrics can be used to measure various factors related to software product development. These factors include estimation, early detection and prevention of problems, product assessment, etc. Their utilization within a measurements framework in combination to the use of automated tools can aid towards development process control and higher quality software for e-government and e-commerce systems.

Our focus was placed on the particular factors affecting the quality of e-government or e-commerce software. Such software can be measured effectively using a combination of generic internal software metrics and external metrics. The former are appropriate for most types of software, whereas the latter are designed especially for e-government or e-commerce systems.

REFERENCES

Deming, W. (1986). *Out of the crisis.* Cambridge, MA: MIT Center for Advanced Engineering Study.

Fenton, N., Krause, P., & Neil, M. (2002). Software measurement: Uncertainty and causal modeling. *IEEE Software, 19*(4), 116-122.

Fenton, N. E., & Pfleeger, S. L. (1997). *Software metrics: A rigorous & practical approach.* Boston: PWS Publishing Co.

Goodman P. (2004) *Software metrics: Best practices for successful IT management.* CT: Rothstein Associates Inc.

Hirsh B. (2005). Positive reinforcement as a quality tool. *IEEE Software, 22*(2), 62-63.

Hudepohl, J. P., Aud, S. J., Khoshgoftaar, T. M., Allen, E. B., & Maykand, J. (1996). Emerald: Software metrics and models on the desktop. *IEEE Software, 13*(5), 56-60.

ISO/IEC 9126. (2001). *Software product evaluation—Quality characteristics and guidelines for the user.* Geneva, Switzerland: International Organization for Standardization.

Jorgenson D., & Cable, S. (2002). Facing the challenges of e-government: A case study of the city of Corpus Christi, Texas. *SAM Advanced Management Journal, 67*(3), 15-21.

Jung, H., & Kim, S. (2004). Measuring software product quality: A survey of ISO/IEC 9126. *IEEE Software, 21*(5), 88-92.

Kan, S. H. (2003). *Metrics and models in software quality engineering* (2nd ed.). Boston: Addison-Wesley.

Kitchenham, B., & Pfleeger, S. (1996). Software quality: The elusive target. *IEEE Software, 13*(1), 12-21.

Lavazza, L. (2000). Providing automated support for the GQM measurement process. *IEEE Software, 17*(3), 56-62.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in software quality* (Vols. I, II, III). US Rome Air Development Center Reports (NTIS No. AD/A-049, 14-55), Sunnyvale, California.

Ragland, B. (1995, March). Measure, metric or indicator: What's the difference? *Crosstalk, 8*, 11-21.

Shepperd, M., & Ince, D. (1990). The use of metrics in the early detection of design errors. *Proceedings of Software Engineering,* Washington, DC, February 20-22 (pp. 76-81).

Stamelos, I., & Angelis, L. (2001). Managing uncertainty in project portfolio cost estimation. *Information & Software Technology, 43*(13), 759-768.

Stefani, A., & Xenos, M. (2001). A model for assessing the quality of e-commerce systems. *Proceedings of the PC-HCI 2001 Conference on Human Computer Interaction. Patras* (pp. 105-109).

Xenos, M. (2003). Technical issues related to IT governance tactics: Product metrics, measurements and process control. In W. Grembergen (Ed.), *Strategies for information technology governance* (pp. 216-244). Hershey, PA: Idea Group.

Xenos, M., Thanos, P., & Christodoulakis, D. (1996). QSUP: A supporting environment for preserving quality standards. *Proceedings of the Sixth International Conference on Software Quality,* Dundee, Scotland.

KEY TERMS

External Metric: A metric used to measure attributes of the product that can be measured only with respect to how the product relates to its environment.

Functionality: The external quality factor that refers to a set of functions and specified properties that satisfy stated or implied needs.

Internal Metric: A metric used to measure attributes of the product that can be measured directly by examining the product on its own, irrespective of its behavior.

Measurement: A process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.

Metric: An empirical assignment of a value in an entity aiming to describe a specific characteristic of this entity.

Quality Manual: A manual used by the software developing company that includes the metrics, the measurement techniques, the guidelines for the application of metrics data analysis, and the corrective actions required for improving the software developing process.

Quality Plan: A plan developed particularly for each software project that includes all the metrics, measurement guidelines and goals applicable for this project only.

Usability: The external quality factor that is defined as a set of attributes that bear on the effort needed for the use and on the individual assessment of such use by a stated or implied set of users.

This work was previously published in Encyclopedia of E-Commerce, E-Government, and Mobile Commerce, edited by M. Khosrow-Pour, pp. 1029-1034, copyright 2006 by Idea Group Reference (an imprint of IGI Global).

Chapter 1.17
A Framework for Communicability of Software Documentation

Pankaj Kamthan
Concordia University, Canada

INTRODUCTION

The role of communication is central to any software development. The documentation forms the *message carrier* within the communication infrastructure of a software project.

As software development processes shift from predictive to adaptive environments and serve an ever more hardware diverse demographic, new communication challenges arise. For example, an engineer may want to be able to remotely author a document in a shell environment without the need of any special purpose software, port it to different computer architectures, and provide different views of it to users without making modifications to the original. However, the current state of affairs of software documentation is inadequate to respond to such expectations.

In this article, we take the position that the ability of documents to be able to communicate at all levels intrinsically depends upon their *representation*. The rest of the article proceeds as follows. We first outline the background necessary for later discussion. This is followed by a proposal for a quality-based framework for representing software documentation in descriptive markup and application to agile software documentation. Next, challenges and avenues for future research are outlined. Finally, concluding remarks are given.

BACKGROUND

Since the origins of software, and subsequently the recognition of software engineering as a discipline, documentation has had an important role to play. The use of documentation in software has a long and rich history (Furuta, Scofield, & Shaw, 1982; Goldfarb, 1981; Knuth, 1992).

There are various means that have been used for expressing software documentation. The documents could for example be expressed in structured natural language text (mimicking typewriting); Rich Text Format (RTF) and its implementations such as Microsoft Word; Hy-

pertext Markup Language (HTML), which supports multiple language characters and symbols that can reach a broad demographic worldwide, incorporates features of print publishing, and supports hyperlinking; the Portable Document Format (PDF); and TEX/LATEX and their variations that are oriented to mathematical typesetting. However, these traditional means suffer from one or more of the following limitations: they are proprietary and can only be authored or rendered by a proprietary software; the focus is not on software engineering but other disciplines such as generic office or scientific use; and the focus is mainly on the presentation or processing rather than on representation.

Descriptive markup (Goldfarb, 1981) is based on a rich model of text known as the ordered hierarchy of content objects (OHCO) (Coombs, Renear, & DeRose, 1987; DeRose, Durand, Mylonas, & Renear, 1997) that lends a hierarchical structure to documents. The Standard Generalized Markup Language (SGML) and its simplification the Extensible Markup Language (XML) are exemplary of descriptive markup. SGML/XML are both *meta-markup* mechanisms that lend a suitable basis for a concrete serialization syntax for expressing information in a software document. They define a document in terms of its OHCO structure with mnemonic names, usually inspired by the domain being addressed, for the content objects of the data. There is a large and increasing base of markup languages based on SGML/XML.

The focus in this article is primarily on XML. Indeed, the use of XML for software process documents has been proposed (Clements et al., 2002; Mundle, 2001). The DocBook/XML markup language has been deployed for software user documentation. These efforts, however, are oriented towards technology rather than descriptive markup or communicability (or quality in general); they do not provide comparisons with other means of representations, and they do not include details of challenges posed during document engineering.

DESCRIPTIVE MARKUP AND REPRESENTATION OF SOFTWARE DOCUMENTATION

We look at a software document from two viewpoints, namely that of a *producer* and that of a *consumer.* Based on that, the representation requirements that we consider pertinent for software documentation are the following:

- **Communicability Concerns for a Document Producer:** A provider, who is responsible for both internal and external documentation, could be interested in any of the following aspects: be able to express the domain under consideration well; have the flexibility of authoring and serving documents in different modalities; readily move documents between computing environments and over networks; easily manage the collection of documents, particularly as they scale (grow in number).
- **Communicability Concerns for a Document Consumer.** A consumer, whose main concern is external documentation, could be interested in any of the following aspects: access the documents on the device he/she is using that may be stand alone or connected to the Internet, and in the natural language/characters of choice; read or listen to the documents in the way he/she prefers that they should be presented; may want to simply look at the table of contents before reading further, or look up the definition of a term used in the main document. In some sense, a consumer would like a document to be "personalized."

These requirements motivated the proposal of a communicability framework, which we now discuss in detail.

A Framework for Communicability of Descriptive Markup-Based Software Documentation

The discussion of software documents and their representations that follows is based on the framework given in Table 1.

Semiotics (Nöth, 1990) is concerned with the use of symbols to convey knowledge. From a semiotics' perspective, a representation can be viewed on three interrelated levels: *syntactic*, *semantic*, and *pragmatic*. Our concern here is the pragmatic level, which is the practical knowledge needed to use a language for communicative purposes. Indeed, the goal of pragmatic quality is comprehension (Lindland, Sindre, & Sølvberg, 1994), and communicability is a prerequisite to that.

We acknowledge that there are time, effort, and budgetary constraints on producing a software document. We therefore include *feasibility*, a part of decision theory, as an all-encompassing factor to make the representation framework practical.

We now consider other aspects of the framework in more detail, starting with the principles underlying documents and their descriptive markup realizations.

Document Engineering Principles with a Descriptive Markup Perspective

The principles presented here are inspired by the established software engineering principles (Ghezzi, Jazayeri, & Mandrioli, 2003).

- **[DEP1] Separation of Concerns:** There are various concerns in document production and delivery, and to manage them effectively, each of these semantically different concerns need to be addressed separately. In particular, separation by parts, time, quality, and views are of special interest. SGML/XML provide means for separating structure, presentation, and logic in a document.
- **[DEP2] Abstraction:** This principle is based on the idea that it is often necessary in documents to highlight only the essentials while suppressing the details. The provision for table of contents and index of terms in a document are examples of abstraction. In descriptive markup, this could be realized by using mnemonic labels `<section>` and

Table 1. A high level view of the elements of a framework for communicability of software documentation

Software Document		
Pragmatic goal	Communicability	Feasibility
Quality attributes of concern	Evolvability, Heterogeneity, Interoperability, Processability, Renderability, Traceability, Universality	
Engineering principles	Abstraction, Anticipation of Change, Formality, Generality, Incrementality, Modularity, Separation of Concerns	
Representation model	Descriptive Markup	

<term> to encapsulate the name of a section and a special term, respectively and collating such appearances to obtain the desired result. Other examples in SGML/XML include the use of comments (<!-- ... -->) and in general metadata, the use of entities, or the use of a style sheet to hide content that one typically does not want rendered.

- **[DEP3] Modularity:** This principle is one way to realize separation by parts and therefore a special case of [DEP1] but is significant enough to be included separately here. [DEP2] is a prerequisite for modularity, which is essential for flexibility of future documents (Malloy, 2005). SGML and particularly XML provide various opportunities for modular documents, for example, by basing OHCO structure on a parent-child hierarchy, by including the concept of entities, or by "hiding" attributes inside the definition of an element in a document.
- **[DEP4] Anticipation of Change:** This principle is based on the premise that change in a (nontrivial) document is inevitable and to accommodate change we must be adequately prepared. By being text based (rather than binary) and being vendor and device independent, SGML/XML support this principle.
- **[DEP5] Incrementally:** This principle is related to [DEP1] and [DEP4] and is based on the notion that creating a representation of a document in increments should be encouraged. SGML/XML are in agreement with this. Indeed, it is a common markup practice to develop documents iteratively (in manageable steps) and check for conformance after each iteration.
- **[DEP6] Generality:** This principle is based on the assertion that the more general a document representation is, the broader the audience it can reach and the more reusable in different situations it is. SGML/XML are neutral to user context, vendor, domain, device, network, or programming language. They also support the largest character set currently available, namely the Universal Character Set (UCS)/Unicode.
- **[DEP7] Formality:** This principle, by requiring a formal (logical) syntax and semantics, aims to reduce the potential of ambiguity, contradictions, or misinterpretations and enables a more precise description of documents. XML has a well-defined syntax and semantics (Renear, Dubin, Sperberg-McQueen, & Huitfeldt, 2002). The Document Type Definition (DTD) provides a grammar for structural and data type constraints on the syntax and content of the elements and attributes in SGML/XML documents. In the case of XML, the capabilities of DTD have been strengthened in other grammar languages such as XML Schema and RELAX NG.

Attributes Impacting Communicability of a Software Document

In the following discussion we consider communicability as a "meta-concern" and discuss the low level list of attributes to address it. We also highlight relevant relationships between these attributes and with the abovementioned principles where needed.

- **[CA1] Evolvability:** This attribute especially depends on [DEP1-5]. Once a software document is created, it is highly likely that it will be maintained for corrective, adaptive, or perfective purposes. For example, a user manual may need to be transformed on computers with different capabilities for users with individual needs. The Resource Description Framework (RDF) and Dublin Core Metadata Element Set (DCMES) provide support for metadata (such as author information, date/time, history, or version-

ing) that can help track modifications.

- **[CA2] Heterogeneity:** This attribute especially depends on [DEP4-6] and is related to [CA3]. The elements of a document may need to be represented in a variety of different forms such as text, graphic, mathematical symbols, and so forth, and subsequently aggregated. Therefore, a representation must accommodate the possible compound or heterogeneous nature of a document. SGML/XML documents can be heterogeneous, where fragments of different markup (or even nonmarkup as long as the characters in it do not violate the document's character encoding) could be placed in a single document. XML Inclusions (XInclude) allows multiple possibilities of reuse: using a fragment of another document, a fragment of the current document, or an entire document (Figure 1).
- **[CA3] Interoperability:** This attribute is related to [CA2]. The heterogeneous forms in a document not only need to coexist in the same information container but also need to "talk" (interface) with each other and with the parent document. Namespaces in XML is a mechanism for uniquely identifying XML elements and attributes of a markup language, thus making it possible to create heterogeneous documents that unambiguously mix elements and attributes from multiple different XML documents (Figure 2).
- **[CA4] Processability:** This attribute depends on all [DEP1-7] and supports [CA5]. At times, we may need to manipulate (recast/transform, extract/filter, query, and so on) software documents to suit different circumstances. Support for querying XML documents is provided by XQuery and client- or server-side tree-based processing of XML documents is enabled by the Document Object Model (DOM) for which stable implementations are available. Extensible Stylesheet Language (XSL) is a style sheet language for associating presentation semantics with arbitrarily complex XML documents, while its companion XSL Transformations (XSLT) is a style sheet language for transforming XML documents into other documents, including nonXML, documents (Figure 2).
- **[CA5] Renderability:** This attribute especially depends on [DEP1-2]. A software

Figure 1. Opportunities of reuse for a software document in an XML environment

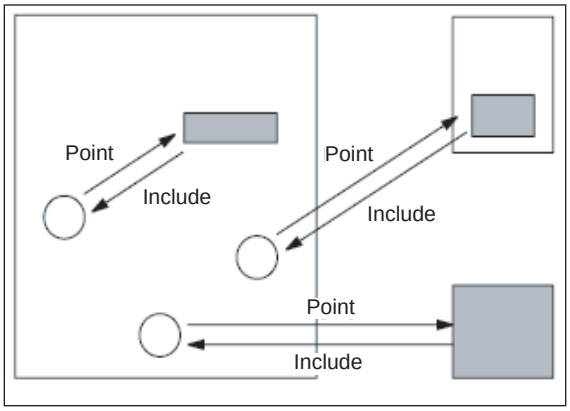
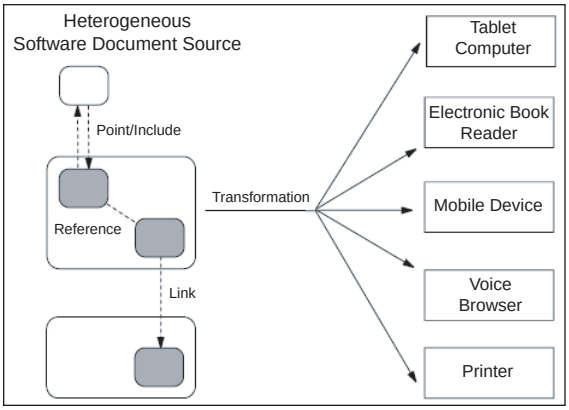


Figure 2. A heterogeneous software document in a descriptive markup transformation environment



document must ultimately be rendered to a user's environment, and therefore presentation semantics (such as fonts, horizontal and vertical layout, pagination, and so on). The Document Style Semantics and Specification Language (DSSSL) is a style sheet language for SGML documents (that inspired XSL/XSLT). The Cascading Style Sheets (CSS) is a style sheet language for presenting simple XML documents on devices and agents with a variety of different configurations such as in Figure 2.

- **[CA6] Traceability:** This attribute depends on [DEP1,3]. A trace from `A` to `B` requires identification and location of, and means to reach, `B` from `A`. XML provides an `id` attribute for local identification of an element that can be used in conjunction with the Uniform Resource Identifier (URI) for global identification. HyTime, an ISO Standard, provides sophisticated linking functionalities for SGML documents including multi-directional links and linking to nonSGML data such as video clips. XML Linking Language (XLink) extends the unidirectional linking support in HTML and provides powerful bidirectional linking capabilities necessary for hypertext.
- **[CA7] Universality:** This attribute depends on [DEP6], and its underlying premise is that software documents should be created for all, irrespective of their individual context. XML is in agreement with the standards for accessibility and internationalization.

Application of the Framework: The Case of Agile Documentation

The bureaucracy inherent to traditional predictive software development processes and their perhaps over emphasis on documentation has in the last decade led to a shift to adaptive environments such as Extreme Programming (XP) and the Unified Process (UP). Although they are not document driven, documents continue to play an important role in stakeholder communication.

The term *agile document* was introduced in (Ambler, 2002) to imply customer-oriented, lightweight documents that could serve XP and UP. A collection of patterns for writing effective agile software documentation is given in (Rüping, 2003). Many of the patterns in it that deal with presentation and representation of documents can be concretely dealt with in our framework. For example, communicability aspects of structuring individual documents patterns are given by [CA1-7], layout and typography patterns are subsumed by [CA4, 5], whereas infrastructure and technical organization patterns are covered by [DEP1] and [CA1-4, 6].

Challenges to the Descriptive Markup Approach to Software Documentation

There are a few technical challenges to representations in descriptive markup. For example, descriptive markup in its source form, particularly when there are complex (usually nonlinear) structural relationships involved, can be error prone for direct authoring and is not considered very readable. There are openly available and mature SGML/XML authoring tools that ameliorate this to a certain extent. The processing based on DOM or transformations based on XSLT lead to in-memory tree, which is not always efficient for large documents for demanding situations (say, being transformed over a low bandwidth network in real time). One solution to this approach has been to use XSLT compilers instead of interpreters. Similarly, event-based processing such as Simple API for XML (SAX) could complement the DOM.

FUTURE TRENDS

As software documents become increasingly large in size and number and aim to serve diverse users, a systematic document engineering process is highly desirable. There are currently limited efforts in that direction (Glushko & McGrath, 2005).

XML, like any other technology, has its own set of issues if not used appropriately while authoring or processing (Harold, 2003) or if used beyond its scope (Megginson, 2005). In fact, addressing the issue of quality in all early software representations is crucial. To do that, an evaluation of XML using the Cognitive Dimensions of Notations (CDs) (Green, 1989), a generic framework for describing the utility of information artifacts by taking the system environment and the user characteristics into consideration, would be of interest.

A natural amplification of the previous discussion is a closer synergy with the current knowledge representation initiatives that are based on descriptive markup. The Semantic Web has recently emerged as an extension of the current Web that adds technological infrastructure for better knowledge representation, interpretation, and reasoning (Hendler, Lassila, & Berners-Lee, 2001) and could provide much more powerful representations of knowledge in a software document.

Finally, in spite of its broad use, documentation is not automatically useful by itself: Its value is realized only if it is well done (Weinberg, 1998), which could be viewed as one of or a combination of more than one dimensions of communicability. As documents become more like interactive applications, where they act as *user interfaces* to services provided by the software, the significance of assuring and evaluating their communicability will only increase.

CONCLUSION

Representation of documents so as to assure their communicability to the participants at all levels is critical. One way to foster that is by *partitioning* communicability into a manageable number of dimensions that could be dealt with directly and by choosing a suitable means of representation of software documents such as descriptive markup. The principles and attributes presented in our framework provide a basis for communicability dimensions and the SGML/XML *family* provides established technologies for descriptive markup.

The most important requirement for communicability from both the producer's and the consumer's standpoint is that they should have *options.* Indeed, if software documents are seen as carriers of commonsense knowledge (Minsky, 2000), then we should not seek one uniform way to present or represent them. This is echoed in the Redundant Recoding principle (Green, 1989), which is the ability to express information in a representation in more than one way, each of which simplifies different cognitive tasks. For that, documents need to be systematically created and strive for high quality. They also need to become *intelligent,* which in computational context usually implies that they carry *knowledge* amenable for automated processing. The Semantic Web provides a vehicle for representing knowledgeable software documents in descriptive markup.

REFERENCES

Ambler, S.W. (2002). *Agile modeling: Effective practices for extreme programming and the unified process.* John Wiley & Sons.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., et al. (2002). *Documenting software architectures: Views and beyond*. Addison-Wesley.

Coombs, J. H., Renear, A. H., & DeRose, S. J. (1987). Markup systems and the future of scholarly text processing. *Communications of the ACM, 30*(11), 933-947.

DeRose, S. J., Durand, D. G., Mylonas, E., & Renear, A. H. (1997). What is text, really? *Journal of Computer Documentation, 21*(3), 1-24.

Furuta, R., Scofield, J., & Shaw, A. (1982). Document formatting systems: Survey, concepts, and issues. *ACM Computing Surveys, 14*(3), 417-472.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). *Fundamentals of software engineering* (2nd ed.). Prentice Hall.

Glushko, R. J., & McGrath, T. (2005). *Document engineering.* Cambridge, MA: MIT Press.

Goldfarb, C. F. (1981, June 8-10). A generalized approach to document markup. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, (pp. 68-73).

Green, T. R. G. (1989). Cognitive dimensions of notations. In V. A. Sutcliffe & L. Macaulay (Eds.), *People and computers* (pp. 443-460). UK: Cambridge University Press.

Harold, E. R. (2003). *Effective XML*. Addison-Wesley.

Hendler, J., Lassila, O., & Berners-Lee, T. (2001). The Semantic Web. *Scientific American, 284*(5), 34-43.

Knuth, D. E. (1992). *Literate programming* (CSLI Lecture Notes, Number 27). USA: Stanford University, Center for the Study of Language and Information.

Lindland, O. I., Sindre, G., & Sølvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software, 11*(2), 42-49.

Malloy, T. (2005, November 2-4). The future of documents. *Proceedings of the 2005 ACM Symposium on Document Engineering (DocEng 2005)*, Bristol, UK, (pp.1-1).

Megginson, D. (2005). *Imperfect XML*. Addison-Wesley.

Minsky, M. (2000). Commonsense-based interfaces. *Communications of the ACM, 43*(8), 66-73.

Mundle, D. (2001, May 15). Using XML for software process documents. *Proceedings of the XML Technologies and Software Engineering (XSE 2001)*, Toronto, Canada.

Nöth, W. (1990). *Handbook of semiotics*. Bloomington: Indiana University Press.

Renear, A., Dubin, D., Sperberg-McQueen, C. M., & Huitfeldt, C. (2002, November 8-9). Towards a semantics for XML markup. *Proceedings of 2002 ACM Symposium on Document Engineering (DocEng 2002)*, McLean, USA (pp. 119-126).

Rüping, A. (2003). *Agile documentation: A pattern guide to producing lightweight documents for software projects.* John Wiley & Sons.

Spiel, C. (2002). Writing documentation, Part III: DocBook/XML. *Linux Gazette, 75.*

Weinberg, G. M. (1998). *The psychology of computer programming (silver anniversary edition).* New York: Dorset House.

KEY TERMS

Agile Document: A customer-oriented lightweight document that need not be perfect but just good enough.

Descriptive Markup: A model of text that focuses on the description of information using markup delimiters for consumption by both humans and machines.

Document Engineering: A discipline that is concerned with principles, tools, and processes that improve the ability to create, manage, and maintain documents.

Knowledge Representation: The study of how knowledge about the world can be represented and the kinds of reasoning that can be carried out with that knowledge.

Ontology: An explicit, formal specification of a conceptualization that consists of a set of terms in a domain and the relations among them.

Semiotics: The field of study of signs and their representations.

Single Source Approach: A technique that encourages a once-only creation of a resource, such as a document, in a manner so that it could be reused or repurposed for different contexts.

This work was previously published in Encyclopedia of Information Science and Technology, Second Edition, edited by M. Khosrow-Pour, pp. 1574-1579, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.18
Intelligent Analysis of Software Maintenance Data

Marek Reformat
University of Alberta, Canada

Petr Musilek
University of Alberta, Canada

Efe Igbide
University of Alberta, Canada

ABSTRACT

Amount of software engineering data gathered by software companies amplifies importance of tools and techniques dedicated to processing and analysis of data. More and more methods are being developed to extract knowledge from data and build data models. In such cases, selection of the most suitable data processing methods and quality of extracted knowledge is of great importance. Software maintenance is one of the most time and effort-consuming tasks among all phases of a software life cycle. Maintenance managers and personnel look for methods and tools supporting analysis of software maintenance data in order to gain knowledge needed to prepare better plans and schedules of software maintenance activities. Software engineering data models should provide quantitative as well as qualitative outputs. It is desirable to build these models based on a well-delineated logic structure. Such models would enhance maintainers' understanding of factors which influence maintenance efforts. This chapter focuses on defect-related activities that are the core of corrective maintenance. Two aspects of these activities are considered: a number of software components that have to be examined during a defect removing process, and time needed to remove a single defect. Analysis of the available datasets leads to development of data models, extraction of IF-THEN rules from these models, and construction of ensemble-based prediction systems that are built based on these data models. The data models are developed using well-known tools such as See5/C5.0 and 4cRuleBuilder, and a new multi-level evolutionary-based algorithm.

Single data models are put together into ensemble prediction systems that use elements of evidence theory for the purpose of inference about a degree of belief in the final prediction.

INTRODUCTION

Many organizations want to prepare reliable schedules of maintenance tasks. Such schedules would lead to on-time realization of these tasks and better management of resources. This is an important issue, especially in the case where maintenance tasks account for more than half of a typical software budget (Glass, 1989; Smith, 1999). Because of that, the software industry is exhibiting an increased interest in improving software maintenance processes. Software engineers use a number of different tools to support maintenance activities and make them more efficient. The most commonly used tools are tools for model-based software component analysis, metrics extraction, measurements presentation, and statistical analysis and evaluation. Besides that, software maintainers need tools that would help them to understand relationships between attributes of software components and maintenance tasks. Knowledge gained in this way would increase understanding of influence of software component attributes, such as size of code, complexity, functionality, and so forth, on efforts associated with realization of maintenance tasks.

There are four different categories of software maintenance: *corrective*—it involves changing software to remove defects; *adaptive*—it leads to changing software due to changes in software operating environment; *perfective*—it embraces activates that lead to improvement of maintainability, performance, or other software quality attributes; and *preventive*—it is defined as maintenance performed for the purpose of preventing problems before they happen. The corrective software maintenance is associated with activities related to elimination of software defects. This process is a key factor in ensuring timely releases of software and its updates, and high quality of software. Different tools and systems are used to support activities that are directly related to correction of defects. However, there is also a need to build systems that support decision-making tasks and lead to preparation of schedules and plans for defect removal processes. These systems should not only provide quantitative predictions but also give indications about plausibility of these predictions. Additionally, they should provide maintenance engineers with knowledge about defect removal efforts that explain obtained predictions. In summary, it is desirable to have a tool equipped with the ability to retrieve knowledge about relationships between attributes describing software and factors that directly or indirectly influence defect elimination activities.

Some of the important questions asked by managers and software maintenance engineers regarding removal of defects from software systems are:

- Does a defect removal process depend on functionality of software components?
- Does a defect removal process depend on the time when a defect entered the system?
- What are the factors that influence time needed to correct a single defect?
- What kind of relations between software component attributes and time needed to remove a defect can be found from software maintenance data?
- How confident can someone be about dependencies that have been found between a defect removal process and attributes of software components?

This chapter focuses on building software maintenance data models and their analysis. The aim is to build a prediction system that is able to provide software maintenance engineers with predictions regarding defect elimination efforts, knowledge about factors that influence these

efforts, and confidence measures for obtained predictions and gained knowledge.

Fulfillment of these objectives is achieved by the application of soft computing and machine learning methods for processing software engineering data. In a nutshell, the main idea of the proposed approach is to use multiple data processing techniques to build a number of data models, and then use elements of evidence theory to "merge" the outcomes of these data models. In this context, a prediction system is built of several rule-based models. The attractiveness of these models comes from the fact that they are built of IF-THEN rules that are easy to understand by people. In this chapter, three different tools for constructing IF-THEN rules are used. One of them constructs rules directly from the data, and the other two build decision trees first and extract rules from the trees. As the result, a large set of IF-THEN rules is created. Each rule is evaluated based on its capability to perform predictions. This evaluation is quantified by degrees of belief that represent goodness of the rules. The degrees of belief assigned to the rules that are fired, for a given input data point, are used to infer an overall outcome of the prediction system. The inference engine used in the system is built based on elements of evidence theory.

This chapter can be divided into three distinctive parts. The first part embraces background information related to data models. It starts with a description of work related to the area of software engineering data models (the *Software Data Models* section). An overview of rule-based systems and ensemble prediction systems is in the *Rule-Based Models and Ensemble Systems* section. The *Evidence Theory and Ensemble-Based System* section contains a description of the proposed ensemble-based prediction system. This section also contains an overview of the concept of the proposed system, its basic components, and the inference engine. The second part of the chapter is dedicated to the description of the datasets used here and the results of their analysis. In the *Software Engineering Maintenace Data* section, a software dataset is presented. Predictions of efforts needed to remove defects are presented in the *Base-Level Data Models, Extracted Knowledge and Confidence in Results* section. This section also includes a set of IF-THEN rules which represent knowledge extracted from the software maintenance models. Descriptions of ensemble-based prediction systems and analysis of their predictions are in the *Ensemble-Based Prediction System* section. And, finally, there is the *Conclusions* section.

The third part of the chapter contains three appendices: Appendix I is a short introduction to Evolutionary Computing; Appendix II is a brief introduction to the topic of decision trees and an evolutionary-based technique used for construction of decision trees; and Appendix III describes elements of evidence theory and the transferable belief model.

SOFTWARE DATA MODELS

Software engineering data collected during development and maintenance activities are seen as a valuable source of information about relationships that exist among software attributes and different aspects of software activities. A very important aspect that becomes very often associated with software systems is software quality. Software quality can be defined two-fold: (1) as the degree to which a system, component, or process meets specified requirements; and (2) as the degree to which a system, component, or process meets customer or user needs or expectations (IEEE 610.12). In light of that definition, the most essential expectation is the absence of defects in software systems. This aspect alone touches almost every phase of a software life cycle. The coding, testing, and integration, as well as the maintenance phase are directly related to the issue of constructing high quality software. However, building and maintaining a defect-free system

is not a trivial task. Many activities, embraced under the umbrella of software quality assurance, are performed in order to detect defects, localize them in a code, remove them, and finally verify that the removal process has been successful. In order to plan these activities effectively, there is a need to understand what makes a code defective, and how much time and effort is needed to isolate and eliminate defects. Development of software prediction models attempts to address those issues.

Prediction models aim at predicting outcomes on the basis of a given set of variables: they estimate what should be the outcome of a given situation with a certain condition defined by the values of the given set of variables. The steps that have to be performed during development of such models are:

- selection of the outcome attribute;
- selection of predictor (input) variables;
- data collection;
- assembly of the model;
- validation of the model; and
- updates and modifications of the model.

In the case of software engineering, prediction models that can be used to predict a number of quantities related to software quality and maintenance activities, are used for many years. These models proved to provide reasonable accuracy (Schneidewind, 1995, 1997). Many different software metrics are utilized as predictor (input) variables. The most common ones are complexity and size metrics, testing metrics, and process quality data.

The most popular software prediction models are models predicting quality-related aspects of software modules. A number of these models have been reported in the literature:

- **Tree-Based Models:** Both classification and regression trees are used to categorize software modules and functions; different regression tree algorithms—CART[1]-LS (least squares), S-PLUS, and CART-LAD (least absolute deviation)—are used to build trees to predict the number of faults in modules in Khoshgoftaar and Seliya (2002); in another case, regression trees are constructed using a concept of fault density (Gokhale & Lyu, 1997); a study on the use of a classification tree algorithm to identify fault prone software modules based on product and process metrics is presented in Khoshgoftaar and Allen (1999); tree-based models are used to uncover relationships between defects and software metrics, and to identify high-defect modules together with their associated measurement characteristics (Takahashi, Muroaka, & Nakamura, 1997; Troster & Tian, 1995);
- **Artificial Neural Network-Based Models:** Neural networks are recognized for their ability to provide good results when dealing with data that have complex relationships between inputs and outputs; neural networks are used to classify program modules as either high or low risk based on two criteria—a number of changes to enhance modules and a number of changes to remove defects from modules (Khoshgoftaar & Allen, 1998; Khoshgoftaar & Lanning, 1995);
- **Case-Based Reasoning Models:** Case-based reasoning (CBR) relies on previous experiences and uses analogy to solve problems; CBR is applied to predict software quality of the system by discovering fault-prone modules using product and process metrics as independent variables (Berkovich, 2000);
- **Fuzzy-Based Models:** Concepts of fuzzy sets and logic are used to build data models using linguistic labels; the work related to building fuzzy-based systems for prediction purposes is presented in Reformat, Pedrycz, and Pizzi (2004), where fuzzy neural networks are constructed for defect

predictions; a fuzzy clustering is used in Yuan, Khoshgoftaar, Allen, and Ganesan (2000), where a modeling technique that integrates fuzzy subtractive clustering with module-order modeling for software quality prediction is presented, a case study of a large legacy telecommunication system to predict whether each module should be considered fault-prone is conducted;

- **Bayesian Belief Network-Based Models:** Bayesian belief networks (BBN) address a complex web of interconnections between multiple factors; belief networks are used for modeling the complexities of software taking uncertainty into consideration (Neil & Fenton, 1996; Neil, Krause, & Fenton, 2003). BBN are also applied to construct prediction models that focus on the structure of the software development process explicitly representing complex relationships between metrics (Amasaki, Takagi, Mizuno, & Kikuno, 2003).

Thorough comparisons of different approaches used for building software quality prediction models can be found in Fenton and Neil (1999, 2005) and Khoshgoftaar and Seliya (2003).

Besides classification of software modules, a close attention is also given to the issues related to prediction of efforts associated with detection and correction of defects. One of the first papers dedicated to the topic of prediction of maintenance efforts is Jorgensen (1995). This paper reports on the development and use of several software maintenance effort prediction models. These models are developed applying regression analysis, neural networks, and the optimized set reduction method. The models are used to predict maintenance task efforts based on the datasets collected from a large Norwegian company. The variables included in the effort prediction models are: a cause of task, a degree of change on a code, a type of operation on a code, and confidence of maintainer. An explanation of efforts associated with software changes made to correct faults while software is undergoing development is investigated in Evanco (1999, 2001). In this case, the ordinal response models are developed to predict efforts needed to isolate and fix a defect. The predictor variables include extent of change, a type of change, an internal complexity of the software components undergoing the change, as well as fault locality and characteristics of the software components being changed. The models are developed and validated on three Ada projects. A model for estimating adaptive software maintenance efforts in person hours is described in Hayes, Patel, and Zhao (2004). A number of metrics, such as the lines of code changed and the number of operators changed, are found to be strongly correlated to maintenance efforts.

RULE-BASED MODELS AND ENSEMBLE SYSTEMS

Data models can be categorized into two major groups: black-box models and white-box models. The black-box models provide a user with the output values without indicating a way in which those outputs are calculated. This means that knowledge about relationships existing between the inputs and the output is not discovered. Conversely, the white-box data models, also called transparent models, allow their users to gain knowledge about the data being modeled. A careful inspection of a model's structure and its analysis provides an insight into relationships existing among values of data attributes. Rule-based models are well-known examples of white-box models. A rule-based model consists of a number of IF-THEN rules. A number of different techniques for development of IF-THEN rules exist. Some of these techniques construct rules directly from the data, while others build decision tress first and then extract rules from the trees.

Rule-Based Models

Rule-based modeling is a most common form of computational model. Rules are generally well suited to study behavior of many different phenomena. These models receive information describing a situation, process that information using a set of rules, and produce a specific response as their output (Luger, 2002; Winston, 1992). Their overall structure is presented in Figure 1.

In its simplest form, a rule-based model is just a set of IF-THEN statements called rules, which encode knowledge about phenomenon being modeled. Each rule consists of an IF part called the premise or antecedent (a conjunction of conditions), and a THEN part called the consequent or conclusion (predicted category). When the IF part is true, the rule is said to fire, and the THEN part is asserted—it is considered to be a fact.

A set of IF-THEN rules is processed using Boolean logic. The expert system literature distinguishes between "forward chaining" and "backward chaining" as a method of logical reasoning. Forward chaining starts with a set of characteristics about a situation—a feature vector of independent variables—and applies these as needed until a conclusion is reached. Backward chaining, in contrast, starts with a possible conclusion—a hypothesis—and then seeks information that might validate the conclusion. Forward chaining systems are primarily data-driven, while backward chaining systems are goal-driven. A forward chaining method of reasoning is used in prediction systems.

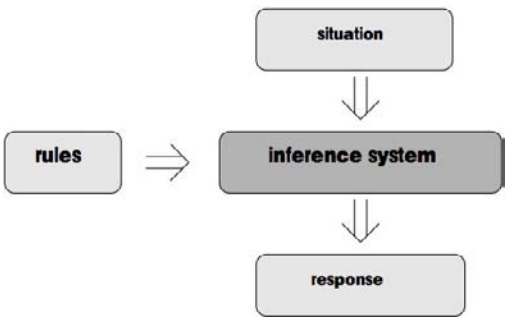
The IF-THEN based model with forward chaining works in the following way: the system examines all the rule conditions (IF) and determines a subset, the conflict set, of the rules whose conditions are satisfied. Of this conflict set, one of those rules is triggered (fired). When the rule is fired, any actions specified in its THEN clause are carried out. Which rule is chosen to fire is a function of the conflict resolution strategy. Which strategy is chosen can be determined by the problem and is seen as one of the important aspects of the development of rule-based systems.

In any case, it is vital as it controls which of the applicable rules are fired and thus how the entire system behaves. There are several different strategies, but here are a few of the most common ones:

- **First Applicable:** Is based on the fact that the rules are put in a specified order. The first applicable rule is fired.
- **Random:** It is based on a random selection of a single rule from the conflict set. This randomly selected rule is fired.
- **Most Specific:** This category is based on the number of conditions attached to each rule. From the conflict set, the rule with the most conditions is chosen.
- **Least Recently Used:** Is based on the fact that each rule is accompanied by a time stamp indicating the last time it was used. A rule with the oldest time stamp is fired.
- **"Best" Rule:** This is based on the fact that each rule is given a "weight" which specifies how much it should be considered over other rules. The rule with the highest weight is fired.

Another important aspect of the development of rule-based models, besides the reason-

Figure 1. Structure of a rule-based model



ing scheme, is the generation of rules. As it has been mentioned earlier, there are many different methods and techniques of doing that. IF-THEN rules can be generated on the basis of expert knowledge where they are created by a person during interaction with a domain expert, or automatically derived from available data using a variety of different approaches and tools (see Appendix II; RuleQuest; 4cData).

Ensemble Systems

A ensemble system is composed of several independently built models called base-level models. Each base-level model is developed differently by applying different construction techniques/methods using a single set of training data points, or using single technique that is applied to different data point subsets. The prediction outcome of such a system is based on processing outputs coming from all base-level models being part of the system. The process of construction of a ensemble system embraces two important phases (Todorovski & Dzeroski, 2000):

- the generation of a diverse set of base-level models; and
- the combination of outcomes generated by these models.

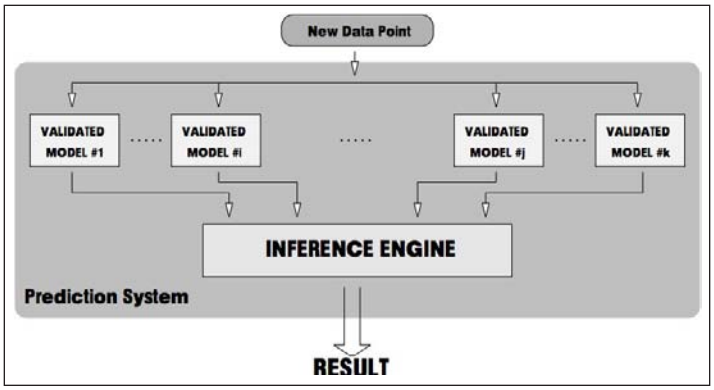
There are two groups of approaches for generation of base-level models. The first group can be represented by probably the most popular and simplest approach where a single learning algorithm is applied to different subsets of training data. Two best known methods are: random sampling with replacement called bagging (Breiman 1996; Todorovski et al., 2000), and re-weighting misclassified training data points called boosting (Freund & Schapire, 1996; Todorovski et al., 2000). The other group of methods is based on applying some modifications to model construction algorithms while using an identical set of training data points. A number of works is dedicated to comparison of such methods (Ali, 1995; Ali & Pazzani, 1996; Kononenko & Kovacic, 1992; Kwok & Carter, 1990).

The second important task of constructing ensemble systems is the fusion of outputs generated by base-level models. The most popular techniques applied here are distribution summation, voting, and naïve Bayesian combination (Kononenko et al., 1992). Another technique of combining models' outputs is based on the application of meta-decision trees. The role of meta-decision trees is to specify which model should be used to obtain a final classification. An extensive experimental evaluation of a new algorithm for learning meta-decision trees based on C4.5 algorithm was performed in Todorovski et al. (2000). In total, five learning algorithms of generating multi-model systems have been compared: two algorithms for learning decision trees, a rule learning algorithm, a nearest neighbor algorithm, and a naïve Bayes algorithm (Winer, Brown, & Michels, 1991). A building system based on multiple models can improve the accuracy and stability of the system significantly.

EVIDENCE THEORY AND ENSEMBLE-BASED SYSTEM

Application of different methods for generation of IF-THEN rules may lead to the discovery of different relationships among data attributes. The same situation occurs when a single rule-generation method is used on different subsets of the same data. The aim is to take advantage of that and build a system that combines many rule-based models and generates a single output based on the outputs of the base-level models. The fact that the rules, which constitute the models, are not equally good creates a need for taking that into consideration. This means that a process of combining the outputs of rules has to use information about prediction capabilities of the rules, that is, information about a number of proper and

Figure 2. The structure of an ensemble-based prediction system



false predictions made by each rule. The proposed system addresses these issues. It is based on the utilization of the following ideas:

- **Application of a number of rule-based models constructed using different methods on the same/or different subsets of data:** This provides means for thorough exploitation of different extraction techniques and increases possibilities of discovering significant facets of knowledge embedded in the data.
- **Application of the concept of basic belief masses from evidence theory (Shafer, 1976; Smets, 1988) that are used to represent goodness of the rules:** This provides assessment of quality of the rules from the point of view of their prediction capabilities.
- **Utilization of the transferable belief model (Smets, 1994):** Built on the evidence theory to reason, based on basic belief masses, about a given data point belonging to a specific category and to provide a confidence measure for the result.

Concept

The pivotal point of the proposed system is the application of elements of evidence theory. The basic concept of the theory—basic belief mass (see Appendix III for details)—is also a fundamental concept used to build the proposed prediction system. In a nutshell, the approach can be described in just a few sentences: all IF-THEN rules are treated as propositions equipped with basic belief masses (*bbm* in short); the *bbm* of all rules which are fired at a given time are used by an inference engine to derive probabilities of occurrence of different outcomes (the universe of possible outcomes is defined a priori and is related to the phenomenon under investigation).

The *bbm* represents the degree of belief that something is true. Once *bbm* is assigned to a rule it means that if the rule is satisfied by a given data point then there is the belief equal to *bbm* that this data point belongs to a category indicated by the rule. At the same time, the belief of value *1-bbm* is assign to a statement that it is not known to which category the data point belongs. In other words, every rule, which is satisfied by a given data point "generates" two numbers:

- one that indicates a belief that a given data point belongs to a category indicated by the rule (its value is equal to *bbm*); and
- one that indicates that a given data point can belong to any category (its value is *1-bbm*).

Of course, the higher the *bbm* value of the rule, the higher the belief that a given data point belongs to a category indicated by the rule, and the smaller the belief that a data point can belong to any category.

Figure 2 presents the structure of the system and the data flow during a prediction process. The system is composed of a set of validated models and an inference engine. For prediction purposes, a new data point is fed into each model. The *bbm* values of all rules fired by the data point together with categories identified by these rules constitute the input to the inference engine. Details of this engine are presented in the sub-section, *Inference Engine.*

Development and Validation of IF-THEN Models

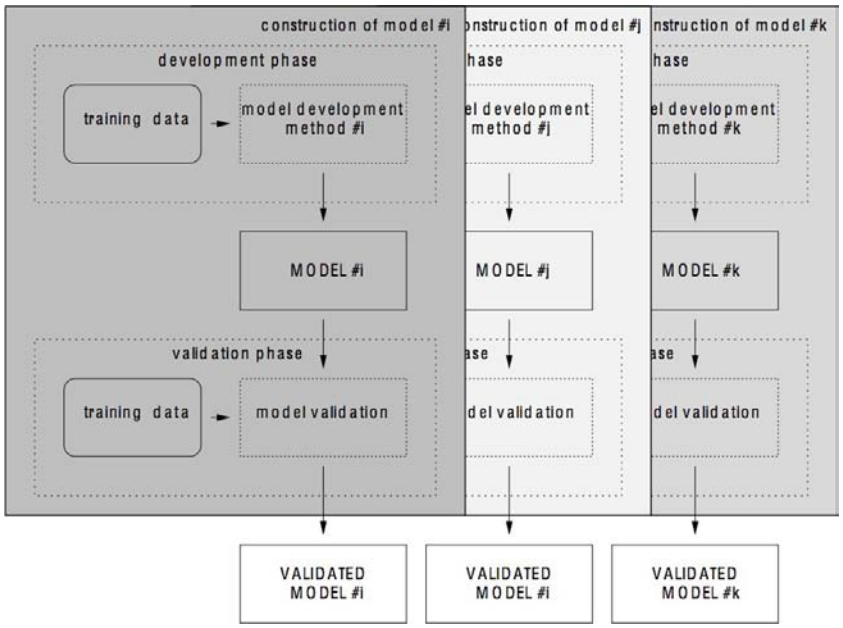
The construction stages of the proposed ensemble-based prediction system are shown in Figure 3. The first step in this process is the development of IF-THEN based models using different techniques. These models are built based on a subset of available data, which is called a training dataset.[2] This process can be performed multiple times using different training sets. This means better exploration and exploitation of the available data.

The value of *bbm* is assigned to each rule based on its coverage and prediction rate. Each rule of each model is "checked" against all training data points. This process results in generation of bbm_T (*T* stands for training) values. They are indicators of goodness of the rules. The formula (called the Laplace ratio) used for calculations of bbm_T is:

$$bbm_T = \frac{No\_Classified_T + 1}{No\_Classified_T + No\_Misclassified_T + 2} \quad (1)$$

where $No\_Classified_T$ represents a number of training data points properly classified for a given category, and $No\_Misclassified_T$ represents a total number of training data points that has been misclassified by the rule.

Figure 3. Construction of an ensemble-based prediction system: Development and validation stages



The first step in the evaluation of goodness of rules is performed based on their prediction capabilities tested against the data used for their development. To make this evaluation more reliable, the second step of the evaluation of rules' goodness is performed. This process is based on the performance of the rules when they are "tested" against data that has not been used during the development purposes. This subset of the data is called a validation dataset. Bbm_V (where V stands for validation) values are calculated:

$$bbm_V = \frac{No\_Classified_V + 1}{No\_Classified_V + No\_Misclassified_V + 2} \quad (2)$$

This time, $No\_Classified_V$ represents a number of validation data points properly classified for a given category, and $No\_MisClassified_V$ represents a total number of validation data points misclassified by the rule.

Following that, both bbm_T and bbm_V values are combined. It means that bbm_T values are *updated*. For this purpose, the Dempster's combination rule is used (Appendix III, Equation 8). The formula is:

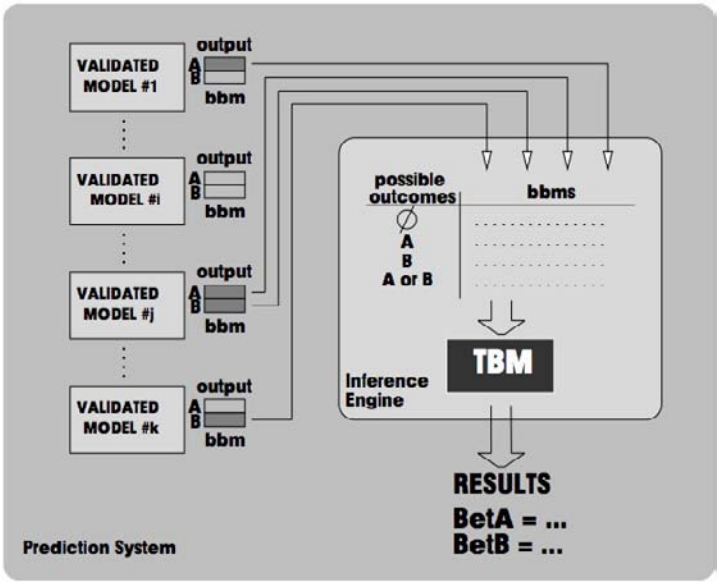
$$bbm_{UPDATE} = (1.0 - (1.0 - bbm_T) * (1.0 - bbm_V)) \quad (3)$$

These two stages, as shown in Figure 3, constitute the construction process of the ensemble-based prediction system. For each split of data into training and validation sets, a number of different models can be constructed. Overall, the process results in a number of different base-level models, which are building blocks of the proposed prediction system.

Inference Engine

The base-level models developed using different methods and data subsets together with the inference engine constitute the ensemble-based prediction system (Figure 2). The inference engine is an implementation of the following concept. A new data point is sent to all models. It activates a number of rules that leads to generation of

Figure 4. The inference engine, its inputs and outputs (a case with two possible outcomes)



outputs and *bbm* (bbm_{UPDATE} to be precise) values. The *bbm* values indicate the belief that this data point belongs either to the same or to a number of different categories. In the case where a single category is identified by all outputs, the inference engine is not engaged—it is predicted that the data point belongs to the identified category. In the case where a number of different categories are identified, the transferable belief model (TBM) (Smets & Kennes, 1994) is used to derive possibilities that the data point belongs to each of the identified categories. This process is represented in Figure 4. All models that have their rules fired provide results and *bbm* values to the inference engine. A table filled with *bbm* values is constructed. Each row of this table represents a possible combination of existing categories in the domain of interest (*Ø, A, B, A or B* in Figure 4). The TBM processes this table. The results are presented as pignistic probabilities of the data point belonging to different categories. This process is illustrated by a simple example.

Table 1. Bbm values of model outputs

Model	Output	IF-THEN rule	bbm
first	I	M1-c1-R1	0.60
		M1-c1-R2	**0.85**
		M1-c1-R3	**0.75**
	II	M1-c2-R1	0.67
		M1-c2-R2	0.94
		M1-c2-R3	0.80
second	I	M2-c1-R1	0.80
		M2-c1-R2	0.90
		M2-c1-R3	0.67
	II	**M2-c2-R1**	**0.90**
		M2-c2-R2	0.94
		M2-c2-R3	**0.85**

Example

Two IF-THEN based models have been built to represent relationships between an n-dimensional input and a single-dimensional output. There are two categories distinguished in the output space: *I* and *II*. Each of the models contains three IF-THEN rules for each category. Rules and their *bbm* values are shown in the Table 1.

Let's assume that a new data point has been obtained and the system is used to predict which category this point belongs to. When checked against all rules listed previously, the input data point satisfies the following set of rules: from the first model—M1-c1-R2, M1-c1-R3, and from the second—M2-c2-R1 and M2-c2-R3. *Bbm* values associated with satisfied rules are: 0.85 and 0.75 from the first model, and 0.90 and 0.85 from the second model. Table 2 is prepared based on these values. In the case of the first model, two rules are fired. The values 0.85 and 0.75 represent beliefs that the data point belongs to the category *I*, the values 0.15 and 0.25 indicate that there is still belief that this point can belong to any category—in this case, category *I* or category *II*. The same process is repeated for the rules from the second model. The values in Table 2 are the input to TBM.

Inside TBM, the *bbm* values are combined using the conjunctive combination rule (Appendix III, Equation 11). The results of this operation are presented in Table 3.

The next step is to calculate pignistic probabilities *BetP*. One of these probabilities represents the belief that the data point belongs to the category *I*, another that it belongs to the category

Table 2. Bbms assigned to each possible category by satisfied rules

Possible outcome	M1-c1-R2	M1-c1-R3	M2-c2-R1	M2-c2-R3
0	0	0	0	0
Category I	0.85	0.75	0	0
Category II	0	0	0.90	0.85
Category I or II	0.15	0.25	0.10	0.15

Table 3. Basic belief masses assigned to each possible output

x	m(x)
0	0.9481
Category I	0.0144
Category II	0.0369
Category I or II	0.0006

Table 4. Pignistic probability (BetP) values

BetP(Category I)	0.2834
BetP(Category II)	0.7166

II. These pignistic probabilities *BetP* are computed based on the *bbm* values according to the TBM using Equation 8 from Appendix III as shown in Box 1.

As the result of these calculations, Table 4 was created. Based on the values from this table, it can be said that in this particular case, a prediction is as follows: the new data point belongs to *Category I* with the belief of 0.2834, and to the *Category II* with the belief of 0.7166.

SOFTWARE ENGINEERING MAINTENANCE DATA

The process of development of base-level prediction models, extraction of knowledge, and construction of the ensemble-based prediction system is illustrated using the dataset from the Naval Research Laboratories (The Data and Analysis Centre for Software). The details regarding the origin of the data and their attributes are presented next. For the experimental purposes, the dataset has been divided into three sets: a training set containing data points that are used to develop base-level data models, a validation set containing data that are used to validate confidence in rules extracted from the models, and a testing set that contains data points used to evaluate prediction capabilities of the models.[3]

The data were collected at the Naval Research Laboratories (NRL) to evaluate development methodologies used on the project called The Architecture Research Facility (ARF). This project was to aid the rapid simulation of different computer architectures for research and evaluation purposes.

The original dataset consists of 117 defect reports dealing with 143 defects, which were isolated and corrected during ARF development. The dataset also includes 253 records describing each component in the project. For the purposes of the chapter, the original dataset is pre-processed. The main emphasis is put on defect elimination activates. In particular, two aspects are of interest: a number of components that have to be examined during elimination of a defect; and the effort needed to correct a defect.

Box 1. Equation 8 from Appendix III

$$BetP(CatI) = \frac{m(CatI) + \frac{m(CatIorII)}{2}}{1 - m(0)} = \frac{0.0144 + \frac{0.0006}{2}}{1 - 0.9481} = 0.2834$$

$$BetP(CatII) = \frac{m(CatII) + \frac{m(CatIorII)}{2}}{1 - m(0)} = \frac{0.0369 + \frac{0.0006}{2}}{1 - 0.9481} = 0.7166$$

The modified dataset contains only data related to components with defects. There are 129 data points. Table 5 contains a few examples of these data points. The data points have been divided into three sets: a training set of 86 data points, a validation set containing 20 data points, and a testing set which has 23 data points.

The number of components that have to be examined during elimination of a defect is the first aspect of maintenance activities investigated in the chapter. The attributes of data points used in this case are represented in Table 6. There are two groups of attributes in the INPUT set. The first one represents types of defects of interest and development phases during which these defects have been introduced into a system. The second set contains attributes describing software components such as line of codes, number of comments, number of preprocessor statements, as well as subjective complexity and functionality. The OUTPUT is *Number of Components Examined*.

The second point of interest is related directly to a process of elimination of defects. In this case, the attention is put on the time needed to remove a single defect. The attributes of data points used in this case are shown in Table 7. As it can be seen, the attributes chosen here are almost identical to the ones from the previous investigation. The only difference is that the attribute *Number of Examined Components* has been added to the first group of INPUT attributes. This time, the OUTPUT attribute is *Effort needed to Eliminate a Defect*.

Table 5. Examples of data points

Type of defect	Phase of Entering	Lines of Code	No. of Comm.	Preproc. Statements	Subjective Complexity	Funct.	No. of Exam. Comp.	Elimination Time
clerical	code test	88	29	6	easy	error handling	single	<1 hour, 1day>
language	code test	83	42	2	easy	control	single	<1 hour, 1day>
clerical	code test	40	14	2	easy	control	single	< 1 hour
single design	code test	24	16	2	easy	control	single	<1 hour, 1day>
multiple	design	56	37	6	moderate	control	multiple	<1 hour, 1day>
multiple design	design	269	165	4	hard	computational	single	<1 hour, 1day>
multiple	code test	105	63	8	moderate	control	multiple	> 1 day
language	code test	188	116	3	hard	computational	multiple	<1 hour, 1day>
language	code test	168	103	5	moderate	computational	single	<1 hour

Table 6. List of attributes of the NRL data—Set I

INPUT	
type of defect	requirements_incorrect, functional_spec_incorrect, single_design_error, multiple_design_error, language_error, clerical_error, multiple_errors, other
phase when defect entered system	requirements_definition, functional_specification, design, code_testing
lines of code	<0, 635>
no. of comments	<0, 360>
no. of preprocessor statements	<0, 42>
subjective complexity	easy, moderate, hard
functionality	computational, control, data_processing, error_handling
OUTPUT	
no. of components examined	single component multiple components

Table 7. List of attributes of the NRL data—Set II

INPUT	
type of defect	requirements_incorrect, functional_spec_incorrect, single_design_error, multiple_design_error, language_error, clerical_error, multiple_errors, other
phase when defect entered system	requirements_definition, functional_specification, design, code_testing
no. of components examined	single component, multiple components
lines of code	<0, 635>
no. of comments	<0, 360>
no. of preprocessor statements	<0, 42>
subjective complexity	easy, moderate, hard
functionality	computational, control, data_processing, error_handling
OUTPUT	
time needed to eliminate a defect	less than 1 hour between 1 hour and 1 day more than 1 day

BASE-LEVEL DATA MODELS, EXTRACTED KNOWLEDGE AND CONFIDENCE IN RESULTS

The first step in data analysis and modeling is dedicated to the development of base-level data models that can be used for prediction purposes. The base-level models used here are rule-based models (see the *Rule-Based Models and Ensemble System* section). Three methods of building these models are selected in order to illustrate the idea of utilization of very different model development techniques. The first rule-based model is generated using See5/C5.0 (RuleQuest). The See5/C5 tool is an updated version of the well-known C4.5 algorithm (Quinlan, 1993). It has the capability of generating classifiers that are expressed as decision trees or sets of IF-THEN rules. The commercially-available tool called 4cRuleBuilder (4cData) is used to build the second data model. This tool uses supervised learning techniques to generate a data model from discrete numerical or nominal data, and has built-in discretization schemes for continuous attributes. Overall, 4cRuleBuilder generates compact rules that use a small number of selectors. The third model is a set of rules extracted from a decision tree constructed using the evolutionary-based technique GAGP described in Appendix II. In general, different model development techniques can be used to build rule-based data models which become elements of the proposed prediction system.

The following step in the analysis of data focuses on the evaluation of IF-THEN rules that represent relationships between attributes of software components and different defect measures: a number of components examined to remove a defect and the time needed to eliminate a defect. At that point, the goodness of rules is estimated via monitoring and validation of prediction capabilities of the rules (see the *Development and Validation of the IF-THEN Models* sub-section).

Number of Components Examined During Elimination of Defect

First, the issue of a number of components that has to be examined in order to eliminate a defect is investigated. Three base-level prediction models have provided the prediction rates that are presented in Table 8. A visual inspection of decision trees (for See5/C5.0 and GAGP) and rules (for 4cRuleBuilder) has indicated existence of terminal nodes of trees and rules for both categories, *single*

Table 8. NRL data prediction models: Prediction rates for number of examined components

Model	Prediction Rate
See5/C5.0	65.22%
4cRuleBuilder	60.87%
GAGP	56.52%

component examined and *multiple components examined*. The model generated by See5/C5.0 tool provides the highest prediction rate.[4]

Validation Process

Eleven IF-THEN rules are extracted from the decision tree generated by See5C5.0—four rules for the category *single component examined* and seven for *multiple components examined*. The application of 4cRuleBuilder tool results in six rules for *single component examined*, and five rules for *multiple components examined*. GAGP method leads to the generation of 21 rules: 14 for *single component examined* and seven for *multiple components examined*. In total, 43 rules are generated. Right now, the task is to select a relatively small set of rules that are the most significant and valuable for maintainers. Following the idea of goodness of rules (see the *Development and Validation of the IF-THEN Models* sub-section), the bbm_{UPDATE} values are used to "rank" all the rules and select the best ones.

Table 9. Original and updated bbm values

rule	bbm_T	bbm_{UPDATE}
See5/C5.0		
NC_See5_c1_1	**0.791**	**0.833**
NC_See5_c1_2	**0.770**	**0.750**
NC_See5_c2_1	**0.833**	**0.883**
NC_See5_c2_2	**0.800**	**0.889**
4cRuleBuilder		
NC_4cRB_c1_1	**0.950**	0.950
NC_4cRB_c1_2	0.929	**0.985**
NC_4cRB_c1_5	**0.941**	**0.988**
NC_4cRB_c2_1	0.818	**0.900**
NC_4cRB_c2_2	**0.898**	0.898
NC_4cRB_c2_3	**0.857**	**0.923**
GAGP		
NC_GAGP_c1_2	**0.909**	**0.952**
NC_GAGP_c1_9	**0.889**	**0.941**
NC_GAGP_c2_1	**0.800**	**0.889**
NC_GAGP_c2_3	**0.857**	0.857
NC_GAGP_c2_7	0.765	**0.907**

The *bbm* values assigned to the rules are presented in Table 9. The table contains only the rules that are best after the development stage (with the highest values of bbm_T) and the ones that are the best after the validation stage (with the highest value of bbm_{UPDATE}).

The rules generated by See5/C5.0 that are the best after the development process are also the best after the validation process (see bold entries in Table 9). For the rules NC_See5_c1_1[5] and NC_See5_c_2, their *bbm* values have increased after the validation phase. As it can be seen on the basis of other rules, none of the updated *bbm* values is smaller that the original one (there are some cases where *bbm* values do not change—this means that a rule has not been satisfied by any validation data point).

The same validation process has been performed on rules generated by 4cRuleBuilder. In the case of two rules NC_4cRB_c1_1 and NC_4cRB_c2_3, their *bbm* values have increased and the rules stay among the best ones. For the other two rules—NC_4cRB_c1_1 and NC_4cRB_c2_2—their *bbm* values have not changed. However, the *bbm* values of the other two rules NC_4cRB_c1_2 and NC_4cRB_c2_1 have increased and have become higher than the *bbm* values of NC_4cRB_c1_1 and NC_4cRB_c2_2. As the result, a set of best rules has been modified.

The validation process performed on rules generated by GAGP method has brought a single change in the category—*multiple components examined* (see Table 9). The rule NC_GAGP_c2_3 has been replaced by NC_GAGP_c2_7, which has the higher value of *bbm*. The *bbm* values of the rules from the category *single component examined* have improved.

Analysis of Best Rules

For the category *single component examined*, the range of bbm_{UPDATE} values is from 0.750 to 0.988. The rules with highest bbm_{UPDATE} values are generated by 4cRuleBuilder: NC_4cRB_c1_2 with bbm_{UPDATE} of 0.985, and NC_4cRB_c1_5 with bbm_{UPDATE} of 0.988. The rules NC_See5_c1_1 and NC_See5_c1_2, generated by See5/C5.0, have the lowest bbm_{UPDATE} values among three sets.

An interesting observation can be made when all the rules from the category *single component examined* are compared. Rules NC_GAGP_c1_9 and NC_4cRB_c1_5 are similar (see the following rules). It looks like both of them use the attributes *defect type* and *functionality* to predict a single component examination. The set of values of the attribute *defect type* from both rules are overlapping. In the case of the attribute *functionality*, the values used by the GAGP generated rule are a subset of the values used by the rule generated by 4cRuleBuilder. Such similarity and high bbm_{UPDATE} values of these rules indicate that both of them can be used to support prediction for a single component examination. The conclusion which can be derived here is: if *defect type* is *Clerical_Error* or *Requirements_Incorrect* and *functionality* of components with these defects is *Control* or *Error_Handling* then there is a need to examine just one component to be able to understand an impact of these defects and remove them.

Rule: NC_4cRB_c1_5:

if DEFECT_TYPE is *Multiple_Design_Error* or *Clerical_Error* or *Functional_Spec_Incorrect* **&**
FUNCTIONALITY is *Error_Handling* or *Control* or *Data_Accessing*
then SINGLE COMPONENT is examined

Rule: NC_4cRB_c1_9:

if DEFECT_TYPE is *Clerical_Error* or *Requirements_Incorrect* **&**
FUNCTIONALITY is *Error_Handling* or *Control*
then SINGLE COMPONENT is examined

Another pair of rules that draws attention is the pair containing the rules NC_4cRB_c1_2 and NC_GAGP_c1_2 (see the following rules). The rule generated by GAGP method uses a subset of values of the attribute *defect type* that are used by the rule NC_4cRB_c1_2. Also in the case of the attribute *lines of code*, there is some substantial overlapping. Overall, the rule generated by 4cRuleBuilder seems more specific. Once again, high values of bbm_{UPDATE} make these rules good candidates for prediction activities.

Rule NC_4cRB_c1_2:

if DEFECT_TYPE is *Single_Design_Error* or *Multiple_Design_Error* or *Clerical_Error* or *Language_Use_Error* or *Functional_Spec._Incorrect* or *Multiple_Error* **&**
PHASE WHEN ERROR ENTERD SYSTEM is *Requirements_Definition* or *Functional_Specification* or *Code_Testing* **&**
LINES OF CODE is less than 85 or in the range <108, 167> or more than 207 **&**
NUMBER OF COMMENTS is less than 29 or in the range <45, 103> or more than 128 **&**
NUMBER OF PREPROCESSOR STATEMENTS is less than 9
then SINGLE COMPONENT is examined

Rule NC_GAGP_c1_2:

if DEFECT_TYPE is *Single_Design_Error* or *Multiple_Design_Error* or *Multiple_Error* **&**
LINES OF CODE is less than 141 **&**
COMPLEXITY is *Easy*
then SINGLE COMPONENT is examined

From rules generated by See5/C5.0, only one rule—NC_See5_c1_1—is attractive. It is a very simple rule with relatively high value of $bbm_{UPDATE,}$ and a high coverage—more than 30 data points satisfied both the antecedent and the consequential parts of this rule. It is also a very intuitive rule. Definitely it is worth considering during a prediction process.

Rule NC_See5_c1_1:

if COMPLEXITY is *Easy*
then SINGLE COMPONENT is examined

In the case of the category *multiple components examined*, the rules have bbm_{UPDATE} values in the range: from 0.833 up to 0.923. Smaller maximum indicates that it is difficult to find a dominating rule describing a relationship among attributes of software components and a need for multiple components examination. The inspection of the rules points only to a single pair of rules NC_See5_c2_2 and NC_GAGP_c2_2 (see the following rules). There is a perfect much in the case of the attributes *defect type* and *complexity*. There is also some overlap existing for the attribute *lines of codes*. Based on that and high values of bbm_{UPDATE}, this pair can be treated as plausible rules describing a relationship among software attributes and a number of components which should be examined to fully understand an impact the defect *Functional_Spec._Incorrect* makes.

Rule NC_See5_c2_2:

if DEFECT_TYPE is *Functional_Spec._Incorrect* &
LINES OF CODE is more than 134 &
COMPLEXITY is *Easy*
then **MULTIPLE COMPONENTS are examined**

Rule NC_ GAGP_c2_2:

if DEFECT_TYPE is *Functional_Spec._Incorrect* &
LINES OF CODE is in the range <141, 481> &
COMPLEXITY is *Easy* &
FUNCTIONALITY is *Computational*
then **MULTIPLE COMPONENTS are examined**

Time Needed to Eliminate a Defect

The prediction rates for the models representing effort needed to eliminate a defect are presented in Table 10. In this case, the highest prediction rate is obtained using the model generated by GAGP method.

Table 10. NRL data prediction models: Prediction rates for defect removal time

Model	Prediction Rate
See5/C5.0	56.52%
4cRuleBuilder	56.52%
GAGP	65.22%

Validation Process

Prediction models developed for the second NRL dataset leads to the extraction of 49 rules. Validation of rules generated by See5/C5.0 has brought only one change (see Table 11). The rule ET_See5_c2_2 has been replaced by the rule ET_See5_c2_4.

The bbm_{UPDATE} value of the rule ET_See5_c2_4 increased and suppressed the bbm_{UPDATE} value of the rule ET_See5_c2_2. A similar situation has occurred in the case of rules generated by 4cRuleBuilder. Only one rule has been replaced by a new rule. In this case, however, it has happened in the category *effort for elimination of a defect less than one hour*. The rule ET_4cRB_c1_1 has been replaced by ET_4cRB_c1_6. Validation of the original set of rules generated by GAGP method has also brought a single change. In the

Table 11. Original and updated bbm values

rule	bbm_T	bbm_{UPDATE}
See5/C5.0		
ET_See5_c1_1	**0.938**	**0.974**
ET_See5_c1_2	**0.667**	**0.800**
ET_See5_c2_1	**0.875**	**0.875**
ET_See5_c2_2	**0.833**	0.833
ET_See5_c2_4	0.727	**0.842**
ET_See5_c3_1	**0.556**	**0.714**
4cRuleBuilder		
ET_4cRB_c1_1	**0.900**	0.844
ET_4cRB_c1_2	**0.944**	**0.966**
ET_4cRB_c1_6	0.857	**0.968**
ET_4cRB_c2_1	**0.882**	**0.938**
ET_4cRB_c2_9	**0.900**	**0.947**
ET_4cRB_c3_1	**0.500**	**0.500**
GAGP		
ET_GAGP_c1_3	**0.889**	**0.970**
ET_GAGP_c1_7	**0.889**	**0.889**
ET_GAGP_c2_1	**0.667**	**0.833**
ET_GAGP_c2_4	**0.833**	**0.714**
ET_GAGP_c2_6	**0.875**	**0.875**

case of the category *effort for elimination of a defect more than one hour but less than one day,* the bbm_{UPDATE} value of the rule ET_GAGP_c2_4 has decreased. At the same time, the bbm_{UPDATE} value of the rule ET_GABP_c2_1 has increased from 0.666 to 0.833, and this rule has become one of the best rules.

Analysis of Best Rules

As in the case of the previous experiment, let's examine the best IF-THEN rules (marked by bold fonts in Table 11). For the category *effort for elimination of a defect less than one hour* the range of bbm_{UPDATE} values is from 0.800 to 0.974. The examination of the rules from this category points to three rules ET_See5_c1_1, ET_GAGP_c1_3, and ET_GAGP_c1_7 (see the following rules). High values of bbm_{UPDATE} and similarity of the rules mean that they can be summed up with a very simple statement: if *defect type* is *Clerical_Error* then *time need to correct this defect is less than one hour.*

Rule ET_See5_c1_1:

if DEFECT TYPE is *Clerical_Error*
then DEFECT ELIMINATION TIME is less than 1 hour

Rule ET_GAGP_c1_3:

if DEFECT TYPE is *Clerical_Error* &
FUNCTIONALITY is *Computational* or *Data_Accessing*
then DEFECT ELIMINATION TIME is less than 1 hour

Rule ET_GAGP_c1_7:

if DEFECT TYPE is *Clerical_Error* &
FUNCTIONALITY is *Control*
then DEFECT ELIMINATION TIME is less than 1 hour

In the category *effort for elimination of a defect more than one hour but less than one day* the rules generated by 4cRuleBuilder have the highest values of bbm_{UPDATE}. A rule ET_4cRB_c2_1 (see the following rules) seems to be a very important one. Two other rules—ET_See5_c2_4 generated by See5/C5.0 and ET_GAGP_c2_6 generated by GAGP method—are very similar. The rule ET_See5_c2_4 overlaps with ET_4cRB_c2_1 in the case of the attributes—*defect type* and *complexity.* The rule ET_GAGP_c2_6 overlaps in the case of the attributes—*defect type* and *functionality.* All this and the high bbm_{UPDATE} values indicate that these three rules can be useful in prediction of time needed to eliminate defects.

Rule ET_4cRB_c2_1:

if DEFECT TYPE is *Multiple_Design_Error* or *Language_Use_Error* or *Functional_Spec._Incorrect* or *Other_Error* **&**
NUMBER OF COMMENTS in less than 130 or more than 181 **&**
NUMBER OF PREPROCESSOR STATEMENTS is less than 5 or more than 10 **&**
COMPLEXITY is Easy or Moderate **&**
FUNCTIONALITY is Computational or Data_Accessing or Error_Handling
then DEFECT ELIMINATION TIME is more than 1 hour but less than 1 day

Rule ET_See5_c2_4:

if DEFECT TYPE is *Language_Use_Error* **&**
COMPLEXITY is *Easy*
then DEFECT ELIMINATION TIME is more than 1 hour but less than 1 day

Rule ET_GAGP_c2_6:

if DEFECT_TYPE is *Single_Design_Error* or *Other_Error* **&**
FUNCTIONALITY is *Computational* or *Data_Accessing*
then DEFECT ELIMINATION TIME is more than 1 hour but less than 1 day

The category *effort for elimination of a defect more than one day* is very poorly represented in the generated sets of rules. A possible reason for that is a very non-uniform distribution of data points among three categories. The category *effort for elimination of a defect more than one day* is represented by only three per cent of data points. The rule generated by See5/C5.0 is the best. Its bbm_{UPDATE} value is 0.714.

Rule ET_See5_c3_1:
if NUMBER_OF_COMPONENTS_EXAMINED is *Multiple* **&**
COMPLEXITY is *Hard*
then DEFECT ELIMINATION TIME is more than 1 day

ENSEMBLE-BASED PREDICTION SYSTEM

The IF-THEN rules have been extracted from the base-level data models (see the *Base-Level Data Models, Extracted Knowledge and Confidence in Results* section). An important aspect of this extraction is related to the concept of rules' goodness. As the result, all rules have been evaluated and confidence measures (*bbm* values) have been assigned to them. The next step is to use this and build a system that combines different rules and takes these confidence measures into account. The results of this task are presented in the following text.

Number of Components Examined

The ensemble-based prediction system for predicting the number of components to be examined during defect removal is constructed. It is tested on all data points from the testing set. Two aspects of the results are important—a prediction rate and confidence measures for the obtained predictions.

The testing set contains 23 data points. Using only a single set of rules generated by See5/C5.0, 4cRuleBuilder, and GAGP, the prediction rates are 65.22%, 60.87%, and 56.52%, respectively (see Table 8). In the case of the ensemble-based prediction systems, the prediction rate has increased. The system that uses voting as the fusion method provides the prediction rate of 69.57%, and the system proposed here, that uses the elements of evidence theory, has the prediction rate of **73.91**% (this translates to 17 proper predictions). Among all testing data points, seven data points satisfy rules that indicate contradicting predictions. The rest of the data points are uniquely identified as belonging to a single category (most data points satisfy at least a single rule from each model, however some points satisfy only rules coming from two models). In all these cases, the pignistic probability BetP that a given data point belongs to a single category is very high at 0.99. More interesting is the case of seven data points that have led to non-unanimous prediction results. The rules that are satisfied by these seven points are presented in Table 12.

It can be easily observed that each of these points is not univocally classified to a single category. Rules generated by See5/C5.0 have problems with data points no. 1 (two rules of different categories fired), 4 (wrong rule fired), and 7 (two rules of different categories fired). Rules generated by 4cRuleBuilder are misled for points no. 2 (two rules of different categories fired), 3

Table 12. Rules satisfied by the seven "confusing" data points

	See5/C5.0	4cRuleBuilder	GAGP	Original Category	Predicted Category
1	c1_4 c2_4	c1_3	c1_10	I	I
2	c1_4	c1_5 c2_2	c1_4	I	I
3	c1_4		c2_3	I	II
4	c1_4	c2_1	c1_13	II	II
5	c1_1	c2_1	c1_12	I	I
6	c2_7	c1_2 c2_1 c1_6	c2_7	II	II
7	c1_4 c2_5	c2_3	c2_3	II	II

Table 13. BetP values for the seven "confusing" data points

Data points	1	2	3	4	5	6	7
BetP (cat. I: single comp. examined)	0.892	0.993	0.207	0.391	0.843	0.014	0.005
BetP (cat. II: multiple comp. examined)	0.108	0.007	0.793	0.609	0.157	0.986	0.995

(no rules fired), 5 (wrong rule fired), and 6 (two rules of different categories fired). In the case of rules generated by GAGP points no. 3 (wrong rule fired) and 4 (wrong rule fired) are wrongly predicted. The BetP values obtained for these seven points are shown in Table 13.

Special attention should be paid to the data point no. 4. Rules generated by See5/C5.0 (NC_See5_c1_4 with bbm_{UPDATE} of 0.545) and GAGP (NC_GAGP_c1_13 with bbm_{UPDATE} of 0.667) indicate that the point should be classified as category I. However, the rule NC_4cRB_c2_1, which is one of the best 4cRuleBuilder rules and has bbm_{UPDATE} value of 0.900, has a strong influence on the overall result.

Time Needed to Eliminate a Defect

The ensemble-based prediction system is also built for predicting time needed to remove a single defect. Prediction rates and the confidence measures for obtained predictions are recorded. The testing set contains 23 data points. Using only a single set of rules generated by See5/C5.0, 4cRuleBuilder, and GAGP, the prediction rates are 56.52%, 56.52%, and 65.22%, respectively (see Table 10). The prediction rate increases to **78.26**% (this translates to 18 proper predictions) for the proposed ensemble-based prediction system that uses basic belief masses. The ensemble-based system with voting has the prediction rate of

Table 14. Rules satisfied by the 12 "confusing" data points

	See5/C5.0	4cRuleBuilder	GAGP	Original Category	Predicted Category
1	c1_3 c2_6		c1_6	I	I
2	c1_3 c2_6	c2_3	c2_2	III	II
3	c2_6	c1_6	c2_2	III	II
4	c1_3 c2_1		c2_9	I	I
5	c2_4	c2_2 c3_1 c2_5 c2_6	c2_2	II	II
6	c3_1	c1_4 c2_2 c3_1	c2_3	III	III
7	c1_3 c2_6	c1_1 c2_4	c2_1	I	II
8	c2_8	c2_2 c3_1		II	II
9	c1_3 c2_2	c2_3 c2_7		II	II
10	c1_1 c1_3	c1_2 c2_5 c1_3	c1_3	I	I
11	c1_3 c2_4	c1_1		II	I
12	c1_1	c1_2 c2_5	c1_3	I	I

69.57%. The analysis of the proposed prediction process indicates that 12 data points satisfy rules that indicate contradicting predictions. The remaining 11 data points are unanimously predicted by the rules of the system. Confidence measures for these predictions are around 0.99. The rules fired for the 12 "confusing" points are presented in Table 14.

It can be easily observed that each of these points is not univocally classified to a single category. Rules generated by See5/C5.0 have problems with data points no. 1 (two rules of different categories fired), 2 (two rules of different categories fired), 3 (wrong rule fired), 4 (two rules of different categories fired), 7 (two rules of different categories fired), 9 (two rules of different categories fired), and 11 (two rules of different categories fired). Rules generated by 4cRuleBuilder are misled for points no. 1 (no rules fired), 2 (wrong rule fired), 4 (no rules fired), 5 (two rules of different categories fired), 6 (three rules of different categories fired), 7 (two rules of different categories fired), 8 (two rules of different categories fired), 10 (two rules of different categories fired), 11 (wrong rule fired), and 12 (two rules of different categories fired). In the case of the rules generated by GAGP, the points no. 2, 3, 6, 7, and 11 are wrongly predicted due to firing of the wrong rule. The values of the pignistic probabilities BetP calculated by TBM are included in Tables 15 and 16.

The most interesting are data points no. 3, 6, and 7. In the case of the point number 3, the rules generated by See5/C5.0 and GAGP indicate that this point should belong to the category I, however a higher bbm_{UPDATE} value of the 4cRuleBuilder rule (comparing to bbm_{UPDATE} of See5/C5.0 and GAGP rules) has changed the verdict of the prediction system. It can be seen that the probability that the point belongs to the category II is relatively high—0.878. The point number 6, on the other hand, satisfies one rule of the category I, two rules of the category II, and two rules of the category III. The prediction system classifies this data point to a proper category. However, in the case of BetP probabilities, it can be seen that BetP(category III) is only slightly larger than BetP(category II). In the case of the point no. 7, a high bbm_{UPDATE} value of GAGP rule ET_GAGP_c2_1 (supported by bbm_{UPDATE} values of ET_See5_c2_6 and ET_4cRB_c2_4) has caused false prediction; the rules ET_See5_c1_3 and ET_4cRB_c1_1 have bbm_{UPDATE} values too small to increase the value of the probability BetP for Category I.

Table 15. BetP values for the 12 "confusing" data points: Points 1 to 6

Data points	**1**	**2**	**3**	**4**	**5**	**6**
BetP (cat. I: time less than 1 hour)	0.638	0.031	0.599	0.055	0.001	0.108
BetP (cat. II: time between 1 hour and 1 day)	0.308	0.962	0.381	0.932	0.996	0.434
BetP (cat. III: time more than 1 day)	0.054	0.007	0.020	0.013	0.003	0.458

Table 16. BetP values for the 12 "confusing" data points: Points 7 to 12

Data points	**7**	**8**	**9**	**10**	**11**	**12**
BetP (cat. I: time less than 1 hour)	0.119	0.014	0.002	0.999	0.673	0.999
BetP (cat. II: time between 1 hour and 1 day)	0.878	0.913	0.998	0.001	0.309	0.001
BetP (cat. III: time more than 1 day)	0.003	0.055	0.000	0.000	0.018	0.000

CONCLUSION

In this chapter, an intelligent method of analysis of software engineering data is illustrated. It applies evolutionary computation methods for development of models, and elements of evidence theory to combine base-level models in order to create an ensemble-based prediction system that has capabilities to determine confidence measures for the generated predictions. Application of a number of rule-based models developed with different techniques enhances prediction capabilities and allows for more comprehensive extraction of knowledge from the data. Confidence measures associated with predictions and extracted knowledge help to better understand the relationships between attributes of data and the mechanisms leading to generation of predictions.

Proposed method is applied to analysis of software corrective maintenance data. The purpose of this analysis is to find and understand relationships between attributes of software, types of defects, and efforts needed to conduct defect elimination tasks. Special attention is put on two aspects of defect elimination process: a number of software components that have to be examined to remove a single defect and a total time needed to remove a defect. The prediction systems, extracted knowledge, and results of prediction processes are described and investigated.

ACKNOWLEDGMENT

The author would like to acknowledge the support of the Alberta Software Engineering Research Consortium (ASERC) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

4cData. Data mining and knowledge discovery tools and services. Retrieved September 25, 2006, from http://www.4cdata.net

Ali, K. (1995). *A comparison of methods for learning and combining evidence from multiple models.* Technical Report 95-47. Dept. of Information and Computer Science, University of California, Irvine.

Ali, K., & Pazzani, M. (1996). Error reduction through learning multiple descriptions. *Machine Learning, 24,* 173-202.

Amasaki, S., Takagi, Y., Mizuno, O., & Kikuno, T. (2003). A Bayesian belief network for assessing the likelihood of fault content. *14th International Symposium on Software Reliability Engineering,* Denver, Colorado, USA, November 17-20 (pp. 215-226). Washington, D.C.: IEEE Computer Society.

Back, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (2000). *Evolutionary computations I.* Bristol, UK: Institute of Physics Publishing.

Berkovich, Y. (2000). *Software quality prediction using case-based reasoning.* PhD thesis. Boca Raton, FL: Florida Atlantic University.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24,* 123-140.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

The Data and Analysis Centre for Software. Retrieved September 25, 2006, from http://www.thedacs.com

Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of Royal Statistical Society, Series B*(30), 205-247.

Evanco, W. M. (1999). Analyzing change effort in software during development. In *Proceedings of Sixth International Software Metrics Symposium (METRICS'99),* Boca Raton, FL, November 4-6 (pp. 179-188). Washington, D.C.: IEEE Computer Society.

Evanco, W. M. (2001). Prediction models for software fault correction effort. *Proceedings of the Fifth Conference on Software Maintenance and Reengineering, CSMR 2001,* Lisbon, Portugal, March 14-16 (pp. 114-120). Washington, D.C.: IEEE Computer Society.

Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions of Software Engineering, 25*(5), 675-689.

Fenton, N. E., & Neil, M. (2005). A critique of software defect prediction models. In D. Zhang, & J. J. P. Tsai (Eds.), *Machine learning applications in software engineering* (pp. 72-86). Singapore: World Scientific Publishing Co.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *13th International Conference on Machine Learning,* Bari, Italy, July 3-6 (pp. 148-156). San Francisco, CA: Morgan Kaufmann.

Glass, R. (1989). Software maintenance documentation. *Annual ACM Conference on Systems Documentation,* Pittsburgh, PA, November 8-10 (pp. 18-23). New York: ACM Press.

Gokhale, S. S., & Lyu., M. R. (1997). Regression tree modeling for the prediction of software quality. In the *Proceedings of the Third International Conference on Reliability and Quality of Design,* Anaheim, CA, USA, March 12-14 (pp. 31-36). New Brunswick, NJ: International Society of Science and Applied Technology.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Hayes, J. H., Patel, S. C., & Zhao, L. (2004). A metrics-based software maintenance effort model. In the *Proceedings of the Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'04),* Tempere, Finland, March 24-26 (pp. 254-260). Washington, D.C.: IEEE Computer Society.

Holland, J. H. (1972). *Adaptation in natural and artificial systems* (2nd ed.). Cambridge, MA: MIT Press.

IEEE 610.12. *IEEE Standard Glossary of Software Engineering Terminology.*

Jorgensen, M. (1995). Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions of Software Engineering, 21*(8), 674-681.

Khoshgoftaar, T. M., & Allen, E. B. (1998). Neural networks for software quality prediction. In W. Pedrycz, & J. F. Peters (Eds.), *Computational intelligence in software engineering,* the Advances in Fuzzy Systems – Applications and Theory Series (pp. 33-63). Singapore: World Scientific.

Khoshgoftaar, T. M., Allen, E. B., Naik, A., Jones, W., & Hudepohl, J. (1998). Modeling software quality with classification trees. In the *Proceedings of the Fourth International Conference on Reliability and Quality of Design,* Seattle, August 12-14 (pp. 178-182). New Brunswick, NJ: International Society of Science and Applied Technology.

Khoshgoftaar, T. M., & Lanning, D. L. (1995). A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software, 29*(1), 85-91.

Khoshgoftaar, T. M., & Seliya, N. (2002). Tree-based software quality models for fault prediction. In the *Proceedings of Eight International Software Metrics Symposium*, Ottawa, Canada, June 4-7 (pp. 203-214). Los Alamitos, CA: IEEE Computer Society.

Khoshgoftaar, T. M., & Seliya, N. (2003). Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering, 8*, 255-283.

Kononenko, M., & Kovacic, M. (1992). Learning as optimization: Stochastic generation of multiple knowledge. *9th International Workshop on Machine Learning*, Aberdeen, UK, July 1-3 (pp. 257-262). San Francisco, CA: Morgan Kaufmann.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Kwok, S., & Carter, C. (1990). Multiple decision trees. *Uncertainty in Artificial Intelligence, 4*, 327-335.

Luger, G. (2002). *Artificial intelligence: Structures and strategies for complex problem-solving*. Reading, MA: Addison Wesley.

Mitchell, T. (1997). *Machine learning*. Boston, MA: McGraw Hill.

Neil, M., & Fenton, N. (1996). Predicting software quality using Bayesian belief networks. *21st Annual Software Engineering Workshop*, NASA/Goddard Space Flight Centre, Washington D.C., December 4-5 (pp. 217-230).

Neil, M., Krause, P., & Fenton, N. E. (2003). Software quality prediction using Bayesian networks. In T. M. Khoshgoftaar (Ed.), *Software engineering with computational intelligence* (Chapter 6). International Series in Engineering and Computers Science. Hingham, MA: Kluwer Academic Publishers.

Quinlan J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann Publishers.

Reformat, M., Pedrycz, W., & Pizzi, N. (2004). Building a software experience factory using granular-based models. *Fuzzy Sets and Systems, 145*(1), 111-140.

RuleQuest Research Data Mining Tools. Retrieved September 25, 2006, from http://www.rulequest.com

Savage, L. J. (1954). *Foundations of statistics*. New York: Wiley.

Schneidewind, N. F. (1995). Software metrics validation: Space shuttle flight software example. *Annals of Software Engineering, 1*, 287-309.

Schneidewind, N. F. (1997). Software metrics model for integrating quality control and prediction. In the *Proceedings of the 8th International Symposium on Software Reliability Engineering*, Albuquerque, NM, USA, November 2-5 (pp. 402-415). Washington, D.C.: IEEE Computer Society.

Shafer, G. (1976). *A mathematical theory of evidence*. Princeton, NJ: Princeton University Press.

Smets, P. (1988). Belief functions. In P. Smets, A. Mamdani, D. Dubois, & H. Prade (Eds.), *Non standard logics for automated reasoning* (pp. 253-286). London: Academic Press.

Smets, P. (1992). The concept of distinct evidence. *IPMU 92 Proceedings*, Palma de Mallorca, July 6-10, (pp. 253-286). *Lecture Notes in Computer Science, LNCS 682*. Berlin, Germany: Springer Verlag.

Smets, P., & Kennes, R. (1994). The transferable belief model. *Artificial Intelligence, 66*, 191-234.

Smith, D. (1999). *Designing maintainable software*. New York: Springer.

Takahashi, R., Muroaka, Y., & Nakamura, Y. (1997). Building software quality classification trees: Approach, experimentation, evaluation. In the *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, Albuquerque, New Mexico, USA, November 2-5 (pp. 222-233). Washington, D.C.: IEEE Computer Society.

Todorovski, L. & Dzeroski, S. (2000). Combining multiple models with meta decision trees. *4th European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 54-64). Berlin: Springer.

Troster, J., & Tian, J. (1995). Measurement and defect modeling for a legacy software system. *Annals of Software Engineering, 1*, 95-118.

Winer, B. J., Brown, D. R., & Michels, K. M. (1991). *Statistical principles in experimental design*. Boston, MA: McGraw-Hill.

Winston, P. H. (1992). *Artificial intelligence*. Reading, MA: Addison Wesley.

Yuan, X., Khoshgoftaar, T. M., Allen, E. B., & Ganesan, K. (2000). An application of fuzzy clustering to software quality prediction. In the *Proceedings of the Third IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)* (pp. 85-90).

ENDNOTES

1 CART stands for Classification And Regression Trees (Breiman, Friedman, Olshen, & Stone, 1984).

2 The proposed approach assumes the following split of data: 20% of data points are randomly selected to constitute a testing dataset, this set does not change for the time of the whole experiment, the data points from this set are used for evaluation of constructed ensemble-based prediction systems; the remaining 80% of data is randomly split into a training set (60%) and a validation set (20%).

3 The dataset is divided into training, validation, and testing sets to comply with the construction process of the ensemble-based prediction system (see the *Evidence Theory and Ensemble-Based System* section).

4 All results presented in the chapter are for testing data.

5 The following schema has been applied to identify generated rules: <two letters identifying dataset name>-<construction method>-<class number>-<rule number>.

APPENDIX I: EVOLUTIONARY COMPUTING

A variety of evolutionary algorithms (EAs), such as genetic algorithms (Goldberg, 1989) and genetic programming (Koza, 1992), have been successfully applied to numerous problems (Back, Fogel, & Michalewicz, 2000) both at the level of structural and parametric optimization. EAs are search methods utilizing the principles of natural selection and genetics (Holland, 1972). Concisely, EAs operate on a set of candidate solutions, called a population, to a given problem. The candidate solutions are evaluated based on their ability to solve the problem. The results of the evaluation are used in a process of forming a new set of solutions. The choice of individuals that are passed to the next population is performed in a process called selection. This process is based on "goodness" of candidate solutions. Additionally, genetic operators, that is, crossover and mutation, are employed to modify selected candidates. Such sequence of actions is repeated until some final criterion is fulfilled. The sequence of activities of EAs is presented in Figure 5.

Genetic Algorithms

Genetic algorithms (GAs) (Goldberg, 1989) are one of the most popular EAs. In the case of GAs, candidate solutions to a given problem are encoded into chromosome-like data structures named genotypes. The problem to be solved is encoded into a special function called a fitness function. This function embraces all requirements imposed on the solution to the problem. The genotypes, after being decoded into phenotypes which are solutions in the problem domain, constitute inputs to the fitness function. They are evaluated based on their ability to solve the problem. The results of the evaluation are used in a selection process based on favoring individuals with higher fitness values. The selection can be performed in numerous ways. One of them is called stochastic sampling with replacement. In this method, the entire population is mapped onto a roulette wheel where each individual is represented by the area corresponding to its fitness. Individuals of the next population are chosen by repetitive spinning of the roulette wheel.

Finally, the operations of crossover and mutation are performed on the individuals. Crossover allows exchange of information among individual genotypes in the population and provides innovative capability to the GAs. Its role is to explore the search space in a process of exchanging chromosome parts. Two randomly selected chromosomes are "cut" and "merge" to "produce" a new one. Two parent chromosomes are randomly selected, and two children are created by swapping sequences of genes. In its simplest form, a single crossover point is used. The percentage of individuals that go through crossover is "controlled" by the crossover probability pc.

The role of mutation is to introduce diversity into the population and promote a probabilistic-like traversing of the search space. Mutation builds a new chromosome by making an alteration to a single gene. All genes in all chromosomes are altered with the mutation probability pm.

The evolution of candidate solutions is halted when perfect fitness is achieved or some specified maximum number of generations is passed. The candidate solution that has the best fitness, after termination, is deemed as the discovered solution. Originally, only chromosomes with binary genes have been used. However, right now there are versions of GAs that use integer or floating-point numbers as genes.

Genetic Programming

Genetic programming (GP) (Koza, 1992) can be seen as an extension of the genetic paradigm into the area of programs. It means, that objects, which constitute population, are not fixed-length strings that encode candidate solutions to the given problem, but they are programs. In general, these programs are expressed as parse trees, rather than as lines of code.

In the case of GP, each candidate solution is built using two sets:

- the function set, $F = \{f_{1(b\_1)}; f_{2(b\_2)}; \ldots; f_{n(b\_n)}\}$, of functions with arity > 0; each function from F takes a specified number of arguments, defined as $b_1; b_2; \ldots b_n$;
- the terminal set $T = \{ t_1; t_2; \ldots; t_n \}$ of 0-arity functions or constants.

Sets F and T are defined a priori. The process of constructing candidate solutions starts by selecting a function, f_i, randomly from the set F. For each of the b_i arguments, this process is repeated where a random function or terminal may be selected to fill each argument position. If a terminal is selected, the generation process is completed for this branch of the function. If a function is selected, the generation process is recursively applied to each argument of this function. A user specifies a maximum depth of parse trees in order to limit the size of the candidates. For example, using the function set F = {AND; OR; NOT} and the terminal set T = {a; b; c; d}, two sample programs that could be created are shown in Figure 6.

Depending on the type of elements of the function and terminal sets, different structures can be built and different problems can be targeted. All stages of evolutionary algorithms—selection, crossover and mutation—are applied to all structures. Modifications of these structures are performed via manipulations on the lists (Koza, 1992).

Detailed descriptions of GP, GAs, or other evolutionary-based algorithms can be found in Back et al. (2000), Goldberg (1989), Holland (1972), and Koza (1992).

APPENDIX II: DECISION TRESS AS DATA MODELS AND THEIR EVOLUTIONARY-BASED CONSTRUCTION

Decision Trees

Decision trees are very attractive representations of underlying data relationships (Mitchell, 1997). They are very intuitive and relatively easy to understand by users. Their graphical representation provides a simple way of finding connections among data points and translating them into IF-THEN rules. The trees are very well suited to perform classification/prediction tasks.

Decision trees consist of a series of nodes with a top node called the root node. Each tree is built by means of a top down growth procedure. The building process is based on training data that contain data points (vectors) with a number of attributes. One of these attributes determines the category to which the data point belongs. At each node, called the attribute node, the training data is partitioned to the children nodes using a splitting rule (see Figure 7). A splitting rule can be of the form: if $V < c$ then s belongs to L, otherwise to R, where V is a selected attribute, c is a constant, s is the data sample and L and R are the left and right branches of the node. In this case, splitting is done using one attribute and a node has two branches (two children). In general, a node can have more branches and the splitting can be done based on several attributes. The best splitting for each node is searched based on a "purity" function calculated from the data. The data is considered pure when they contain data points from only one category. Most frequently used purity functions are entropy, gini-index, and twoing-rule. The data portion that falls into each children node is partitioned again in an effort to maximize the purity function. The tree is constructed until the purity of the data points in each leaf node is at a pre-defined level. Each leaf node is then labeled with a category determined based on majority rule: node is labeled to the category to which majority of the training data points at that node belong. Such nodes are called terminal nodes.

Evolutionary-Based Construction of Decision Trees

Concept

The approach used for construction of decision trees is based on combination of genetic algorithms (GAs) and genetic programming (GP). This approach allows for targeting two problems simultaneously—searching for the best splitting in attribute domains (discretization problem) and finding the most suitable attributes for splitting rules. The approach combines GAs' strength to search through numeric spaces together with GP's strength to search symbolic spaces. Moreover, a process of construction of decision trees—defining splitting and selecting node attributes—can be "controlled" by different functions which represent different objectives essential for a given classification/prediction process. In a nutshell, the approach is to construct a decision tree via performing two-level optimization: parametric using GAs and structural using GP.

A graphical illustration of the approach is shown in Figure 8. The GA is used at the higher level and its goal is to search for the best splitting of attributes' domains. The GA operates on a population of strings, where each string—GA chromosome—is a set of possible splittings for all attributes. Each GA chromosome is evaluated via a fitness function. Evaluation of each chromosome is performed by running

a GP optimization. This means that the GP searches for the best decision tree that can be constructed for the splitting represented by the GA chromosome. Consequently, the GP operates on a population of lists that are blueprints of decision trees. In other words, each individual of the GP population—a list—represents a single decision tree. The population of decision trees evolves according to the rules of selection and genetic operations of crossover and mutation. Each GP individual is evaluated by means of a certain fitness function. The GP returns the best value of the fitness function together with the best decision tree to the GA. The fitness value returned by the GP becomes the fitness value of the evaluated GA chromosome. Such approach ensures evaluation of each GA chromosome based on the best possible decision tree that can be constructed for the splitting represented by this chromosome.

To find the best solution, the GA creates and continuously modifies chromosomes—splittings and then, for each splitting, it invokes GP to generate the best decision tree. The following algorithm summarizes the steps that are used with the multi-level approach:

STEP 1: randomly generate an initial population of GA chromosomes (each chromosome contains information about a number of splittings and about splitting points for each attribute);
STEP 2: WHILE termination criterion not reached DO:
  STEP 2_1: evaluate each GA chromosome (splitting) by invoking GP
    STEP 2_1_1: randomly generate an initial population of trees for a given splitting;
    STEP 2_1_2: WHILE termination criterion not reached DO:
      STEP 2_1_2_1: calculate a performance measure (fitness value) representing how well each candidate solution performs the designated task, in the case when candidate solutions are decision trees it means classification/prediction of data points from a training dataset;
      STEP 2_1_2_2: select the decision trees for reproduction;
      STEP 2_1_2_3: modify decision trees using crossover and mutation;
    STEP 2_1_3: identify the fittest solution to the problem -- the best decision tree and return it to GA
  STEP 2_2: select candidate solutions—splittings—for reproduction to the next generation;
  STEP 2_3: apply genetic operators (crossover and mutation) to combine and modify splittings;
STEP 3: identify the fittest solution—the best splitting and associated with it the best decision tree.

Optimization Objective

One of the most important aspects of the approach is its flexibility to construct decision trees based on objectives that can reflect different requirements regarding the classification/prediction process and the different character of data. These objectives are represented by fitness functions. The role of the fitness function is to assess how well the decision tree classifies the training data. The simplest fitness function represents a single objective ensuring the highest classification/prediction rate without taking into consideration the classification/prediction rate for each data category. In such case, the fitness function is as follows:

$$Fit\_Fun_A = \frac{K}{N} \tag{4}$$

where N represents the number of data points in a training set, and K represents the number of correctly classified/predicted data points. Such fitness function gives good results when the numbers of data points in each category are comparable. In many cases, the character of processed data is such that not

all categories are represented equally. In this case, a fitness function should be such that it ensures the highest possible classification/classification rate for each category. An example of such fitness functions is presented in the following:

$$Fit\_Fun_B = \prod_{i=1}^{c} \frac{k_i + 1}{n_i} \tag{5}$$

where c represents a number of different categories, n_i represents a number of data samples that belong to a category i, and k_i is a number of correctly classified data points of a category i.

APPENDIX III: BASICS OF EVIDENCE THEORY

Let's start with a set of worlds Ω called the *frame of discernment.* One of the worlds, denoted ω_0, corresponds to the actual world. The agent (it may be a piece of software, system) does not know which world in Ω corresponds to the actual world ω_0. Nevertheless, the agent has some idea, some opinion about which world might be the actual one. So for every subset I of Ω, called the *focal element*, the agent can express the strength of its opinion that the actual world ω_0 belongs to *A.* This strength is denoted *bel(A)* and called a belief function (see the following text for the formal definition). Extreme values for *bel* denote full belief (1) or no belief at all (0). The larger *bel(A),* the stronger the agent believes that ω_0 belongs to *A.*

Basic Belief Assignments

One of the concepts of the theory is a basic belief assignment (*bba*). Related to belief function *bel*, one can define its so-called Moebius transform, denoted *m* and called a basic belief assignment. Let

$$m : 2^{\Omega} \rightarrow [0,1]$$

where *m(A)* is called the basic belief mass (*bbm*) given to $A \subseteq \Omega$. The value of *m(A)* represents belief that supports *A*—that is, the fact that the actual world ω_0 belongs to *A* without supporting any more specific subset. In the case, when ω_0 belongs to *A*, and nothing more is known about the value of ω_0, then some part of belief will be given to *A*, but no subset of *A* will get any positive support. In that case, *m(A) > 0* and *m(B)=0* for all $B \subseteq A$ and $B \neq A$.

Belief Functions

The *bbm m(A)* does not quantify belief, denoted *bel(A),* that the actual world ω_0 belongs to *A.* Indeed, the *bbm m(B)* given to any subset *B* of *A* also supports that ω_0 belongs to *A.* Hence, the belief *bel(A)* is obtained by summing all the *bbm m(B)* for $B \subseteq A$. At the end:

$$bel(B) = \sum_{\varnothing \neq B \subseteq A} m(B) \qquad \forall A \subseteq \Omega, A \neq \varnothing \qquad bel(\varnothing) = 0 \tag{6}$$

The belief function *bel* satisfies the following inequality:

$\forall n > 1, \forall A_1, A_2, ..., A_n \subseteq \Omega$:

$$bel(A_1 \cup A_2 \cup ... \cup A_n) \geq \sum_i bel(A_i) - \sum_{i>j} bel(A_i \cap A_j) ... - (1)^n bel(A_1 \cap A_2 \cap ... \cap A_n) \tag{7}$$

As such, the meaning of these inequalities is not obvious except when *n=2*. These inequalities generalize the idea that agent's belief that the actual world belongs to $A \subseteq \Omega$ can be larger than the sum of the beliefs the agent gives to the elements of a partition of *A*.

Combination of Two Belief Functions

Suppose there are two "distinct" pieces of evidence Ev1 and Ev2 produced by two sources of information. Let bel_1 and bel_2 be the belief functions induced by each piece of evidence. These two belief functions, with the focal elements A_i and B_j respectively, may be combined into a new belief function using Dempster's (1968) rule of combination. The rule specifies the combined belief mass, *m*, assigned to each focal element C_k, where *C* is the set of all subsets produced by *A* and *B*. The rule is:

$$m(C_k) = \frac{\sum_{A_i \cap B_j = C_k; C_k \neq \varnothing} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \varnothing} m(A_i)m(B_j)} \tag{8}$$

where the focal elements of $bel_1=A=\{A_1, A_2, \ldots, A_i\}$ and $bel_2=B=\{B_1, B_2, \ldots, B_j\}$. The combination of two belief functions is also know as taking the orthogonal sum, $\oplus$, and is written as:

$$bel_3 = bel_1 \oplus bel_2 = (m(C_1),...,m(C_k)) \tag{9}$$

The meaning of distinct for two pieces of evidence has been left undefined. It lacks rigorous definition. Intuitively, it means the absence of any relation. In this case, the belief function bel_2 induced by the second source is not influenced by the knowledge of the belief function bel_1 induced by the first source and vice versa (Smets, 1992).

Transferable Belief Model

Dempster-Shafer theory has been used to develop the transferable belief model (TBM) (Smets, 1994). The model represents quantified beliefs and is based on belief functions *bel*. The TBM departs from the classical Bayesian approach in that the additivity encountered in probability theory is not assumed. For instance, there is no assumption that *bel(A)=0* implies that $bel(\neg A) = 1$. In fact $bel(A) = bel(\neg A) = 0$ is even possible. The additivity property is replaced by inequalities like:

$$bel(A \cup B) \geq bel(A) + bel(B) - bel(A \cap B) \tag{10}$$

In the TBM, one assumes that bel is a capacity monotone of order infinite. Given a belief function bel, a probability function is generated that is used to make decision by maximizing expected utilities. It requires the construction of the betting frame, that is, a list of alternatives on which the bet must be made. Let BetFrame denotes the betting frame. The granularity of BetFrame is such that if by necessity two alternatives are not distinguishable from a consequence-utility point of view, than they are pooled into the same granule.

Once the betting frame is determined, the *bbms* are transformed by the so-called pignistic transformation into the pignistic probabilities $BetP : 2^{\Omega} \rightarrow [0,1]$ with:

$$BetP(A) = \sum_{X \subseteq Bet_{Frame}, X \neq \varnothing} \frac{m(X)}{1 - m(\varnothing)} \frac{\#(A \cap X)}{\#(X)} \tag{11}$$

for all

$$A \subseteq Bet_{Frame}, A \neq \varnothing$$

#(X) is the number of granules of the betting frame BetFrame in *X*, and *m()* is called the basic belief mass. By construction, the pignistic probability function *BetP* is a probability function, but it is qualified as pignistic to avoid the error that would consist in considering this probability function as someone's beliefs. Someone's beliefs are represented by *bel*, and *BetP* is just the additive measure needed to compute expected utilities when decision must be made (Savage, 1954).

This work was previously published in Advances in Machine Learning Applications in Software Engineering, edited by D. Zhang and J. Tsai, pp. 14-51, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 1.19
An Overview of Software Quality Concepts and Management Issues

Alain April
École de technologie supérieure, Québec, Canada

Claude Y. Laporte
École de technologie supérieure, Québec, Canada

ABSTRACT

This chapter introduces the generally accepted knowledge on software quality that has been included in the (SWEBOK) Software Engineering Body of Knowledge (ISOTR 19759, 2005). One chapter of the SWEBOK is dedicated to software quality (April et al., 2005). It argues that ethics play an important role in applying the quality models and the notions of cost of quality for software engineers. It also describes the minimal content required in a software quality assurance plan. Finally an overview of what to expect in the upcoming international standards on software quality requirements, which transcend the life cycle activities of all IT processes, is presented.

INTRODUCTION

The business value of a software product results from its quality as perceived by both acquirers and end users. Therefore, quality is increasingly seen as a critical attribute of software, since its absence results in financial loss as well as dissatisfied users, and may even endanger lives. For example, Therac-25, a computer-driven radiation system, seriously injured and killed patients by massive overdosing (Levenson & Turner, 1993). Improving recognition of the importance of setting software quality requirements and of assessing quality causes a shift in the "center of gravity" of software engineering from creating technology-centered solutions to satisfying stakeholders.

Software acquisition, development, maintenance, and operations organizations confronted with such a shift are, in general, not adequately equipped to deal with it. Until recently, they did not have at their disposal the quality models or measurement instruments to allow (or facilitate) the engineering of quality throughout the entire software product life cycle. The objective of software product quality engineering is to achieve the required quality of the product through the definition of quality requirements and their implementation, measurement of appropriate quality attributes, and evaluation of the resulting quality. The objective is, in fact, software product quality.

This chapter is structured in accordance with the SWEBOK classification of the software quality body of knowledge (www.swebok.org) shown in Figure 1.

SOFTWARE QUALITY FUNDAMENTALS

Agreement on quality requirements, as well as clear communication on what constitutes quality, require that the many aspects of quality be formally defined and discussed. Over the years, authors and organizations have defined the term "quality" differently. IBM used the phrase "market-driven quality", which is based on achieving total customer satisfaction. This definition was influenced by the total quality management approach of Phil Crosby (1979), who defined quality as "conformance to user requirements". Watts Humphrey (Humphrey, 1990), looking at quality in the software industry, defined it as "achieving excellent levels of fitness for use". More recently, quality has been defined in ISO 9001 (2000) as "the degree to which a set of inherent *characteristics* fulfills the *requirements*." The next section looks at how organizational culture and individual ethics play a role in quality in the organization.

Culture and Ethics

Edward B. Tylor (1871) defined human culture as "that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society." Culture guides the behaviors, activities,

Figure 1. Adapted breakdown of software quality topics (ISOTR 19759, 2005)

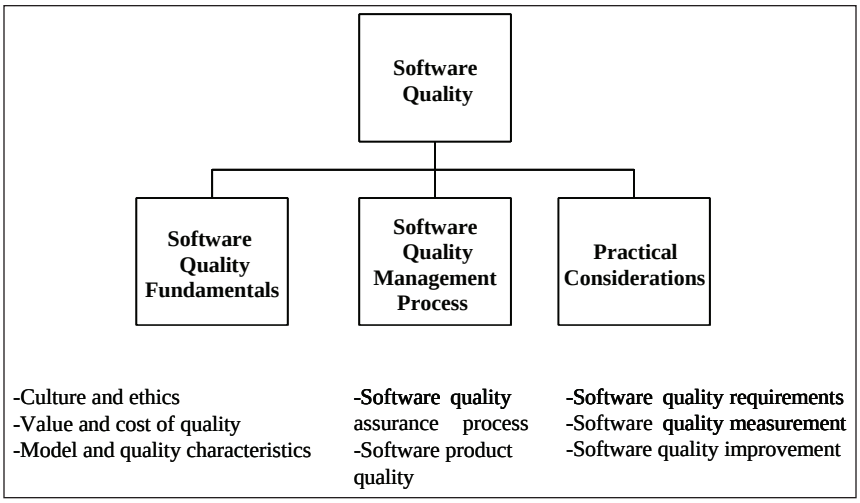
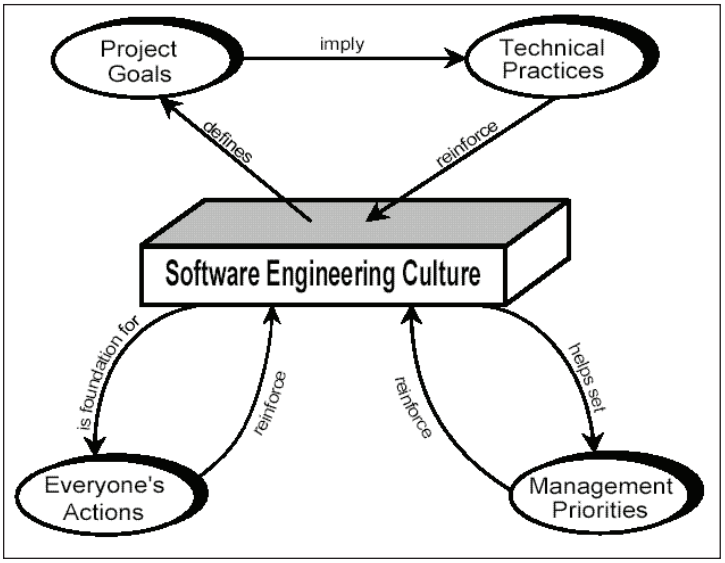


Figure 2. A software engineering culture (Wiegers, 1996)



priorities, and decisions of an individual, as well as those of an organization. Karl Wiegers (1996), in his book *Creating a Software Engineering Culture*, illustrates (see Figure 2) the interaction between the software engineering culture of an organization, its software engineers and its projects. Why should we, as technical people, care about such a "soft" issue? First, a quality culture cannot be bought. It needs to be developed, mostly at the beginning, by the founders of the organization. Then, as employees are selected and hired, the initial leader's culture will start to slowly adjust to the pressures of the environment, as shown in Figure 2. Quality culture cannot be "bolted on" to an organization; it has to be designed-in and nurtured. The ultimate aim of upper management is to instill a culture that will allow the development of high-quality products and offer them at competitive prices, in order to generate revenues and dividends in an organization where employees are committed and satisfied.

The second reason why we should be interested in the cultural aspects of quality is that any change an organization wants to make, for example, moving up on the Capability Maturity Model integration^{SM} (CMMi^{SM}) (SEI, 2002) maturity scale, cannot simply be ordered; the organization has to cope with the current culture when making a change in maturity, especially when such a change implies a profound change in that culture. An organization cannot just "buy and deploy" off-the-shelf processes that contain quality. It has been demonstrated that one of the major inhibitors of change is the culture of the organization. The management of change is imperative (Laporte & Trudel, 1998) in order to achieve the desired result.

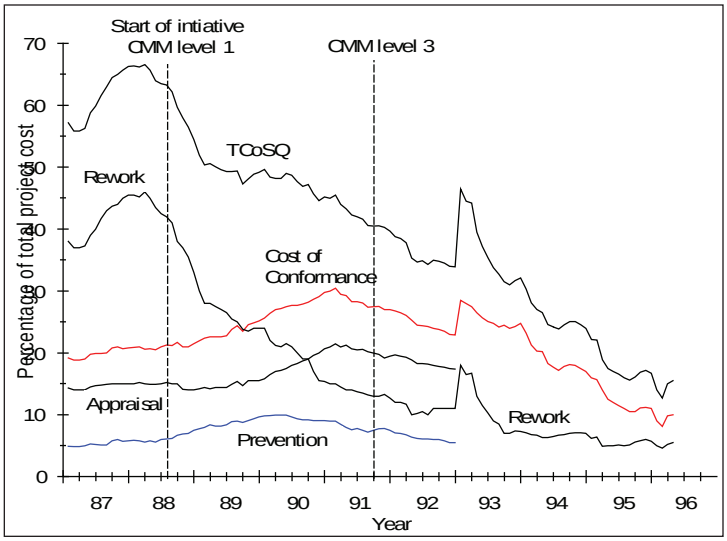
The issue of quality is also central to the Code of Ethics, developed by and for software engineers, which was released in 1999 (Gotterbarn, 1999; Gotterbarn et al., 1999; IEEE-CS, 1999). The Code describes eight top-level technical and professional obligations against which peers, the public, and legal bodies can measure a software engineer's ethical behavior. Each top-level obligation, called a principle, is described in one sentence and supported by a number of clauses which gives examples and details to help in inter-

pretation and implementation of that obligation. Software engineers adopting the Code commit to eight principles of quality and morality. The following are examples: Principle 3 (Product) states that software engineers shall ensure that their products and related modifications meet the highest professional standards possible. This principle is supported by 15 clauses, and clause 3.10 reads that software engineers shall ensure adequate testing, debugging, and review of software and related documents on which they work. The Code has been translated into eight languages: Chinese, Croatian, English, French, Hebrew, Italian, Japanese, and Spanish, and all versions are available at http://seeri.etsu.edu/Codes/default.shtm. It has been publicly adopted by many organizations, as well as by a few universities as part of their curriculum in software engineering.

Value and Costs of Quality

To promote a quality approach to software, a number of arguments must be developed to support its value and benefits. Quality, as a whole, is not always perceived positively and is a hard sell for software project managers. One famous book (Crosby, 1979) addressing the cost of quality has been published on this topic. Since its publication, many organizations, mainly manufacturers, have successfully promoted the use of the cost of quality concepts and framework in their project management processes. A few papers have been published on the adoption of these concepts in the software industry (Campanella, 1990; Diaz, 2002; Dobbins, 1999; Galin, 2004a; Mandeville, 1990; Slaughter et al., 1998), but very few case studies have been published by the software industry itself (Dion, 1993; Haley, 1996; Houston, 1999; Knox, 1993). A paper by Haley of Raytheon (Haley, 1996) illustrates very well the links between the cost of rework and the investment needed to reduce waste. Haley also illustrates that a long-term perspective is needed, as well as a commitment from senior management, in order to capture the benefits. Another fact illustrated by Haley is the link between the investment/benefits and the Capability Maturity Model for Software (CMM)® (Paulk et al., 1993) maturity level of an organization (see Figure 3). At the initial CMM maturity level, most, if not all, organizations have

Figure 3. Improvement data (Dion, 1993; Haley, 1996)



no reliable data or measurement system. When an organization has reached levels 2 and 3, it has the foundation to measure and select beneficial process improvement areas. Many organizations have published the results obtained in climbing the maturity ladder. They show the relationships between maturity level, quality, productivity, and project cost.

Galin (2004b), in a recent software quality assurance book, illustrates the cost and benefits of quality practices. As shown in Table 1, the addition of effective quality practices, such as inspection and review, even reduce the cost of a project by eliminating rework. The left-hand column lists typical quality assurance activities, such as reviews and inspections, of an on-going project. The middle column shows the effectiveness of typical defect removal activities included in a software quality plan. It shows that such a process would deliver a product where 93.1% of the total defects have been detected, leaving 6.9% of them, for a QA cost of 1014.3 units. On the right-hand side, two activities have been added to the quality plan: a design inspection and a code inspection. Even though these activities are not free (18.5 and 158.6 units, respectively), the net result is that 97.4% of the total defects have been detected, leaving only 2.6% of the defects in operation. Note also that the total QA cost is 760.4 units, compared to 1014.3 units previously.

The following quote from Norman Augustine (1997) summarizes this section: "It costs a lot to build bad products."

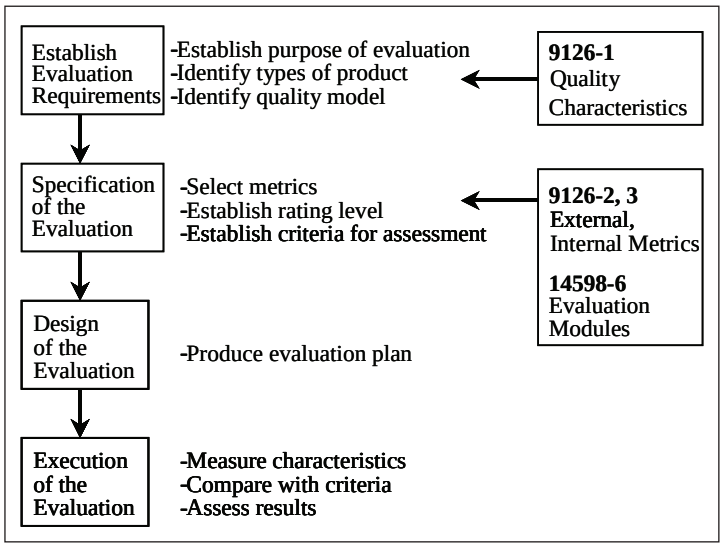
Model and Quality Characteristics

Quality models for software have been developed to assess both software processes and software products. Software process engineering has made many advances in the last decade and has introduced numerous quality models and standards concerned with the definition, implementation, measurement, management, change, and improvement of the software engineering process itself. Assessing the quality of software products requires that the quality characteristics of software be defined and measured. The quality of a software product is determined by its properties. Some of these are inherent to the software product, while others are assigned, by priority, to the software product during its requirements definition process. Process quality is concerned with the technical and managerial activities within the software engineering process that are performed during software acquisition, development, maintenance, and operation.

Table 1. Illustration of the cost and benefits of quality practices (Galin, 2004a)

Quality assurance activity	Standard plan		Comprehensive plan	
	Percentage of defects removed	Cost of removing defects (cost units)	Percentage of defects removed	Cost of removing defects (cost units)
1. Requirements specification review	7.5%	7.5	9%	9
2. Design inspection	--	--	28.7%	71.8
3. Design review	21.3%	53.2	7.4%	18.5
4. Code inspection	--	--	24.4%	158.6
5. Unit test – code	25.6	166.4	4.2%	27.3
6. Integration test	17.8%	284.8	9.8%	156.8
7. Documentation review	13.9%	222.4	9.9%	158.4
8. System test	7.0%	280	4%	160.
Total for internal QA activities	**93.1%**	**1014.3**	**97.4**	**760.4**
Defects detected during operation	6.9%	759	2.6%	286
Total	**100.0%**	**1773.3**	**100%**	**1046.4**

Figure 4. ISO/IEC 14598 – Evaluation process (Sur03)



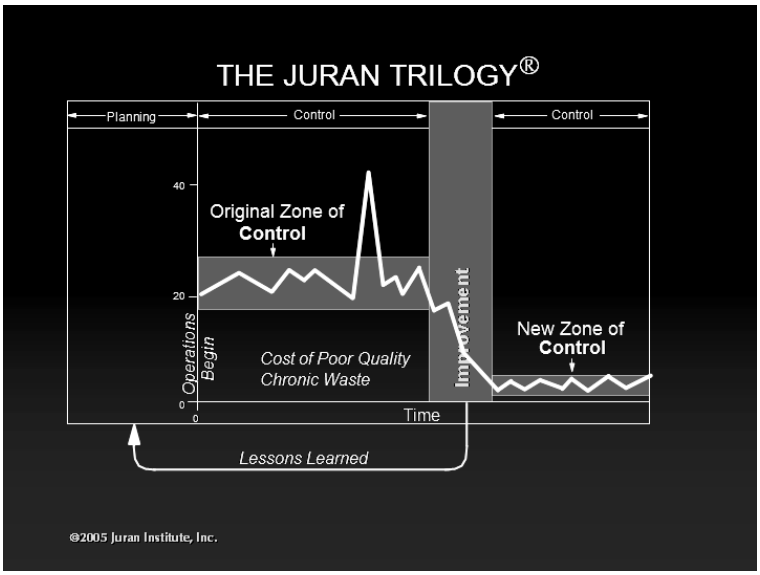
In contrast, software product quality models describe the many quality attributes of software. These have been presented by many authors (Boehm et al., 1978; McCall et al., 1977) and have more history. Although this is an older topic, it has not quite reached the popularity of software process quality models. During the early years, the terminology for software product quality differed from one model of software quality to another. Each model had a different number of hierarchical levels and number of characteristics, and the different naming conventions were especially confusing. Based on these efforts, the International Organization for Standardization (ISO) normalized three software product quality models (i.e., internal quality, external quality, and quality in use) (ISO 9126, 2001) and accompanied them with a set of guides like ISO/IEC14598 which explains how to evaluate a software product (see Figure 4).

SOFTWARE QUALITY MANAGEMENT PROCESS

Software quality management (SQM) defines processes, process owners, requirements for those processes, measurements of the process, their outputs, and, finally, feedback channels (Arthur, 1992). SQM processes are useful for evaluating the process outcome as well as the final product.

The Juran Trilogy Diagram, illustrated in Figure 5, depicts quality management as comprising three basic processes: quality planning, quality control, and quality improvement. The diagram shows how waste caused by mismanaging quality, or not managing it, can be reduced by introducing quality improvement and feeding back the lessons learned from this into the quality planning process. Many, if not most, software engineering managers are not aware of the magnitude of the waste, i.e., between 30 and 40%, in their processes.

Figure 5. Juran trilogy diagram



Software quality management processes consist of many activities. Some may find defects directly, while others indicate where further examination may be valuable. Planning for software quality involves:

1. defining the required product in terms of its quality characteristics;
2. planning the processes to achieve the required product quality.

Some of the specific SQM processes are defined in standard IEEE 12207 (1996):

- Quality assurance process
- Verification process
- Validation process
- Review process
- Audit process

These processes encourage quality and also find possible problems, but they differ somewhat in their emphasis. SQM processes help ensure better software quality in a given project. They also, as a by-product, provide management with general information, including an indication of the quality of the entire software engineering process.

SQM processes consist of tasks and techniques to indicate how software plans (e.g., management, development, and configuration management) are being implemented and how well the intermediate and final products are meeting their specified requirements. Results from these tasks are assembled into reports for management before corrective action is taken. The management of an SQM process is tasked with ensuring that the results of these reports are accurate.

As described in this Knowledge Area (KA), SQM processes are closely related; they can overlap and are sometimes even combined. They seem largely reactive in nature because they address the processes as practiced and the products as produced, but they have a major role at the planning stage in being proactive in terms

of the processes and procedures needed to attain the quality characteristics and degree of quality needed by the stakeholders in the software.

Risk management can also play an important role in delivering quality software. Incorporating disciplined risk analysis and management techniques into the software life cycle processes can increase the potential for producing a quality product (Charette, 1989). Refer to the Software Engineering Management KA of the SWEBOK for material related to risk management.

Software Quality Assurance Process

Software quality assurance processes have been found to have a direct effect on the quality of a software product. During a software development project, it is difficult to completely distinguish the quality of the process used to develop the software from the quality of the product itself. Process quality assurance influences process quality, which in turn influences the quality characteristics of the software product.

Process quality is addressed by two important process models, ISO9001 and CMMi^{SM} (SEI, 2002). First, ISO9001:2000 (ISO 9001, 2000) is an international standard, while CMMi is a process model. Compliance with the key quality concepts of the ISO9001:2000 standard requires an interpretation for each industry. Software, a service industry, has initially had more difficulty interpreting and implementing the ISO9001 standards (Bouman et al., 1999; May, 2002; Niessink, 2000). Service organizations also experience greater difficulty dissociating the final product from the processes used in its production (May, 2002). To help with this problem, additional guidance has been developed by the ISO for the implementation of ISO9001:2000 in the software industry (Hailey, 1998; May, 2002). This guide, numbered ISO90003 (2004), helps link all the software-related standards to the quality system required by ISO9001:2000.

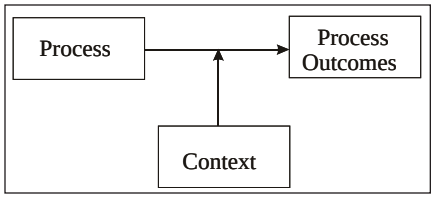
In contrast, CMMi (SEI, 2002) is a process model which proposes proven practices at different maturity levels. It is divided into process areas, and the following are key to quality management: (a) process and product quality assurance; (b) process verification; and (c) process validation.

There was initially some debate over whether ISO9001:2000 or CMMi should be used by software engineers to ensure quality. It has been observed that many organizations use their ISO9001 certification assets as a stepping stone toward meeting the CMMi requirements. It is believed that level 3 of CMMi is reachable for an organization that already has ISO9001:2000 certification. In competitive situations, it is not unusual to find that the two are used concurrently to meet shared quality objectives.

In general, process quality models are concerned with process outcomes. However, in order to achieve the process outcomes desired (e.g., better quality, better maintainability, greater customer satisfaction), we have to implement and measure the process in action.

Current quality models are not complete at this time, as there are many other factors to consider which have an impact on outcomes. Other factors, such as the context, industry domain, capability of the staff, and the tools used also play an important role (see Figure 6). "Furthermore, the extent to which the process is institutionalized or implemented (i.e., process fidelity) is important, as it may explain why 'good' processes do not give the desired outcomes" (ISOTR19759, 2005).

Figure 6. Relationships between process and outcomes (ISOTR19759, 2005)



A word of caution is needed with regard to the term *software quality assurance* (SQA). It is actually a misnomer, because SQA cannot *assure* quality, but only *provide adequate assurance*. ISO 12207 (ISO/IEC12207.0-96) defines an SQA process as a process for providing adequate assurance that the software products and processes in the project life cycle conform to their specified requirements and adhere to their established plans.

The persons or groups providing SQA services need to be independent of the developers in order to provide an unbiased opinion on the conformance of products and processes to the standards imposed by a customer, an organization, or a country. SQA services can be provided by an entity that may be internal or external to the developing organization.

An SQA plan, often mandated by an SQA standard, derives its requirements from the contract, the project plan, and the organization's quality system. It defines the means that will be used to ensure that software developed for a specific product satisfies the user's requirements and is of the highest quality possible within project constraints. The content and format of an SQA plan are available in government documents, such as ESA and NASA, or publicly, in standards such as IEEE standard 730 (IEEE730, 2002).

ISO/IEC 12207 (1996) proposes a minimal content for the SQA plan:

- Quality standards, methodologies, procedures, and tools for performing quality assurance activities (or their references in the organization's official documentation);
- Procedures for contract review and their coordination;
- Procedures for identification, collection, filing, maintenance, and disposition of quality records; resources, schedule, and responsibilities for conducting the quality assurance activities;
- Selected activities and tasks from processes, such as verification, validation, joint review, audit, and problem resolution.

To illustrate the importance of SQA practices, Table 2 (Cusumano et al., 2003) lists a few practices included in a sample of Japanese and American organizational processes. The data show the percentage of sampled projects using a particular practice.

Table 2. Organizational quality practices (Cusumano et al., 2003)

Practices Used	Japan	US
Architectural specifications 7	0.4% 5	4.8%
Functional specifications 9	2.6% 7	4.2%
Detailed designs 8	5.2%	32.3%
Design reviews 1	00%	77.4%
Code reviews	74.1%	71%
Regression testing	96.3%	71%

Table 3. Performance comparison (Cusumano et al., 2003)

Performance Crieria	Japan	US
Median output (see note 1) 4	69 2	70
Median defect rate (see note 2)	.020 .	400
Note 1: Output per programmer-month of effort in new Lines of Code (LOC).		
Note 2: Number of defects reported per 1000 LOC (in the 12 months after delivery to customers).		

Table 4. Productivity and quality improvements over a decade (Cusumano et al., 2003)

	Japan – 1990	Japan - 2003	US – 1990	US -2003
Productivity	389	469	245	270
Quality	.20	.020 .	83 .	400

Table 3 from the same paper shows the performance data resulting from implementation of the practices. As can be observed, the increased performance of the practices in Japan resulted in increased productivity and quality performance in that country.

Table 4 illustrates the important improvements, especially in quality, measured in Japan and the US over the 10-year period.

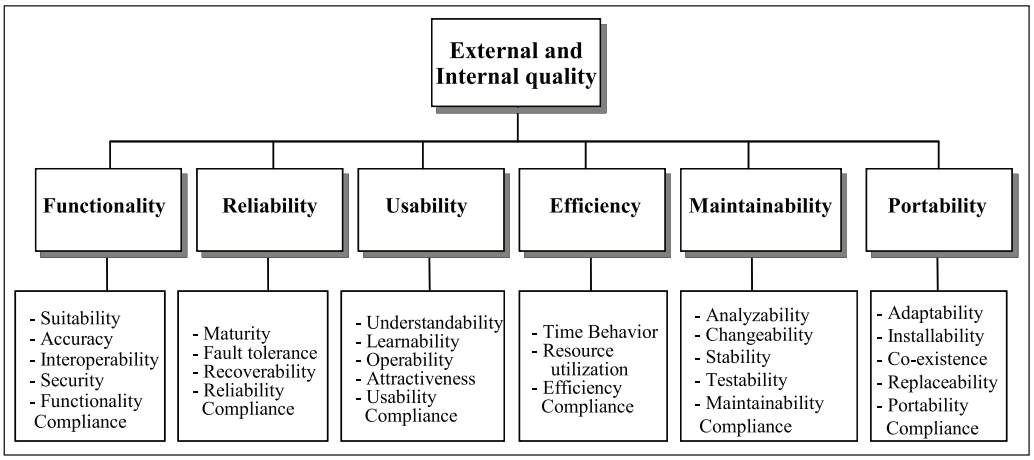
Software Product Quality

The quality of software is not only influenced by the quality of the process used to develop it, but by the quality of the product as well. Software product quality has also been the subject of much research and of many international standards and publications. Software engineers determine, with the stakeholders, the real purpose of future software. In this regard, it is of prime importance to keep in mind that the customer's requirements come first and that they should include non-functional requirements like quality, not just functionality. Thus, the software engineer has a responsibility to elicit quality requirements from future software which may not be made explicit at the outset. It is key that the software engineer assess their relative importance, as well as the level of difficulty in achieving each of them. The section on value and cost of quality presented the additional processes associated with defining and assessing software quality that need to be added to software projects, and each carries additional costs.

Software product quality standards define the related quality characteristics and subcharacteristics, as well as measures which are useful for assessing software product quality (Suryn et al., 2003). In the standards, the meaning of the term *product* is extended to include any artifact which is a software process outcome used to build the final software product. Examples of a product include, but are not limited to, an entire system requirements specification, a software requirements specification for a software component of a system, a design module, code, test documentation or reports produced as a result of quality analysis tasks.

The software product quality model provided in ISO/IEC 9126-1 (ISO9126, 2001) defines six quality characteristics: functionality, reliability, usability, maintainability, portability and efficiency, as well as quality in use, which is defined as

Figure 7. External/internal quality of ISO/IEC 9126 (ISO9126-2001)



effectiveness, productivity, safety, and satisfaction (see Figure 7). The six quality characteristics have defined subcharacteristics, and the standard also allows for user-defined components. The intention is that the defined quality characteristics cover all quality aspects of interest for most software products and, as such, can be used as a checklist for ensuring complete coverage of quality early in the specifications phase.

The quality model suggests three views: internal quality, external quality, and quality in use.

Internal quality provides a 'white box' view of the software and addresses properties of the software product that typically are available during development. External quality provides a 'black box' view of the software and addresses properties related to its execution. "The quality-in-use view is related to application of the software in its operational environment, for carrying out specified tasks by specified users. Internal quality has an impact on external quality, which again has an impact on quality in use. (Suryn et al., 2003)

External and internal quality models presented by ISO/IEC9126 are less trivial and have to be explained further. Let us take the maintainability of software as a quality attribute, and see both the external and internal perspectives in action.

From an external point of view, maintainability attempts to measure the effort required to diagnose, analyze, and apply a change to a software product. Thus, an external measure of maintainability would constitute the effort to make a specific change.

From an internal product point of view, the quality model proposes measurement of the attributes of the software that influence the effort required to modify it. A good example is source code complexity. Structural complexity measures are generally extracted from the source code using observations on the program/classes/module/functions and graph representation of software source code. The more complex the source code, the more difficult it is to grasp its meaning and to understand and document it. These studies are based on work carried out in the 1970s by Curtis, Halstead, and McCabe (Curtis, 1979; Halstead, 1978; McCabe, 1976). These are all internal measures, which in turn affect the external point of view of maintainability.

While most product quality publications have described product quality in terms of the final software and system performance, sound software engineering practice requires that key artifacts impacting quality be evaluated throughout the software development and maintenance processes.

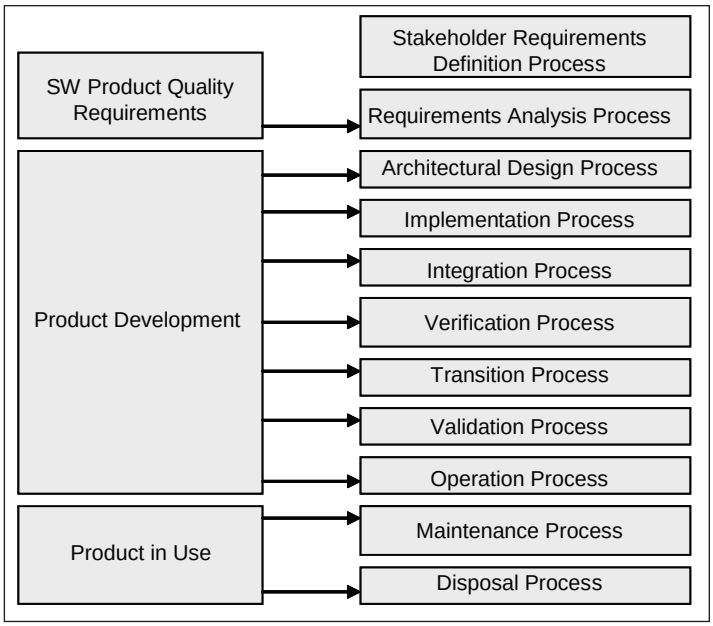
PRACTICAL CONSIDERATIONS

Software Quality Requirements

In order to implement software process and product quality, the software engineer must choose and document quality characteristics at the outset of the projects during the requirements definition process. A generic life cycle model like ISO/IEC 15288—*System life cycle processes* demonstrates generic stages of a typical development process (Figure 8) and the mapping to show *how* standards can be used across all stages of the life cycle of a software product.

We have already stated that both functional and non-functional requirements are captured using the requirements definition process (Pfleeger, 2001). Functional requirements specify what a product should do. Non-functional requirements, also called quality requirements, place constraints on the functional requirements. The quality requirements need to refer to a quality model, like the one described in ISO/IEC 9126-1, to be well understood. Quality requirements address important issues of quality for software products. Software product quality requirements are needed for:

Figure 8. Product life cycle mapping to the technical process life cycle (Suryn et al., 2003)



- Specification (including contractual agreement and call for tender)
- Planning (e.g., feasibility analysis and translation of external quality requirements into internal quality requirements)
- Development (early identification of quality problems during development)
- Evaluation (objective assessment and certification of software product quality)

The requirements definition process typically takes all stakeholders' needs, requirements, and expectations into consideration and ranks them by priority. All relevant stakeholders must be identified before the exercise begins. A first analysis activity during the requirements definition process is aimed at identifying user requirements, sometimes also called business requirements. This view of software requirements should be documented in wording that is easily understood by both users and customers.

Business requirements describe the functionality that the software executes as needed by the stakeholders. Good functional requirements are expressed clearly and are consistent. All requirements need to be correct, non-redundant, and not be in conflict with other requirements. Each requirement has to be uniquely identified and traceable to identified stakeholders. They should also be documented in the business requirements specification (BRS). If a stakeholder requirement is not taken into consideration, the software engineer needs to state the reason why in the documentation.

The second analysis activity is aimed at transforming stakeholder functional requirements into system requirements that can be used by designers to build software. This more technical view of requirements is called a software requirements specification (SRS). During this second analysis activity, the quality requirements are added to the functional requirements. They need to be stated

clearly and unambiguously. It is the software engineer's responsibility to go through each quality characteristic and assess the need to include it in the project. System requirements should also be verifiable and state both the business requirements and the non-functional requirements that the software must possess in order to satisfy all stakeholder requirements. If they do not, they may be viewed, interpreted, implemented, and evaluated differently by different stakeholders. If the software engineer does not become involved in promoting the non-functional requirements, they may be ignored altogether. This may result in software which is inconsistent with user expectations and therefore of poor quality.

We have often talked about stakeholders, but who are they? Stakeholders include all individuals and organizations with a legitimate interest in the software. Different stakeholders will express different needs and expectations that represent their own perspective. For example, operators, testers, and quality engineers are stakeholders. These needs and expectations may change throughout the system's life cycle and need to be controlled when changed. The stakeholders rarely express non-functional requirements, as they may have an unclear understanding of what quality really means in a software process. In reality, they typically perceive it initially as more of an overhead than anything else. Quality requirements will often appear as part of a contractual agreement between acquirer and developer. They may also be required for a specific product quality evaluation when some level of software criticality is reached and human lives are at risk.

We have already stated that a software quality requirement should refer to a specific quality model. To clearly specify quality requirements, ISO/IEC working group six (WG6) of the software engineering subcommittee (SC7) is developing a standard as part of the new ISO/IEC 25000 — *SQuaRE* (*S*oftware Product *Qua*lity *R*equirements and *E*valuation) initiative. The Quality Requirements standard will include a new section for the specification of quality requirements.

A number of obligations are likely to be imposed on software engineers that demand conformance to this new section of ISO/IEC 25000. First, each quality requirement will need to be categorized and prioritized according to an explicit quality model, like:

- Quality-in-use requirement;
- External quality requirement;
- Internal quality requirement.

A software quality requirement will also need to be specified with reference to a set of functional properties. It is also possible that critical functional properties will create an obligation to impose some quality characteristics on a quality requirement.

Since each quality requirement needs to be assessed throughout all the life cycle stages, it needs to be specified in terms of a measurement function and be assigned target values. This new standard will provide examples of measures which may be adapted to specific organizational needs for specifying quality requirements. This new standard is also likely to recommend that software engineers develop and maintain a list of quality measures that are used consistently across their organization. Applying the same set of measures makes it possible to create a homogeneous repository of historical data, which in turn helps in the creation of predictive models such as productivity and estimation models.

The new standard could also focus on the notion of the degree of uncertainty associated with a quality requirement. Measures tending to yield large deviations need to be explained because of their erratic behavior. Explanations include the measurement process of that measure and the errors associated with each of its components.

Many quality standards already identify the review and approval stages, so that the set of

quality requirements is identified and reviewed, and who approved them is clearly stated. The last item likely to appear in such a standard refers to the need for quality requirements, just like functional requirements, to be managed according to a configuration and change management system throughout the entire project.

Software Quality Measurement

The main reason why product quality measurement is only slowly gaining acceptance is the inherent difficulty of measuring in a practical way its quality-impacting processes and artifacts throughout the software engineering process. Some product quality perspectives (e.g., size and source code structure) have been easier to measure than others (e.g., suitability, replaceability, and attractiveness). "Furthermore, some of the dimensions of quality (e.g., usability, maintainability and value to the client) are likely to require measurement in qualitative rather than quantitative form" (ISOTR19759, 2005).

For practical purposes, the software engineer will need to define the quality requirements and the means to assess them early on in the life cycle of the project. During the requirements definition stage, statements about the quality of each quality characteristic are defined as expressing the capability of the software to perform at, and maintain, a specified level of service. The measurement approach requires that individuals who carry out software engineering activities capture the measures and make a judgment about the degree to which the many software properties fulfill their specified requirements as agreed by the stakeholders. Both qualitative and quantitative measures can be collected during all stages of the software life cycle.

The ISO9126 quality model offers definitions of software properties which can be distinguished as quantitative or qualitative. Before quality attributes can be captured and validated, someone has to design them. Then, the most practical means possible must be implemented to measure them.

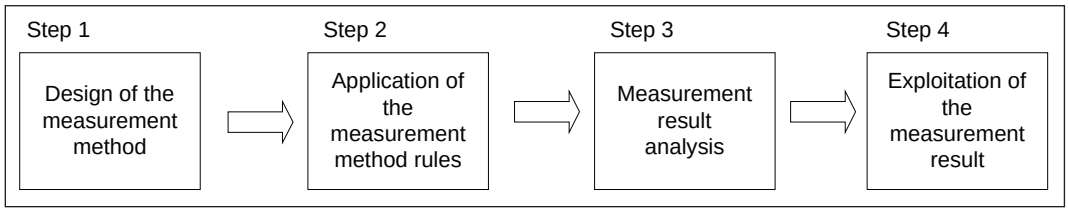
We measure by applying measurement method rules (see Step 2 in Figure 9). A measurement method is a logical sequence of operations that can be used by software engineering personnel to quantify an attribute on a specified scale. The result of applying a measurement method is called a base measure. The ISO9126 quality characteristics that were defined and agreed on can now be measured.

A list of "Quality Measures" is available in ISO/IEC 9126 and ISO/IEC 14598, and the standards present examples of mathematical definitions and guidance for practical measurement of internal quality, external quality, and quality in use.

The proposed ISO/IEC 25000–*SQuaRE* will help by providing guidance, as follows:

- **Measurement reference model and guide** — to present introductory explanations, the

Figure 9. Software measurement method (Abran & Jacquet, 1997)



reference model and the definitions that are common to measurement primitives, internal measures, external measures, and quality-in-use measures. The document will also provide guidance to users for selecting (or developing) and applying appropriate measures;

Example: the construction phase creates intermediate, non-executable software products that can only be evaluated from a static point of view. In this case, users will be directed to the internal quality measures standard, where they can choose the measures that best serve their information needs;

- **Measurement primitives** — to define a set of base and derived measures, or the measurement constructs for the internal quality, external quality, and quality-in-use measurements;
- **Measures for internal quality** — to define a set of internal measures for quantitatively measuring internal software quality in terms of quality characteristics and subcharacteristics;
- **Measures for external quality** — to define a set of external measures for quantitatively measuring external software quality in terms of quality characteristics and subcharacteristics;
- **Measures for quality in use** — to define a set of measures for measuring quality in use. The document will provide guidance on the use of the quality-in-use measures.

Software Quality Improvement

As defined by the ISO/IEC 12207 (1996) standard, an Improvement Process is a process for establishing, assessing, measuring, controlling, and improving a software life cycle process. This definition assumes that an organization already has a written development process in place. A first step in any improvement strategy is to assess the current situation (i.e., a baseline), and then to identify improvement goals and implement the actions required to achieve the goals. Similarly, quality improvement should follow the same steps. Important assumptions are that the organization has stable processes such that data collected are reliable and that they are quality data. If no data are available, some baseline activities have to be performed. The measurement and analysis process area of the CMMi (SEI, 2002) describes the basic components of such a process.

The organization then has to establish quality goals. One approach is to use the well-known Goal Quality Metric (GQM) method (Basili & Rombach, 1988). The organization determines its business goals, and then the technical personnel establish quality goals in order to meet the organization's goals. The quality goals could be established using the Cost of Quality approach or a standard business case approach, that is, by evaluating the cost of prevention and evaluation and the cost of internal and external failure.

Quality could be improved, first, by removing defects as early as possible in the development life cycle using techniques such as reviews, walkthroughs, and inspections (IEEE-1028, 1997), and second, by putting in place defect prevention activities. Software inspection (Gilb & Graham, 1993; IEEE-1028, 1997; Radice, 2002) is an example of a widely recognized quality improvement practice. A Web site displays this technique and provides a wealth of supporting information (www.goldpractices.com/practices/fi/index.php). The inspection process is typically composed of six stages: plan, kick-off meeting, document checking, logging meeting, editing (i.e., rework), and follow-up (Figure 10).

As shown in Table 5, the defect detection effectiveness of inspections can reach 84%. Radice (2002) has collected data from organizations of different CMMi maturity levels which show an effectiveness varying from as low as 50% to as much as 100% (Table 6).

Figure 10. Inspection process (Adapted from Holland, 1998)

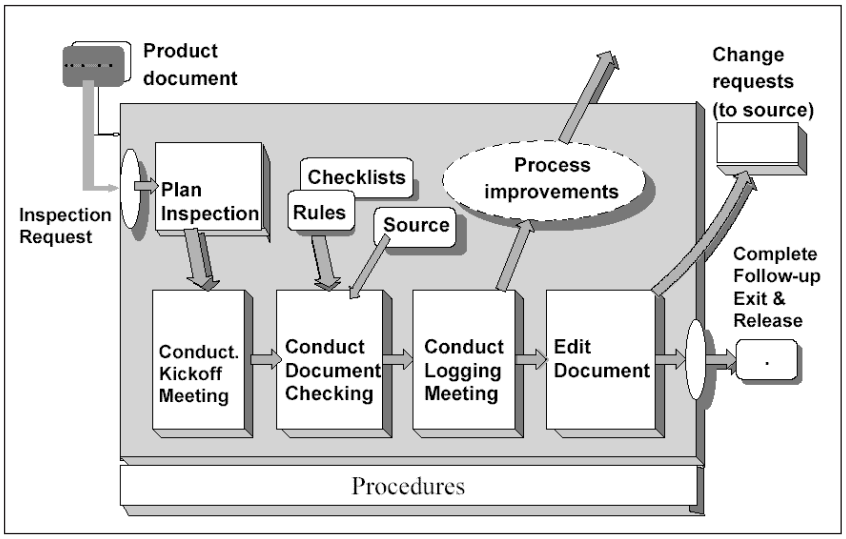


Table 5. Distribution of defect detection effectiveness of inspections (Briand et al., 1998)

Defect Detection Technique	Minimum Value	Most Likely Value	Maximum Value
Design Inspections	25%	57%	84%
Code Inspections	19%	57%	70%

Table 6. Defect detection effectiveness (Radice, 2002)

CMM Maturity Level	Defect Removal Effectiveness
Level 5	90% - 100%
Level 4	75% - 90%
Level 3	65% - 75%
Level 2	50% - 65%
Level 1	Less than 50%

Table 7. Distribution of average effort to detect defects (Briand et al., 1998)

Defect Detection Technique	Minimum Value	Most Likely Value	Maximum Value
Design Inspections	0.58	1.58	2.9
Code Inspections	0.67	1.46	2.7
Testing	4.5	6	17

It is interesting to note that using inspection to detect defects is more effective than testing. Table 7 (Briand et al., 1998) shows the average effort required to detect defects in hours per defect. Unfortunately, even though it has been around for close to 30 years (Fagan, 1976), most organizations ignore this practice. Many organizations, if not most, rely only on tests performed toward the end of the development process to identify defects, even though tests are much less effective than inspections at defect removal. One of the reasons is that inspections find defects where they occur, while testing finds only the symptoms, and then only very late in the development cycle.

FUTURE TRENDS AND CONCLUSION

To illustrate the current state of the quality of many shrink wrap software vendors, here is a typical warranty statement as printed on a software product: *By opening this sealed software media package, you accept and agree to the terms and conditions printed below. If you do not agree, **do not open the package**. Simply return the sealed package. The software media is distributed on an 'AS IS' basis, without warranty. Neither the authors, the software developers nor Acme Inc make any representation, or warranty, either express or implied, with respect to the software programs, their quality, accuracy, or fitness for a specific purpose. Therefore, neither the authors, the software developers nor Acme Inc shall have any liability to you or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by the programs contained on the media. This includes, but is not limited to, interruption of service, loss of data, loss of consulting or anticipatory profits, or consequential damages from the use of these programs. If the media is defective, you may return it for a replacement.*

As more customers demand quality from their vendors, the competitive marketplace will eventually respond to these demands, and software products will be warranted in the same way as any other products. Vendors will be held responsible for making the appropriate corrections at their expense like other industrial sectors. Already a few organizations have demonstrated that quality, productivity, cycle time, and cost could all be improved provided that senior management commit adequate resources and time to improvement. Diaz (1997, 2002) has published results of major improvement programs at Motorola and General Dynamics. For example, at General Dynamics, rework dropped from 23.2% to 6.8%, phase containment effectiveness increased from 25.5% to 87.3%, defects reported by customers dropped from 3.2 per thousand source lines of code (KSLOC) to 0.19, and productivity increased by a factor of 2.9. Such organizations have the data to manage the quality of their processes and their products. As stated by Boehm and Sullivan (2000), since design is an investment activity, knowing the economics of software engineering, these organizations can offer their customers an optimized value proposition by using the proper processes, tools, and metrics.

There is no instant gain, however, as illustrated in Figure 3. It takes time to define and stabilize a process, measure its performances, and make improvements. Those organizations that have succeeded know that improvements require not only resources, but also a major culture change. By definition, a cultural change takes time and can rarely be imposed, and it has to be managed from the top, with senior executives no longer requesting unrealistic schedules with unrealistic budgets to deliver quality products. Moreover, if software developers in the West do not shape up rapidly, what has happened and is still happening to the automobile industry may happen to them, that is, a take-over by foreign, high-quality, lower-cost manufacturers.

With the arrival of India (Moitra, 2001) and China (Ju, 2001) into the software market with their focus on quality and process maturity, the software industry will be pushed to move faster; otherwise, Darwin will favor those who survive this fight.

REFERENCES

Abran, A., & Jacquet, J.-P. (1997, June 2-6). From software metrics to software measurement methods: A process model. *Proceedings of the Third International Symposium and Forum on Software Engineering Standards (ISESS97)*, Walnut Creek, CA.

April, A., Reeker, L., & Wallace, D. (2005). Software quality. *Guide to the software engineering body of knowledge (Ironman version)* (pp. 157-170). Los Alamitos, CA: IEEE-Computer Society Press.

Arthur, L. J. (1992). *Improving software quality: An insider's guide to TQM*. New York: John Wiley & Sons.

Augustine, N. R. (1997). *Augustine's Laws* (6th ed.). Reston, VA: American Institute of Aeronautics and Astronautics.

Basili, V. R., & Rombach, H. D. (1988). The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering, 14*(6), 758-773.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., & Merritt, M. (1978). Characteristics of software quality. *TRW series on software technologies*. Amsterdam, The Netherlands: North-Holland.

Boehm, B. W., & Sullivan, K. J. (2000, June 4-11). Software economics: A roadmap. *Proceedings of the Future of Software Engineering, International Conference on Software Engineering*, Limerick, Ireland.

Bouman, J., Trienekens, J., & Van der Zwan, M. (1999). Specification of service level agreements, clarifying concepts on the basis of practical research. *Proceedings of Software Technology and Engineering Practice* (pp. 11-19). Los Alamitos, CA: IEEE Computer Society.

Briand, L., El Emam, K., Laitenberger, O., & Fussbroich, T. (1998). Using simulation to build inspection efficiency benchmarks for development projects. *Proceedings of the IEEE 20th International Conference on Software Engineering (ICSE 1998)* (pp. 340-349). Los Alamitos, CA: IEEE Computer Society.

Campanella, J. (1990). *Principles of quality costs* (3rd ed). Milwaukee, WI: American Society for Quality Control.

Charette, R. N. (1989). *Software engineering risk analysis and management*. New York: McGraw-Hill.

Crosby, P. B. (1979). *Quality is free*. New York: McGraw-Hill.

Curtis, B. (1979). In search of software complexity. *Proceedings of the IEEE PINY Workshop on Quantitative Software Models* (pp. 95-106). Los Alamitos CA: IEEE Computer Society.

Cusumano, M., MacCormack, A., Kemerer, C., & Grandall, B. (2003). Software development worldwide: The state of the practice. *IEEE Software, 20*(6), 28–34. *Figures abstracted from IEEE Software.*

Diaz, M. (1997). How software process improvement helped Motorola. *IEEE Software, 14*(5), 75-81.

Diaz, M. (2002). How CMM impacts quality, productivity, rework, and the bottom line. *Crosstalk Journal*, U.S. Department of Defense, Hill Air Force Base UT, 15(3), 9-14.

Dion, R. (1993). Process improvement and the corporate balance sheet. *IEEE Software, 10*(4), 28-35. *Figure abstracted from IEEE Software.*

Dobbins, H. D. (1999). The cost of software quality. *Handbook of software quality assurance*. (3rd ed.) (pp. 195-216). Upper Saddle River, NJ: Prentice Hall PTR.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM System Journal*, 15(3), 182-211.

Galin, D. (2004a). *Toward an inclusive model for the costs of software quality. Software Quality Professional, 6*(4), 25-31.

Galin, D. (2004b). *Software quality assurance*. Harlow, UK: Addison-Wesley.

Gilb, T., & Graham, D. (1993). *Software inspection*. Wokingham, UK: Addison-Wesley.

Gotterbarn, D., Miller, K., & Rogerson, S. (1999). Computer society and ACM approve software engineering code of ethics. *IEEE Computer, 32*(10), 84-88.

Gotterbarn, F. (1999) How the new software engineering code of ethics affects you. *IEEE Software, 16*(6), 58-64.

Hailey, V. A. (1998). A comparison of ISO9001 and the SPICE framework, SPICE: an empiricist's perspective. *Proceedings of the Second IEEE International Software Engineering Standards Symposium (ISESS 1998)* (pp. 233-268). Los Alamitos, CA: IEEE Computer Society.

Haley, T. J. (1996). Software process improvement at Raytheon. *IEEE Software, 13*(6), 33-41. *Figure abstracted from IEEE Software.*

Halstead, M. H. (1978, August 21-22). Software science: A progress report. *U.S. Army Computer Systems Command Software Life Cycle Management Workshop.* IEEE.

Holland, D. (1998). Document inspection as an agent of change. In A. Jarvis & L. Hayes (Eds.), *Dare to be excellent*. Upper Saddle River, NJ: Prentice Hall.

Houston, D. (1999). Cost of software quality: Justifying software process improvement to managers. *Software Quality Professional, 1*(2), 8-16.

Humphrey, W. (1990). *Managing the software process*. Software Engineering Institute: Addison-Wesley.

IEEE 730, IEEE Std 730-2002. (2002). *IEEE Standard for Software Quality Assurance Plans.* IEEE.

IEEE 1028, IEEE Std 1028-1997. (1997). *IEEE Standard for Software Reviews.* IEEE.

IEEE 12207, IEEE/EIA 12207.0-1996. (1996). *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes.* IEEE.

IEEE-CS, IEEE-CS-1999. (1999) Software Engineering Code of Ethics and Professional Practice, IEEE-CS/ACM, 1999. Retrieved February 2, 2005, from http://www.computer.org/certification/ethics.htm

ISO9001. (2000, December 15). Quality management systems — Requirements. *International Organisation for Standardisation* (3rd ed.). Geneva, Switzerland: International Organisation for Standardisation.

ISO90003, International Standards Organization. Software Engineering: *Guidelines for the application of ISO9001:2000 to computer software*, ISO/IEC Standard 90003:2004. (2004). Geneva Switzerland: International Organization for Standardization/International Electrotechnical Commission.

ISO9126, International Standards Organization. (2001). *Software Engineering-Product Quality-Part 1: Quality Model*, ISO/IEC Standard 9126-1. Geneva, Switzerland: International Organization for Standardization/International Electrotechnical Commission.

ISOTR19759, International Standards Organization. (2005). *Software Engineering Body of Knowledge* (Tech. Rep. No. ISO/IEC PRF TR 19759).

Ju, D. (2001). China's budding software industry. *IEEE Software, 18*(3), 92-95.

Knox, S. T. (1993). Modeling the cost of software quality. *Digital Technical Journal, 5*(4), 9-16.

Laporte, C. Y., & Trudel, S. (1998). Addressing the people issues of process improvement activities at Oerlikon Aerospace. *Software Process-Improvement and Practice, 4*(1), 187-198.

Levenson, N., & Turner, C. (1993). An investigation of the Therac-25 accidents. *IEEE Computer, 26*(7), 18-41.

Mandeville, W. A. (1990). Software cost of quality. *Proceedings of the IEEE Journal on Selected Areas on Communications, 8*(2), 315-318.

May, W. (2002). *A global applying ISO9001:2000 to software products. Quality Systems Update, 12*(8), 7-13.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308-320.

McCall, J., Richards, P., & Walters, G. (1977). *Factors in software quality* (Vol. I-III). New York: Rome Air Defense Centre.

Moitra, D. (2001). India's software industry. *IEEE Software, 18*(1), 77-80.

Niessink, F. (2000). *Perspectives on improving software maintenance*. Doctoral dissertation, Dutch Graduate School for Information and Knowledge Systems, Utrecht, The Netherlands.

Paulk, M., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). *Capability maturity model for software version 1.1* (Tech. Rep. No. CMU-SEI-93-TR-24). Software Engineering Institute.

Pfleeger, S. L. (2001). *Software engineering: Theory and practice* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.

Radice, R. (2002). *High quality low cost software inspections.* Andover, MA: Paradoxicon.

SEI, Software Engineering Institute. (2002). *Capability maturity model integration for software engineering (CMMi)* (Version 1.1) (Tech. Rep. No. CMU/SEI-2002-TR-028) (pp. 94-528). Pittsburgh, PA: Carnegie Mellon University.

Slaughter, S. A., Harter, D. E., & Krishnan, M. A. (1998). Evaluating the cost of software quality. *Communications of the ACM, 41*(8), 10-17.

Suryn, W., Abran, A., & April, A. (2003). ISO/IEC SQuaRE: The second generation of standards for software product quality. *Proceedings of the Seventh IASTED International Conference on Software Engineering and Applications*, Marina del Rey, CA.

Tylor, E. B., Sir. (1871). *Primitive culture: Researches into the development of mythology, philosophy, religion, art and custom*. London: John Murray.

Wiegers, W. (1996). *Creating a software engineering culture*. New York: Dorset House.

This work was previously published in Measuring Information Systems Delivery Quality, edited by E. Duggan and J. Reichgelt, pp. 28-54, copyright 2006 by Idea Group Publishing (an imprint of IGI Global).

Chapter 1.20
Handling of Software Quality Defects in Agile Software Development

Jörg Rech
Fraunhofer Institute for Experiemental Software Engineering (IESE), Germany

ABSTRACT

Software quality assurance is concerned with the efficient and effective development of large, reliable, and high-quality software systems. In agile software development and maintenance, refactoring is an important phase for the continuous improvement of a software system by removing quality defects like code smells. As time is a crucial factor in agile development, not all quality defects can be removed in one refactoring phase (especially in one iteration). Documentation of quality defects that are found during automated or manual discovery activities (e.g., pair programming) is necessary to avoid wasting time by rediscovering them in later phases. Unfortunately, the documentation and handling of existing quality defects and refactoring activities is a common problem in software maintenance. To recall the rationales why changes were carried out, information has to be extracted from either proprietary documentations or software versioning systems. In this chapter, we describe a process for the recurring and sustainable discovery, handling, and treatment of quality defects in software systems. An annotation language is presented that is used to store information about quality defects found in source code and that represents the defect and treatment history of a part of a software system. The process and annotation language can not only be used to support quality defect discovery processes, but is also applicable in testing and inspection processes.

INTRODUCTION

The success of software organizations—especially those that apply agile methods—depends on their ability to facilitate continuous improvement of their products in order to reduce cost, effort, and time-to-market, but also to restrain the ever increasing complexity and size of software systems. Nowadays, industrial software development

is a highly dynamic and complex activity, which is not only determined by the choice of the right technologies and methodologies, but also by the knowledge and skills of the people involved. This increases the need for software organizations to develop or rework existing systems with high quality within short periods of time using automated techniques to support developers, testers, and maintainers during their work.

Agile software development methods were invented to minimize the risk of developing low-quality software systems with rigid process-based methods. They impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. These methods (and especially extreme programming (XP)) are based upon several core practices, such as *simple design*, meaning that systems should be built as simply as possible and complexity should be removed, if at all possible.

In agile software development, organizations use quality assurance activities like refactoring to tackle defects that reduce software quality. *Refactoring* is necessary to remove *quality defects* (i.e., bad smells in code, architecture smells, anti-patterns, design flaws, negative design characteristics, software anomalies, etc.), which are introduced by quick and often unsystematic development. As time is a crucial factor in agile development, not all quality defects can be removed in one refactoring phase (especially in one iteration). But the effort for the manual discovery, handling, and treatment of these quality defects results in either incomplete or costly refactoring phases.

A common problem in software maintenance is the lack of documentation to store this knowledge required for carrying out the maintenance tasks. While software systems evolve over time, their transformation is either recorded explicitly in a documentation or implicitly through a versioning system. Typically, problems encountered or decisions made during the development phases get lost and have to be rediscovered in later maintenance phases. Both expected and unexpected CAPP (corrective, adaptive, preventive, or perfective) activities use and produce important information, which is not systematically recorded during the evolution of a system. As a result, maintenance becomes unnecessarily hard and the only countermeasures are, for example, to document every problem, incident, or decision in a documentation system like bugzilla (Serrano & Ciordia, 2005). The direct documentation of quality defects that are found during automated or manual discovery activities (e.g., code analyses, pair programming, or inspections) is necessary to avoid wasting time by rediscovering them in later phases.

In order to support software maintainers in their work, we need a central and persistent point (i.e., across the product's life cycle) where necessary information is stored. To address this issue, we introduce our annotation language, which can be used to record information about quality characteristics and defects found in source code, and which represents the defect and treatment history of a part of a software system. The annotation language can not only be used to support quality defect discovery processes, but is also applicable for testing and inspection processes. Furthermore, the annotation language can be exploited for tool support, with the tool keeping track and guiding the developer through the maintenance procedure.

Our research is concerned with the development of techniques for the discovery of quality defects as well as a quality-driven and experience-based method for the refactoring of large-scale software systems. The instruments developed consist of a technology and methodology to support decisions of both managers and engineers. This support includes information about where, when, and in what configuration quality defects should be engaged to reach a specific configuration of quality goals (e.g., improve maintainability or reusability). Information from the diagnosis of quality defects supports maintainers in select-

ing countermeasures and acts as a source for initiating preventive measures (e.g., software inspections).

This chapter targets the handling of quality defects in object-oriented software systems and services. It is concerned with the theory, methodology, and technology for the handling of defects that deteriorate software qualities as defined in ISO 9126 (e.g., maintainability, reusability, or performance). We describe the relevant background and related work concerning quality defects and quality defect handling in agile software projects, as well as existing handling techniques and annotation languages. The subsequent section encompasses the morphology of quality defects as well as their discovery techniques. As the core of this chapter, we present the techniques for handling quality defects after their discovery in an agile and time-critical environment and define an annotation language to record information about quality defects and their history in source code. Thereafter, a section is used to describe the annotation language that is used to record the treatment history and decisions in the code itself. Finally, we summarize several lessons learned and requirements one should keep in mind when building and using quality defect handling methods and notations in an agile environment. At the end of this chapter, we summarize the described approach and give an outlook to future work and trends.

BACKGROUND

This section is concerned with the background and related work in agile software engineering, refactoring, and quality defects. It gives an overview of quality defect discovery, the documentation of defects, as well as source code annotation languages.

Agile Software Development

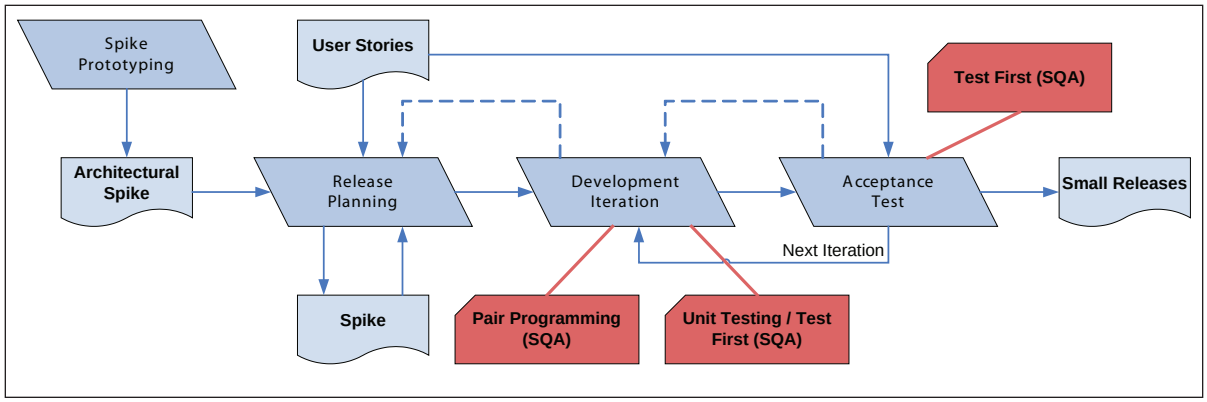
Agile software development methods impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. Agile methods have in common that small releases of the software system are developed in short iterations in order to create a running system with a subset of the functionality needed for the customer. Therefore, the development phase is split into several activities, which are followed by small maintenance phases. In contrast to traditional, process-oriented SE, where all requirements and use cases are elicited, agile methods focus on few essential requirements and incrementally develop a functional system in several short development iterations.

Today, extreme programming (XP) (Beck, 1999) is the best-known agile software development approach. 0 shows the general process model of XP, which is closely connected to refactoring, basically being its cradle (Beck & Fowler, 1999).

These agile methods (and especially extreme programming (XP)) are based upon 12 principles (Beck, 1999). We mention four of these principles, as they are relevant to our work.

1. **Planning Game** is the collective planning of releases and iterations in the agile development process and is necessary for quickly determining the scope of the next release. If the requirements for the next iteration are coherent and concise, more focus can be given to one topic or subsystem without making changes across the whole system.
2. **Small releases** are used to develop a large system by first putting a simple system into production and then releasing new versions in short cycles. The smaller the change to the system, the smaller the risk of introducing complexity or defects that are overlooked in the refactoring (or SQA) phases.

Figure 1. Agile software development (here the XP process)



3. **Simple design** means that systems are built as simply as possible, and complexity in the software system is removed, if at all possible. The more understandable, analyzable, and changeable a system is, the less functionality has to be refactored or reimplemented in subsequent iterations or maintenance projects.
4. **Refactoring** is necessary for removing qualitative defects that are introduced by quick and often unsystematic development. Decision support during refactoring helps the software engineer to improve the system.

In the highly dynamic processes used in agile methods, teams and organizations need automated tools and techniques that support their work without consuming much time. Especially in the refactoring phase, where the software is revised, automation can be used to detect *quality defects* such as code smells (Fowler, 1999), antipatterns (Brown, Malveau, McCormick, & Mowbray, 1998), design flaws (Riel, 1996), design characteristics (Whitmire, 1997), or bug patterns (Allen, 2002). Techniques from KDD support the refactoring of software systems (Rech, 2004), and techniques from knowledge management can foster experience-based refactoring (Rech & Ras, 2004).

Quality Defect Discovery

A central research problem in software maintenance is still the inability to change software easily and quickly (Mens & Tourwe, 2004). To improve the quality of their products, organizations often use quality assurance techniques to tackle defects that reduce software quality. The techniques for the discovery of quality defects are based upon several research fields.

- **Software Inspections** (Aurum, Petersson, & Wohlin, 2002; Ciolkowski, Laitenberger, Rombach, Shull, & Perry, 2002), and especially code inspections are concerned with the process of manually inspecting software products in order to find potential ambiguities as well as functional and non-functional problems (Brykczynski, 1999). While the specific evaluation of code fragments is probably more precise than automated techniques, the effort for the inspection is higher, the completeness of an inspection

regarding the whole system is smaller, and the number of quality defects searched for is smaller.

- **Software testing** (Liggesmeyer, 2003) and debugging is concerned with the discovery of defects regarding the functionality and reliability as defined in a specification or unit test case in static and dynamic environments.
- **Software product metrics** (Fenton & Neil, 1999) are used in software analysis to measure the complexity, cohesion, coupling, or other characteristics of the software product, which are further analyzed and interpreted to estimate the effort for development or to evaluate the quality of the software product. Tools for software analysis in existence today are used to monitor dynamic or static aspects of software systems in order to manually identify potential problems in the architecture or find sources for negative effects on the quality.

Furthermore, several specific techniques for quality defect discovery already exist (Marinescu, 2004; Rapu, Ducasse, Girba, & Marinescu, 2004). Most of the tools such as Checkstyle, FindBugs, Hammurapi, or PMD analyze the source code of software systems to find violations of project-specific programming guidelines, missing or overcomplicated expressions, as well as potential language-specific functional defects or bug patterns. Nowadays, the Sotograph can identify "architectural smells" that are based on metrics regarding size or coupling (Roock & Lippert, 2005).

But the information from these techniques and the resulting CAPP or *refactoring* activities are typically lost after some time if they are not documented in external documents or *defect management* systems (e.g., bugzilla). And even these external data sources are prone to get lost over several years of maintenance and infrastructure changes. The only information that will not get lost is typically the source code itself.

Refactoring

Beside the development of software systems, the effort for software evolution and maintenance is estimated to amount to 50% to 80% of the overall development cost (Verhoef, 2000). One step in the evolution and development of software systems is the process of reworking parts of the software in order to improve its structure and quality (e.g., maintainability, reliability, usability, etc.), but not its functionality. This process of improving the internal quality of object-oriented software systems in agile software development is called *refactoring* (Fowler, 1999). While refactoring originates in from the agile world, it can, nevertheless, be used in plan-driven (resp. heavyweight) software engineering. In general, refactoring (Fowler, 1999; Mens et al., 2004) is necessary to remove quality defects that are introduced by quick and often unsystematic development.

The primary goal of agile methods is the rapid development of software systems that are continuously adapted to customer requirements without large process overhead. During the last few years, refactoring has become an important part in agile processes for improving the structure of software systems between development cycles. Refactoring is able to reduce the cost, effort, and time-to-market of software systems. Development, maintenance, and reengineering effort are reduced by restructuring existing software systems (on the basis of best practices, design heuristics, and software engineering principles), especially in the process of understanding (the impact of new changes in) a system. A reduction of effort also reduces the length of projects and therefore, cost and time-to-market. Furthermore, refactoring improves product quality and therefore is able to reduce the complexity and size of software systems. Especially in agile software

development, methods as well as tools to support refactoring are becoming more and more important (Mens, Demeyer, Du Bois, Stenten, & Van Gorp, 2003).

However, performing manual discovery of quality defects that should be refactored result in either very short or costly refactoring phases. While several automations for refactoring have already been developed (e.g., "extract method" refactoring), the location, analysis, and removal is still an unsystematic, intuitive, and manual process. Today, several techniques and methods exist to support software quality assurance (SQA) on higher levels of abstraction (e.g., requirement inspections) or between development iterations (e.g., testing). Organizations use techniques like refactoring to tackle *quality defects* (i.e., bad smells in code (Beck & Fowler, 1999), architecture smells (Roock et al., 2005), anti-patterns (Brown et al., 1998), design flaws (Riel, 1996; Whitmire, 1997), and software anomalies (IEEE-1044, 1995), etc.) that reduce software quality.

Refactoring does not stop after discovery; even if we had solved the problem of discovering every quality defect possible, the information about the defect, the rationales of whether it is removed (or not), and the refactorings used have to be documented in order to support maintainers and reengineers in later phases. If one knows how to remove a specific quality defect or a group of quality defects, one still needs support, as it is not clear where and under which conditions refactoring activities should be used. Furthermore, product managers need support to organize chains of refactorings and to analyze the impact of changes due to refactorings on the software system. Analogously, quality managers and engineers need information to assess the software quality, identify potential problems, select feasible countermeasures, and plan the refactoring process as well as preventive measures (e.g., code inspections).

Defect Documentation

Today, various repositories exist for documenting of information about defects, incidents, or other issues regarding software changes. This information can be stored in configuration management systems (e.g., CVS, SourceSafe), code reuse repositories (e.g., ReDiscovery, InQuisiX), or *defect management systems.*

The last category is also known as bug tracking (Serrano et al., 2005), issue tracking (Johnson & Dubois, 2003), defect tracking (Fukui, 2002), or source code review systems (Remillard, 2005). They enable a software engineer to record information about the location, causes, effects, or reproducibility of a defect. Typical representatives of defect management systems are open-source variants such as Bugzilla (Serrano et al., 2005), Scarab (Tigris, 2005), Mantis (Mantis, 2005), or TRAC (TRAC, 2005). Commercial versions include Tuppas (Tuppas, 2005), Census from Metaquest (MetaQuest, 2005), JIRA from Atlassian (Atlassian, 2005), or SSM from Force10 (Force10, 2005). These tools are predominantly used in defect handling to describe defects on the lower abstractions of software systems (i.e., source code) (Koru & Tian, 2004) separated from the code.

Defect classification schemes (Freimut, 2001; Pepper, Moreau, & Hennion, 2005) like ODC (Orthogonal Defect Classification) (Chillarege, 1996) are used, for example, in conjunction with these tools to describe the defects and the activity and status a defect is involved in. The ODC process consists of an opening and closing process for defect detection that uses information about the target for further removal activities. Typically, removal activities are executed, but changes, decisions, and experiences are not documented at all—except for small informal comments when the software system is checked into a software repository like CVS.

From our point of view, the direct storage of information about defects, decisions about them, or refactorings applied in the code (as a central point of information) via *annotation languages* such as JavaDoc (Kramer, 1999), doxygen (van Heesch, 2005), or ePyDoc (Loper, 2004) seems to be a more promising solution. The next section describes the relevant background and related work for annotation languages, which are used to record historical information about the evolution of a code fragment (e.g., a method, class, subsystem, etc.).

Source Code Annotation Languages

Annotation languages such as JavaDoc (Kramer, 1999), ePyDoc (ePyDoc, 2005), ProgDOC (Simonis & Weiss, 2003), or Doxygen (van Heesch, 2005) are typically used to describe the characteristics and functionality of code fragments (i.e., classes, methods, packages, etc.) in the source code itself or in additional files. Today several extensions, especially to JavaDoc, are known that enable us to annotate which patterns (Hallum, 2002; Torchiano, 2002), aspects (Sametinger & Riebisch, 2002), or refactorings (Roock & Havenstein, 2002) were or will be used on the source code, and which help us to describe characteristics such as invariants, pre-/ post-conditions, or reviewer names (JSR-260, 2005; Tullmann, 2002). These extensions to the annotation language are called taglets. They are used by doclets in the extraction using, for example, the JavaDoc program. These tools collect the distributed information blocks and generate a (online) documentation rendered in HTML or another file format (e.g., PDF) for better viewing. Typically, these documentations describe the application program interface (API) as a reference for software engineers. Similarly, tools and notations like Xdoclet offer additional tags that are used to generate many artifacts such as XML descriptors or source code. These files are generated from templates using the information provided in the source code and its JavaDoc tags.

Typical content of code annotations is, for example, used to describe the:

- Purpose of a class, field, or method.
- Existence of (functional) defects or workarounds.
- Examples of using the code fragment.

In the following sections and tables, we describe the tags currently available for annotating source code using JavaDoc. JavaDoc is a name for an annotation language as well as the name of a tool from Sun Microsystems to generate API documentation and is currently the industry standard for documenting software systems in Java. The tool uses the tags from the JavaDoc language to generate the API documentation in HTML format. It provides an API for creating doclets and taglets, which allows extending the system with one's own tags (via taglets) and the documentation with additional information (via doclets).

As listed in 0, JavaDoc currently consists of 19 tags that might be used to describe distinguished information (e.g., such as return values of a method) or to format text passages (e.g., to emphasize exemplary source code). The standard tags appear as "`@tag`" and might include inline tags, which appear within curly brackets "`{@tag}`." Inline tags only appear within, respectively behind, standard tags or in the description field (e.g., "`@pat.name … {@pat.role …}`").

Developers can use the JavaDoc tags when documenting source code in a special comment block by starting it with "`/**`" and ending it with "`*/`." A tag is indicated by using an "@" ("at") sign right before the tag name. An example of a JavaDoc comment used for a method is in Box 1.

As an extension to JavaDoc, four refactoring tags were developed in Roock et al. (2002) as described in 0.

Box 1.

/**	Start of JavaDoc comment
* Sorts an array using quicksort	Description of the method
* @author John Doe	Indicate the author
* @param productArray	Describe a parameter
* @return Array The sorted array	Describe the return value
*/	End of JavaDoc comment

Table 1. General tags of the JavaDoc annotation language

Tag	Description	Origin	Type
@author	May appear several times and indicates who has created or modified the code.	JavaDoc 1.0	Context
@param	Describes one parameter of a method (or template class).	JavaDoc 1.0	Function
@return	Describes the returned object of a method.	JavaDoc 1.0	Function
@throws	Describes the (exception-) objects that are thrown by this method.	JavaDoc 1.2	Function
@exception	Synonym for @throws.	JavaDoc 1.0	Function
@version	States the version of this code structure.	JavaDoc 1.0	Context
@since	States the version since when this code was implemented and available to others.	JavaDoc 1.1	Context
@deprecated	Indicates that this code structure should not be used anymore.	JavaDoc 1.0	Status
@see	Adds a comment or link to the “See also” section of the documentation. May link to another part of the documentation (i.e., code).	JavaDoc 1.0	Reference
@serialData	Comments the types and order of data in a serialized form.	JavaDoc 1.2	Context
@serialField	Comments a ObjectStreamField.	JavaDoc 1.2	Context
@serial	Comments default serializable fields.	JavaDoc 1.2	Context
<@code>	Formats text in code font (similar to <code>).	JavaDoc 1.5	Format
<@docRoot>	Represents the relative path to the root of the documentation.	JavaDoc 1.3	Reference
<@inherit-Doc>	Copies the documentation from the nearest inherited code structure.	JavaDoc 1.4	Reference
<@link>	Links to another part of the documentation (i.e., code structure) as the @see tag but stays inline with the text and is formated as “code.”	JavaDoc 1.2	Reference
<@linkPlain>	Identical to <@link> but is displayed in normal text format (i.e., not code format).	JavaDoc 1.4	Reference
<@literal>	Displays text without interpreting it as HTML or nested JavaDoc.	JavaDoc 1.5	Format
<@value>	The value of a local static field or of the specified constant in another code.	JavaDoc 1.4	Reference

Table 2. Refactoring tags by Roock et al. (2002)

Tag	Description
@past	Describes the previous version of the signature.
@future	Describes the future signature of the element.
@paramDef	States the default value expected for a parameter. The syntax is @paramDef <parameter> = <value>.
@default	Defines the default implementation of an abstract method.

Table 3. Pattern tags by Torchiano (2002)

Tag	Description
@pat.name	States the standard name of the pattern as defined in (Gamma, Richard, Johnson, & Vlissides, 1994) (and other).
<@pat.role>	Inline-tag of pat.name that describes the part of the pattern that is represented by this element (e.g., “Leaf” in a composite pattern).
@pat.task	Describes the task performed by the pattern or its role.
@pat.use	Describes the use of the pattern or a role, typically by a method.

Table 4. Other tags

Tag	Description
@contract	Defines bounds of a parameter (or other value). Syntax is “@contract <requires> <min> <= <parameter> <= <max>.”
@inv, @invariant	States an invariant. Syntax is “@inv <boolean expression>.”
@pre	States the precondition for a method.
@post	States the postcondition for a method. This includes information about side effects (e.g., changes to global variables, fields in an object, changes to a parameter, and return values (except if stated in @return).
@issue	Indicates a new requirement or feature that could be implemented. Syntax is @issue [description ...].
@reviewedBy	Indicates a code review for the associated class/interface was completed by a reviewer. Syntax is @reviewedby <name> <date> [notes ...].
@license	Indicates the copyright license used for this code fragment. Syntax is @license [description ...].
@category	Annotates the element with a free attribute / category. @category <category>.
@example	@example <description>.
@tutorial	Link to a tutorial.
@index	Defines the text that should appear in the index created by JavaDoc.
@exclude	States that this element should not be included in the API by the JavaDoc command.
@todo	Indicates that further work has to be done on this element.
@internal	Comments to this element that are internal to the developer or company.
@obsolete	Used if deprecated elements are actually removed from the API.
@threadSafe	Indicates whether this element is threadsafe.
@pattern	Formally describes a pattern existence with the syntax @pattern <pattern name>.<instance name> <role name> <text>.
@aspect	Describes an aspect existence with the syntax @aspect <name> <text>.
@trace	Describes a pattern existence with the syntax @trace <name> <text>.

Table 5. Annotation languages in comparison

Language	Extension	# of Tags	Test Info	Inspection Info	Pattern Info	Refactoring Info
JavaDoc 1.5	Standard	19	No	No	No	No
	Roock et al.	5	Semi-Formal (1)	No	No	Informal (4)
	Torchiano	10	No	No	Semi-Formal (3)	No
	Sametinger et al.	3	No	No	Informal (3)	No
	Tullmann	4	No	Informal (1)	No	No
	Kramer	3	Informal (3)	No	No	No
	JSR-260	9	No	No	No	No

To note the existence of patterns in a software system as well as the task and role as described in the pattern definitions, several tags were developed by Torchiano (2002) and are listed in 0.

Furthermore, several other groups of annotations exist for various purposes. The following tags are from Roock et al. (2002) (@contract), Kramer (1998) (@inv, @pre, @post), Tullmann (2002) (@issue, @todo, @reviewedBy, @license), Sametinger et al. (2002) (@pattern, @aspect, @trace), and JSR-260 (2005) (@category, @example, @tutorial, @index, @exclude, @todo, @internal, @obsolete, @threadSafe).

The characteristics of source code annotation languages can be differentiated by the number of tags and the formality of their expressiveness. We differentiate between three categories of formality:

1. **Formal:** An explicit and unambiguous specification of the content. A formal tag might include an informal section like a description or note to the formal part (e.g., the tag "param" in JavaDoc has an informal part to describe the meaning of the parameter). In particular, the formal part of a tag must be processable by a computer.
2. **Semi-formal:** A structured or formal representation that is ambiguous or not directly processable by a computer.
3. **Informal:** An unstructured and possibly ambiguous specification of content.

In summary, the tags used in JavaDoc and its extensions can be used to describe characteristics of the source code on a relatively granular or semi-formal level. The processing of these annotations can be used to generate API documentations with additional information about patterns, aspects, or signature changes. The recording of quality defects discovered and refactorings applied as well as rationales or experiences about their application can only be accomplished using free text in the API description.

Furthermore, these annotation languages and their extensions have different target areas in the field of software quality assurance in order to store information about tests, inspections, patterns, and refactorings. 0 shows a comparison of several annotation languages in relevant areas.

Quality Defects and Quality Defect Discovery

The main concern of software quality assurance (SQA) is the efficient and effective development of large, reliable, and high-quality software systems. In agile software development, organizations use techniques like refactoring to tackle "bad smells in code" (Beck et al., 1999), which reduce software

qualities such as maintainability, changeability, or reusability. Other groups of defects that do not attack functional, but rather non-functional aspects of a software system are architecture smells (Roock & Lippert, 2004), anti-patterns (Brown et al., 1998), design flaws (Riel, 1996; Whitmire, 1997), and software anomalies in general (IEEE-1044, 1995).

In this chapter, we use the umbrella term *quality defects* (QD) for any defect in software systems that has an effect on software quality (e.g., as defined in ISO 9126), but does not directly affect functionality. Whether the quality defect is automatically discoverable (Dromey, 1996, 2003) or not (Lauesen & Younessi, 1998), an annotation language and method that can be used to support the handling of quality defects should record information about quality characteristics and quality defects in order to represent their status and treatment history. This section will elaborate on this concept and describe several quality defects, their interrelation, symptoms, and effects.

Today, various forms of quality defects exist with different types of granularity. Some target problems in methods and classes, while others describe problems on the architecture or even process levels. In this chapter, we only focus on quality defects on the code level. The representatives on this level are:

- **Code Smells:** The term code smell is an abbreviation of "bad smells in code," which were described in Beck et al. (1999). Today, we have many code smells that are semi-formally described and can be used for manual inspection and discovery. There are at least 38 known code smells with 22 in Fowler (1999), 9 new ones in Wake (2003), 5 new ones in Kerievsky (2005), and 2 in Tourwe and Mens (2003). Code smells are indicators for refactoring and typically include a set of alternative refactoring activities in their description, which might be used to remove them.
- **Architectural Smells:** Very similar to code smells are architectural smells that describe problems on the design level. Yet, the 31 architectural smells described in Roock et al. (2005) do not only apply on the design level but also on the code level. They typically describe problems regarding classes in object-oriented software and interrelations between them.
- **Anti-Patterns:** Design patterns (Gamma et al., 1994) and anti-patterns (Brown et al., 1998) represent typical and reoccurring patterns of good and bad software architectures and were the start of the description of many patterns in diverse software phases and products. While patterns typically state and emphasize a single solution to multiple problems, anti-patterns typically state and emphasize a single problem to multiple solutions. An anti-pattern is a general, proven, and non-beneficial problem (i.e., bad solution) in a software product or process. It strongly classifies the problem that exhibits negative consequences and provides a solution. Built upon similar experiences, these anti-patterns represent "worst-practices" about how to structure or build a software architecture. An example is the "lava flow" anti-pattern, which warns about developing a software system without stopping sometimes and reengineering the system. The larger and older such a software system gets, the more dead code and solidified (bad) decisions it carries along.
- **Bug Patterns:** These patterns are concerned with functional aspects that are typically found in debugging and testing activities. In Allen (2002), 15 bug patterns are described, which describe underlying bugs in a software system.
- **Design Flaws and (Negative) Design Characteristics:** Whitmire (1997) describes nine distinct and measurable characteristics of an object-oriented design. These characteristics

such as "similarity" describe the degree to which two or more classes or domain-level abstractions are similar in terms of their structure, function, behavior, or purpose.

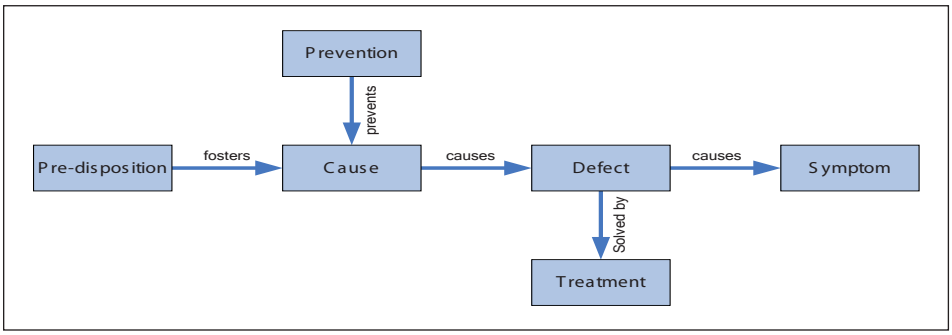
- **Design Heuristics:** Design heuristics provide support on how to construct software systems and, in a way, define quality defects by their absence. They range from the "information hiding" principle to guidelines such as "Eliminate irrelevant classes from your design." There are 61 design heuristics described in Riel (1996) and 14 principles described in Roock et al. (2005).

As listed, there are many quality defects of various granularities and they are described in different forms. To give a more concrete idea of quality, we describe two of them in the following:

1. **Long Method:** In object-oriented programming, one should pay attention to the fact that methods are not too long. The longer a method, the more difficult it is to be understood by a developer. Comprehensibility and readability are negatively affected by the length of a method and thus negatively affect maintainability and testability. Moreover, a short understandable method typically needs less comments than a long one. Another advantage of short methods is the fact that a developer does not constantly scroll and break his reading flow. The most obvious method for discovering long methods is the metric number of lines (LOC) per method. But the question of which method is too long and constitutes a problem is not easily answered. This must either be specified or found by detecting anomalies from the standard distribution. Nevertheless, a method exceeding this threshold value must not necessarily be shortened if other, more important, quality constraints would be negatively affected.
2. **Shotgun Surgery:** This denotes the problem that several classes are always changed in the same group, for example, if the system is adapted to a new database scheme and the same two classes are changed each time. The expandability of the system is thus constrained, and if one class is forgotten during a change, it is more likely to fail. The discovery of shotgun surgery is very difficulty and requires either change metrics or specified rules.

While these problems might not represent problems that have a directly tangible effect on quality, they might become problematic in future evolution or refactoring activities and should be

Figure 2. Conceptual model of the quality defect ontology (software product level)



removed as fast as possible—if the time is available. These are only two of many described quality defects. Nevertheless, they show that quality defects describe problems on different levels of complexity and might occur in parallel in one situation (i.e., in one code fragment).

0 depicts the general model for the concepts that are used to describe quality defects and that are linked to them. A software system might have *predispositions* that foster or enable the creation of quality defects. These defects themselves have *causes* that are responsible for the defects being integrated into the system. The quality *defects* might have a negative as well as a positive effect on specific qualities and are perceivable via specific *symptoms*. Finally, the defects are solved or removed via specific treatments after they are discovered, or the causes might be prevented by special preventive measures.

In software engineering (SE) and especially in the case of quality defects for a software product, the context of a defect can be described as listed in 0.

In the example on the right side, the predisposition "bad architecture" causes a "cluttered functionality," which results in a "shotgun surgery" defect. This quality defect can be discovered by the symptom of "recurring changes to the same set of software units" and might be removed by the "inline class" refactoring. A prevention of this problem would be a "good" systematic architecture with clear separation of functionality (e.g., in the form of a pattern-based architecture).

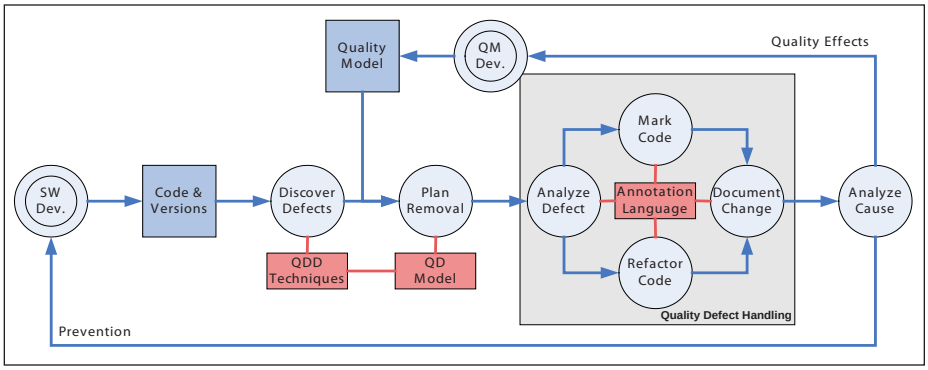
Handling of Quality Defects

Typically, during agile development with short iterations, several quality defects are introduced

Table 6. Examples for software engineering techniques

	Example 1	Example 2
Predisposition	Data processing system	Lack of good architecture/design
Cause	Large data processing algorithms	Distributed functionality
Defect	"Long method"	"Shotgun surgery"
Side-effects of defect	Increase analyzability effort	Increased maintenance effort
Symptom	Many lines of code	Recurrent changes to the same units
Treatment	Extract method	Inline class(es)
Side-effects of treatment	Increased subroutine calls (worsens performance)	Divergent change
Prevention	Optimize algorithm (phase)	Pattern-based architecture

Figure 3. The quality defect discovery and handling process model



into the software system and are discovered especially in the refactoring phase. To facilitate the annotation of source code and the processing of quality defect removal, a mechanism to store the information from the QDD process is required.

The Handling Process

In the following, the general process of quality defect discovery and refactoring is depicted. 0 shows the process model that is either initiated during software development or during a special maintenance (resp. refactoring) activity.

In the execution of the process, the following sub-processes are performed:

- **Discover Defects:** Manual or automatic quality defect discovery techniques are used to analyze the source code and versions thereof from the software repository. Potential quality defects are identified and the affected code (of the most current version) is annotated.
- **Plan Removal:** Based on the discovered quality defects (annotated with a special tag) and a previously defined quality model, a sequential plan for the refactoring of the software system (or part thereof) is constructed.
- **Analyze Defects:** The software engineer processes the list of potential quality defects based on their priority, analyzes the affected software system (or part thereof), and decides if the quality defect is truly present and if the software system can be modified without creating too many new quality defects.
- **Refactor Code:** If the quality defect is to be removed from the software system, the engineer is briefed about the existing quality defects and their rationales as well as about available refactorings, their impact on software quality, and previously made experiences with the kind of quality defect and refactoring at hand.
- **Mark Code:** If a potential quality defect is unavoidable or its removal would have a negative impact on an important quality (e.g., performance), this decision is recorded in the affected part of the software system to prevent future analysis of this part.
- **Document Change:** After the refactoring or marking, the software system is annotated with specific tags about the change or decision, and the experience about the activity is recorded within an experience database (i.e., a database in an experience factory (Basili, Caldiera, & Rombach, 1994b) for storing, formalizing, and generalizing experiences about software development and refactoring activities (e.g., to construct defect patterns from multiple similar defect descriptions)).
- **Analyze Cause:** Statistics, information, and experiences about the existence of quality defects in the software systems are fed back into the early phases of the software development process to prevent or at least reduce their reoccurrence. Preventive measures include, for example, software requirement inspections or goal-oriented training of employees. Furthermore, information about dependencies between qualities, quality defects, and refactorings are fed back into the quality model development process in order to continuously improve the techniques for quality model development.

Decision Support in Handling

In order to support decisions about what to refactor in a software system, we developed several methods and techniques. The following questions act as the guiding theme for the development and enactment of decision-making (i.e., the “plan removal” or “refactor code” phase) as well as

understanding (i.e., the "analyze defect" or "document change" phase) in refactoring phases:

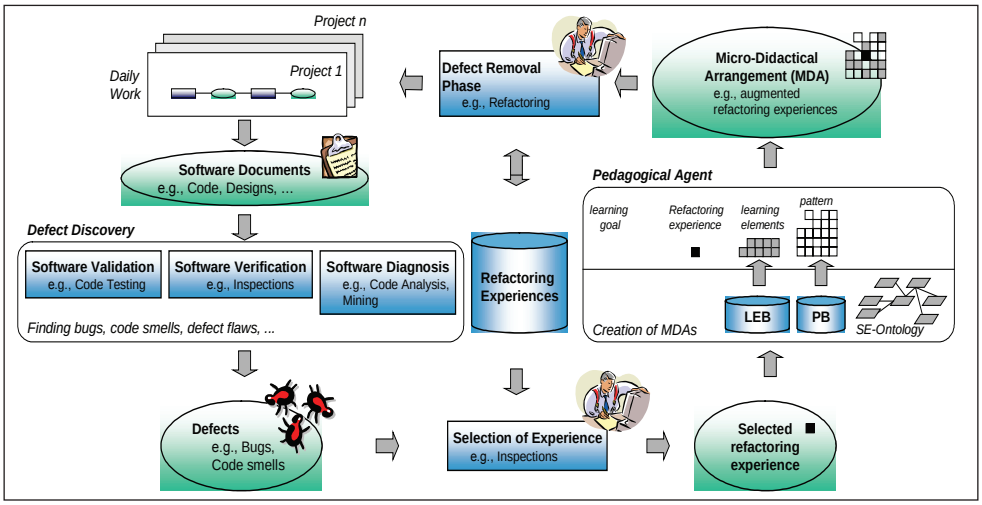
- **Decision problem 1:** Which quality defects should be refactored and which might stay in the system?
- **Decision problem 2:** In which sequence should one refactor multiple quality defects in order to minimize effort?
- **Comprehension problem 1:** How does one understand and detect the quality defect in the concrete situation?
- **Comprehension problem 2:** How does one understand the refactoring in the concrete situation and its effect on the software system?
- **Decision problem 3:** Which refactoring should one use if multiple ones are available?
- **Comprehension problem 3:** Which information should one record after refactoring or marking for later evolution, maintenance, or reengineering activities?
- **Decision problem 4:** Did the understanding of the problem or the refactoring result in valuable experience that should be recorded to support later activities (possibly by others)?
- **Comprehension problem 5:** How should one record the experience?

Decision Support in Software Refactoring

Our approach encompasses several methods for supporting the decision of where, when, and in what sequence to refactor a software system as depicted in 0. Beginning from the left upper corner and going counterclockwise, knowledge about quality defects from defect discovery processes is used to retrieve experiences associated with similar defects from previous refactorings. These experiences are used to handle quality defects in the defect removal phase. Additionally, suitable experiences are augmented by so-called micro-didactical arrangements (MDA) (Ras, Avram, Waterson, & Weibelzahl, 2005), which initiate learning processes and aim at improving the understandability, applicability, and adaptability of the experience in the specific context.

As shown in 0, we define six phases, based on the quality improvement paradigm (QIP) (Basili,

Figure 4. Experience-based semi-automatic reuse of refactoring experiences



Caldiera, & Rombach, 1994a), for the continuous handling of quality defects. In contrast to the quality defect handling process as depicted in 0, these phases are not concerned with quality defects in a specific product, but with the learning process about the quality defects themselves and their effect on the software qualities. 0 represents the realizations of phase 2 ("discover defect"), phase 3 ("plan removal"), and phase 4 (the "quality defect handling" block).

In 0, we first start with the definition of the quality model consisting of qualities that should be monitored and improved. For example, this may result in different goals (i.e., quality aspects), as reusability demands more flexibility or "openness," while maintainability requires more simplicity. Phase 2 is concerned with the measurement and preprocessing of the source code to build a basis for quality defect discovery (i.e., "discover defects"). Results from the discovery process (i.e., quality defects) are represented and prioritized to plan the refactoring in phase 3 (i.e., "plan removal"). Here, the responsible person has to decide which refactorings have to be executed (i.e., "analyze defect") in what configuration and sequence, in order to minimize work (e.g., change conflicts) and maximize the effect on a specific quality. In phase 4, the refactoring itself is (or is not) applied to the software system (i.e., "Refactor Code" or "Mark Code") by the developer, which results in an improved product. Phase 5 compares the improved product with the original product to detect changes and their impact on the remaining system (i.e., "analyze cause"). Finally, in phase 6, we document the experiences and data about the refactoring activity, changes to the software system, and other effects in order to learn from our work and continuously improve the model of relationships between quality, refactorings, and quality defects.

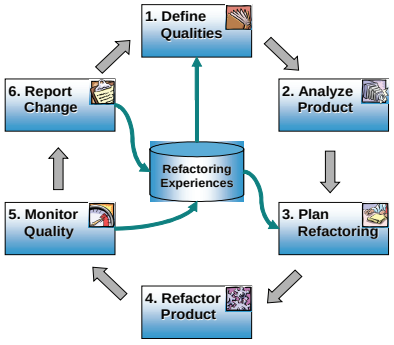
As indicated previously, the KDD sub-processes are grouped in phase 2. We select source code from a specific build, preprocess the code, and store the results in the code warehouse, analyze the data to discover quality defects, discover deviations from average behavior, cluster code blocks with severe or multiple quality defects, and represent discovered and prioritized quality defects to the user.

An Example of DS for QDD

For example, we may detect a method in an object-oriented software system that has a length of 300 LOC. As described in Fowler (1999), this is a code smell called long method. A long method is a problem especially in maintenance phases, as the responsible maintainer will have a hard time understanding the function of this method.

One suitable refactoring for the mentioned code smell might be the refactoring simply called extract method: the source code of the long method is reviewed to detect blocks that can be encapsulated into new (sub-)methods. Experiences with the extract method refactoring are used to support the decision on where, when, how, and if the refactoring has to been implemented. For example, the developer might remark that every block of code that has a common meaning, and could be commented respectively, could also be extracted into several smaller methods. Furthermore, the developer might note that the extraction of (sub-) methods, from methods implementing

Figure 5. Quality-driven refactoring



complex algorithms, can affect the performance requirements of the software system and therefore might not be applicable.

Additionally, the generation of new methods might create another smell called "large class" (i.e., the presence of too many methods in a class), which might complicate the case even further. Finally, the new experiences are annotated by the developer and stored in the refactoring experience base.

While this example only touches a simple quality defect and refactoring, more complex refactorings influence inheritance relations or introduce design patterns (Fowler, 1999).

An Annotation Language to Support Quality Defect Handling

This section describes, defines, and explains a language that will be used to annotate code fragments that are either contaminated by quality defects or checked by a software engineer and cleared of quality defects. As described in the background section, several annotation languages for the annotation of source code already exist that are not adequate. This new language is used to keep decisions about quality defects persistent and over time builds a "medical history" of the source code fragment (e.g., a class).

Goals and Characteristics of Annotation Languages

All annotation languages represent a basis for describing additional information about the software system directly at the code level. *Target groups* (or users) for the documentation/annotation language are:

- **Developers,** who want to use the source code and acquire information via the API descriptions (e.g., for software libraries).
- **Testers,** who want to develop test cases and need information about the pre- and post-conditions as well as the functionality to be tested.
- **Maintainers,** who want to evolve the system and need information about existing quality defects, rationales for their persistence (e.g., refactoring would cause loss of performance), or past refactorings (e.g., to update the software documentation such as design documents).

In our case, an annotation language that is targeted at supporting the handling of quality defects should encompass several key aspects. The requirements for such an annotation language should cover uses such as:

- **Annotate change** for later understanding by the same and other readers (e.g., maintainers).
- **Mark fragment** that a quality defect is detected but can or must stay in the system.
- **Note membership** in a larger quality defect or refactoring activity that encompassed multiple code fragments for later impact analyses.
- **Annotate quality aspects** for later reuse, etc.
- **Tag additional information** in the code fragment freely or based on a classification (e.g., "problematic class," "quicksort algorithm," "part of subsystem X") to support later reuse or maintenance/reengineering activities (similar to social software or Web 2.0 approaches).

We identified the following information blocks of an annotation language that should be recorded with an annotation language and that are based on the six knowledge types from knowledge management (Mason, 2005):

- **Know-what:** Record the currently present *quality defects* that were found manually or automatically.

- **Know-how:** Record the *transformation history* (similar to the medical history of a human patient).
- **Know-why:** Record the *rationales* why a refactoring was applied or why a quality defect is still present in order to prevent recurrent defect analysis or refactoring attempts.
- **Know-where:** Record the *location* in the annotated code as well as associated code fragments that where changed as well.
- **Know-who:** Record the tool or *person* (i.e., developer or maintainer) who applied the refactoring.
- **Know-when:** Record the time or *version* when the quality defect was found or the refactoring was applied. This could also be used to define a trigger when a refactoring has to be applied (e.g., if several other (larger) refactorings or design decision have to be made).
- **Context:** Record the frame of reference or context in which the quality defect was discovered. This includes especially the quality model used to decide which quality defect has a higher priority over other quality defects.

The following requirements for tags and other constructs in such an annotation language to support refactoring and maintenance activities are:

- **Unambiguous:** The names of tags, quality defects, refactorings, or other reoccurring terms should be unique and used consistently throughout the system.
- **Machine-readable:** The syntax of tags should be formal, exact, and consistent to avoid confusion and enable the interpretation and usage by supporting systems (e.g., defect discovery tools).
- **Local completeness:** The power of the syntax should be large enough to cover all existing cases. Full comprehensiveness is probably not possible except by allowing informal free text attributes.
- **Flexibility:** The syntax should not limit the extension by new tags or tag attributes.
- **Independence:** Tags should describe information that is mutually exclusive, and the occurrence of two or more tags should be independent from one another.

Beside the additional documentation of the software system, the annotation language will increase the semantic coupling between code fragments and reduce the presence of quality defects such as “shotgun surgery.”

RAL: The Refactoring Annotation Language

The refactoring annotation language (RAL) is used to record the currently existing quality characteristics, symptoms, defects, and refactoring of a code fragment regarding a specific quality model. Furthermore, it is used to store the rationales and treatment history (e.g., sequence of refactorings).

In the following tables, the core set of tags from RAL are described based on the JavaDoc syntax and using existing JavaDoc and supportive tags, that are used in the description and will be described after the core tags. Information blocks starting with a double cross “#” indicate an ID or standardized term from an external, controlled vocabulary or taxonomy.

A symptom tag as defined in 0 describes a metric or characteristic of the code fragment and is used as an indicator for the statistical or rule-based identification of quality defects. The tag acts as an annotation of a specific symptom from a controlled vocabulary in order to have a unique identifier and a reference for further information about the symptom. The since tag from JavaDoc is used to identify the version based on which the quality symptom was first calculated.

The quality defect as defined in 0 represents a code smell, antipattern, etc. present in this code

Table 7. The @symptom tag

Tag Syntax	@symptom <#Symptom-ID> <@value value> <@since #version>
Example	@symptom “LOC” @value “732” @since 1.2

Table 8. The @defect tag

Tag Syntax	@defect <#QD-ID> <@since #version> <@status #Status> <@rationale text>
Example	@defect “Long Method” @since 1.2 @status “untreated”

Table 9. The @refactoring tag

Tag Syntax	@refactoring <#Refactoring-ID> <@rationale text> <@status #Status> <@link fragment> <@ author name>
Example	@refactoring “Extract Method” “Applied as quality model rates maintainability higher than performance” @status “treated” @link “ExtractedMethod” @author “John Doe”

Table 10. The @quality-model tag

Tag Syntax:	@quality-model Name <@see file>
Example	@quality-model “QM-Dep1-Project2” @see <A href=http://www.sw-quality.de/QM-Dep1-Project2.pdf>

Table 11. Support tags

Tag	Description
@status	“@status #status” indicates the current status of the superior tag or source code using the vocabulary “discovered,” “inWork,” “treated,” or (deliberately) “untreated.”
@rationale	“@rationale text” declares a rationale about the existence or status of the superior tag or source code.

fragment. It is used to annotate a specific quality defect from a controlled vocabulary in order to have a unique identifier and reference for more information about a specific quality defect type and potential treatments. The since tag from JavaDoc is used to identify the version where the quality defect was first noticed.

A refactoring tag as defined in 0 is a description of a single refactoring that was applied for removing one or more quality defects. Optionally, a project-internal URI to other code fragments directly affected by the refactoring (e.g., if two classes interchange a method during the same refactoring) can be stated.

The quality model tag as defined in 0 is used as a reference to the quality model that defines which quality characteristics are important, what priority or decision model lies beneath, and which quality defects are relevant to a specific part of the software system. Optionally, it refers to a URI of a file containing the specific (machine-readable) quality model.

The supportive tags used in the previous tag descriptions are given in 0.

Depending on the processor that would render a quality documentation from these tags, some tags might be used only once and inherited by lower levels. For example, the quality model tag needs only be stated once (e.g., for the whole project) or twice (e.g., for the client and server part) in a software system.

RAL is primarily used to annotate source code. Therefore, in order to annotate documents of higher abstraction, like UML-based design

documents (e.g., platform-independent models in MDA) using the XMI Format or formal requirement documents, similar languages (probably based on other languages such as JavaDoc) need to be defined.

Handling Quality Defects Using RAL

Software annotation languages like JavaDoc or Doxygen and extensions like RAL can now be used to document the functionality, structure, quality, and treatment history of the software system at the code level. The formal basis of the language enables tools to read and write this information automatically to generate special documents or trigger specific actions.

The core tags `@symptom`, `@defect`, and `@refactoring` build on top of each other and might be recorded by several different tools. This enables the intertwined cooperation of different tools, each with a specific focus, such as to calculate metrics or to discover quality defects. For example, one tool might measure the source code and its versions to extract numerical and historical information and write it into `@symptom` tags (e.g., lines of code). Another tool might analyze this information to infer quality defects (e.g., "long method") that are recorded in `@defect` tags. Finally, a tool might offer refactorings to a developer or a maintainer during his work and note applied refactorings or rationales in explicit `@refactoring` tags.

Developers and maintainers of a software system are supported in the handling of quality defects in the following activities:

- Repetitive refactoring of a specific kind of quality defect (e.g., "large method"), as they do not have to switch between different defects or refactoring concepts.
- Reuse of knowledge about the refactoring of specific quality defects to judge new quality defects.
- Recapitulation of the change history of the code fragment to update software documentation such as design documents.
- Retrieval of information about persons who developed or refactored this part of the system and should know about its purpose and functionality.
- Product or quality managers of the software system might use the information to:
- Evaluate the quality based on information extracted via the tags about the amount or distribution of quality defects.
- Analyze specific dates or groups of persons that might have introduced specific kinds of quality defects and might need further training.

SUMMARY AND OUTLOOK

Agile software development methods were invented to minimize the risk of developing low-quality software systems with rigid process-based methods. They impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. To assure quality, agile software development organizations use activities such as refactoring between development iterations. *Refactoring*, or the restructuring of a software system without changing its behavior, is necessary to remove *quality defects* (i.e., bad smells in code, architecture smells, anti-patterns, design flaws, software anomalies, etc.) that are introduced by quick and often unsystematic development. However, the effort for the manual discovery of these quality defects results in either incomplete or costly refactoring phases. Furthermore, software quality assurance methods seem to ignore their recurring application.

In this chapter, we described a process for the recurring and sustainable discovery, handling, and treatment of quality defects in software systems.

We described the complexity of the discovery and handling of quality defects in object-oriented source code to support the software refactoring process. Based on the formal definition of quality defects, we gave examples of how to support the recurring and sustainable handling of quality defects. The annotation language presented is used to store information about quality defects found in source code and represents the defect and treatment history of a part of a software system. The process and annotation language can not only be used to support quality defect discovery processes, but also has the potential to be applied in testing and inspection processes.

Recapitulating, we specified an annotation language that can be used in agile software maintenance and refactoring to record information about quality defects, refactorings, and rationales about them. Similar annotation languages such as JavaDoc or doxygen as well as third party extensions are not able to encode this information in a machine-readable and unambiguous format.

The proposed framework including the handling process promises systematic and semi-automatic support of refactoring activities for developers, maintainers, and quality managers. The approach for recording quality defects and code transformations in order to monitor refactoring activities will make maintenance activities simpler and increase overall software quality. Likewise, the user monitors daily builds of the software to detect code smells, identical quality defects, or groups thereof, and initiates repetitive refactoring activities, minimizing effort caused by task switches.

REQUIREMENTS FOR QUALITY DEFECT HANDLING IN AGILE SE

When building systems and languages for quality defect handling in agile software development, several requirements should be kept in mind.

The annotation language in the form of a code annotation language like JavaDoc or in the form of an external documentation such as a Defect Tracking system or a Wiki should be integrated into the programming language used and into the development environment. If it is not integrated, the information might easily be lost due to the high workload and time constraints in agile development. Especially in an agile environment, the developers, testers, and maintainers should be burdened with as little additional effort as possible.

Therefore, the more formal the descriptions of an annotation language are and the more information can be extracted from the code and development environment (e.g., from the refactoring techniques), the less information is required from the developers.

OUTLOOK

The trend in research is to increase automation of the mentioned processes in order to support the developers with automated refactoring or defect discovery systems.

We expect to further assist software engineers and managers in their work and in decision making. One current research task is the development of taglets and doclets to generate specific evolution documents. Furthermore, we are working on the analysis and synthesis of discovery techniques with statistical and analytical methods based on textual, structural, numerical, and historical information.

Although we can record and use this information in several applications, we currently do not know if the amount of information might overwhelm or annoy the developer and maintainer. If dozens of quality defects are found and additional refactorings are recorded, this might be confusing and should be hidden (e.g., in an editor of the IDE) from the developer. Very old

information (e.g., from previous releases of the software) might even be removed and stored in an external document or database.

REFERENCES

Allen, E. (2002). *Bug patterns in Java*. New York; Berkeley, CA: Apress.

Atlassian. (2005). *JIRA Web site*. Retrieved October 6, 2005, from http://www.atlassian.com/software/jira/

Aurum, A., Petersson, H., & Wohlin, C. (2002). State-of-the-art: Software inspections after 25 years. Software testing. *Verification and Reliability, 12*(3), 133-154.

Basili, V. R., Caldiera, G., & Rombach, D. (1994a). The goal question metric approach. In *Encyclopedia of software engineering* (1st ed., pp. 528-532). New York: John Wiley & Son.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994b). *Experience factory.* In J. J. Marciniak (Ed.), *Encyclopedia of software engineering* (Vol. 1, pp. 469-476). New York: John Wiley & Sons.

Beck, K. (1999). *eXtreme programming eXplained: Embrace change*. Reading, MA: Addison-Wesley.

Beck, K., & Fowler, M. (1999). *Bad smells in code*. In G. Booch, I. Jacobson, & J. Rumbaugh (Eds.), Refactoring: Improving the design of existing code (1st ed., pp. 75-88). Addison-Wesley Object Technology Series.

Brown, W. J., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). *AntiPatterns: Refactoring software, architectures, and projects in crisis.* New York: John Wiley & Sons, Inc.

Brykczynski, B. (1999). A survey of software inspection checklists. *Software Engineering Notes, 24*(1), 82-89.

Chillarege, R. (1996). Orthogonal defect classification. In M. R. Lyu (Ed.), *Handbook of software reliability engineering* (pp. xxv, 850 p.). New York: IEEE Computer Society Press.

Ciolkowski, M., Laitenberger, O., Rombach, D., Shull, F., & Perry, D. (2002). *Software inspections, reviews, and walkthroughs.* Paper presented at the 24th International Conference on Software Engineering (ICSE 2002), New York, USA, Soc.

Dromey, R. G. (1996). Cornering the chimera. *IEEE Software, 13*(1), 33-43.

Dromey, R. G. (2003). Software quality—Prevention versus cure? *Software Quality Journal, 11*(3), 197-210.

ePyDoc. (2005). *Epydoc Web site*. Retrieved May 10, 2005, from http://epydoc.sourceforge.net/

Fenton, N. E., & Neil, M. (1999). Software metrics: Successes, failures, and new directions. *Journal of Systems and Software, 47*(2-3), 149-157.

Force10. (2005). *Software support management system (SSM) Web site.* Retrieved October 6, 2005, from http://www.f10software.com/

Fowler, M. (1999). *Refactoring: Improving the design of existing code* (1st ed.). Addison-Wesley.

Freimut, B. (2001). *Developing and using defect classification schemes* (Technical Report No. IESE-Report No. 072.01/E). Kaiserslautern: Fraunhofer IESE.

Fukui, S. (2002). *Introduction of the software configuration management team and defect tracking system for global distributed development.* Paper presented at the 7th European Conference on Software Quality (ECSQ 2002), Helsinki, Finland, June 9-13, 2002.

Gamma, E., Richard, H., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software* (3rd ed., Vol. 5). Addison-Wesley.

Hallum, A. M. (2002). *Documenting patterns.* Unpublished Master Thesis, Norges Teknisk-Naturvitenskapelige Universitet.

IEEE-1044. (1995). IEEE guide to classification for software anomalies. IEEE Std 1044.1-1995.

Johnson, J. N., & Dubois, P. F. (2003). Issue tracking. *Computing in Science & Engineering, 5*(6), 717, November-December.

JSR-260. (2005). *Javadoc Tag Technology Update (JSR-260).* Retrieved October 6, 2005, from http://www.jcp.org/en/jsr/detail?id=260

Kerievsky, J. (2005). *Refactoring to patterns.* Boston: Addison-Wesley.

Koru, A. G., & Tian, J. (2004). Defect handling in medium and large open source projects. *IEEE Software, 21*(4), 54-61.

Kramer, D. (1999, September 12-14). *API documentation from source code comments: A case study of Javadoc.* Paper presented at the 17th International Conference on Computer Documentation (SIGDOC 99), New Orleans, LA.

Kramer, R. (1998). iContract—The Java(tm) Design by Contract(tm) Tool. In *Technology of object-oriented languages and systems, TOOLS 26* (pp. 295-307). Santa Barbara, CA: IEEE Computer Society.

Lauesen, S., & Younessi, H. (1998). Is software quality visible in the code? *IEEE Software, 15*(4), 69-73.

Liggesmeyer, P. (2003). Testing safety-critical software in theory and practice: A summary. *IT Information Technology, 45*(1), 39-45.

Loper, E. (2004). *Epydoc: API documentation extraction in python.* Retrieved from http://epydoc.sourceforge.net/pycon-epydoc.pdf

Mantis. (2005). *Mantis Web site*. Retrieved October 6, 2005, from http://www.mantisbt.org/

Marinescu, R. (2004, September 11-14). *Detection strategies: Metrics-based rules for detecting design flaws.* Paper presented at the 20th International Conference on Software Maintenance, Chicago, IL.

Mason, J. (2005). From e-learning to e-knowledge. In M. Rao (Ed.), *Knowledge management tools and techniques* (pp. 320-328). London: Elsevier.

Mens, T., Demeyer, S., Du Bois, B., Stenten, H., & Van Gorp, P. (2003). Refactoring: Current research and future trends. *Electronic Notes in Theoretical Computer Science, 82*(3), 17.

Mens, T., & Tourwe, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering, 30*(2), 126-139.

MetaQuest. (2005). *Census Web site*. Retrieved October 6, 2005, from http://www.metaquest.com/Solutions/BugTracking/BugTracking.html

Pepper, D., Moreau, O., & Hennion, G. (2005, April 11-12). *Inline automated defect classification: A novel approach to defect management.* Paper presented at the IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop, Munich, Germany.

Rapu, D., Ducasse, S., Girba, T., & Marinescu, R. (2004). *Using history information to improve design flaws detection.* Paper presented at the 8th European Conference on Software Maintenance and Reengineering, Tampere, Finland.

Ras, E., Avram, G., Waterson, P., & Weibelzahl, S. (2005). Using Weblogs for knowledge sharing and learning in information spaces. *Journal of Universal Computer Science, 11*(3), 394-409.

Rech, J. (2004). *Towards knowledge discovery in software repositories to support refactoring.* Paper presented at the Workshop on Knowledge Oriented Maintenance (KOM) at SEKE 2004, Banff, Canada.

Rech, J., & Ras, E. (2004). *Experience-based refactoring for goal-oriented software quality improvement.* Paper presented at the 1st International Workshop on Software Quality (SOQUA 2004), Erfurt, Germany.

Remillard, J. (2005). Source code review systems. *IEEE Software, 22*(1), 74-77.

Riel, A. J. (1996). *Object-oriented design heuristics.* Reading, MA: Addison-Wesley.

Roock, S., & Havenstein, A. (2002). *Refactoring tags for automatic refactoring of framework dependent applications.* Paper presented at the Extreme Programming Conference XP 2002, Villasimius, Cagliari, Italy.

Roock, S., & Lippert, M. (2004). *Refactorings in großen Softwareprojekten: Komplexe Restrukturierungen erfolgreich durchführen (in German).* Heidelberg: dpunkt Verlag.

Roock, S., & Lippert, M. (2005). *Refactoring in large software projects.* John Wiley & Sons.

Sametinger, J., & Riebisch, M. (2002, March 11-13). *Evolution support by homogeneously documenting patterns, aspects, and traces.* Paper presented at the 6th European Conference on Software Maintenance and Reengineering, Budapest, Hungary.

Serrano, N., & Ciordia, I. (2005). Bugzilla, ITracker, and other bug trackers. *IEEE Software, 22*(2), 11-13.

Simonis, V., & Weiss, R. (2003, July 9-12). *PROGDOC—a new program documentation system.* Paper presented at the 5th International Andrei Ershov Memorial Conference (PSI 2003) Perspectives of System Informatics, Novosibirsk, Russia.

Tigris. (2005). *Scarab Web site.* Retrieved October 6, 2005, from http://scarab.tigris.org/

Torchiano, M. (2002, October 3-6). *Documenting pattern use in Java programs.* Paper presented at the Proceedings of the International Conference on Software Maintenance (ICSM), Montreal, Que., Canada.

Tourwe, T., & Mens, T. (2003). Identifying refactoring opportunities using logic meta programming. IEEE Computer, Reengineering Forum; Univ. Sannio. In *Proceedings 7th European Conference on Software Maintenance and Reengineering,* Los Alamitos, CA (xi+2420 2091-2100). IEEE Comput. Soc.

TRAC. (2005). *TRAC Web site.* Retrieved October 6, 2005, from http://projects.edgewall.com/trac/

Tullmann, P. (2002). *Pat's taglet collection*. Retrieved October 6, 2005, from http://www.tullmann.org/pat/taglets/

Tuppas. (2005). *Tuppas Web site*. Retrieved October 6, 2005, from http://www.tuppas.com/Defects.htm

van Heesch, D. (2005). *Doxygen—a documentation system.* Retrieved from http://www.doxygen.org/

Verhoef, C. (2000, September 5-7). *How to implement the future?* Paper presented at the Proceedings of the 26th EUROMICRO Conference (EUROMICRO2000), Maastricht, The Netherlands.

Wake, W. C. (2003). *Refactoring workbook* (1st ed.). Pearson Education.

Whitmire, S. A. (1997). *Object-oriented design measurement.* New York: John Wiley & Sons.

This work was previously published in An Overview of Knowledge Management, edited by J.N.D. Gupta, S.K. Sharma, and J. Hsu, pp. 90-113, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 1.21
Different Views of Software Quality

Bernard Wong
University of Technology Sydney, Australia

ABSTRACT

This chapter examines the different definitions of quality and compares the different models and frameworks for software quality evaluation. It will look at both historical and current literature. The chapter will give special attention to recent research on the Software Evaluation Framework, a framework for software evaluation, which gives the rationale for the choice of characteristics used in software quality evaluation, supplies the underpinning explanation for the multiple views of quality, and describes the areas of motivation behind software quality evaluation. The framework has its theoretical foundations on value-chain models, found in the disciplines of cognitive psychology and consumer research, and introduces the use of cognitive structures as a means of describing the many definitions of quality. The author hopes that this chapter will give researchers and practitioners a better understanding of the different views of software quality, why there are differences, and how to represent these differences.

INTRODUCTION

Adopting an appropriate Quality Assurance philosophy has been often viewed as the means of improving productivity and software quality (Hatton, 1993; Myers, 1993). However unless quality is defined, it is very difficult for an organization to know whether it has achieved quality clearly. To date, this has usually involved conformance to a standard such as AS3563 or ISO9001 or following the Capability Maturity Model of the SEI. The challenge often faced is that one finds as many definitions of quality as writers on the subject. Perhaps, the latter have been remarkably few in number considering the obvious importance of the concept and the frequent appearance of the term quality in everyday language.

Though the topic of software quality has been around for decades, software product quality research is still relatively immature, and today it is still difficult for a user to compare software quality across products. Researchers are still not clear as to what is a good measure of software

quality because of the variety of interpretations of the meaning of quality, of the meanings of terms to describe its aspects, of criteria for including or excluding aspects in a model of software, and of the degree to which software development procedures should be included in the definition. A particularly important distinction is between what represents quality for the user and what represents quality for the developer of a software product.

Perceptions of software quality are generally formed on the basis of an array of cues. Most notably, these cues include product characteristics (Boehm et al., 1976; Carpenter & Murine, 1984; Cavano & McCall, 1978; McCall et al., 1977; Kitchenham & Pfleeger, 1996; Kitchenham & Walker, 1986; Sunazuka et al., 1985). The cues are often categorized as either extrinsic or intrinsic to the perceived quality. Simply, intrinsic cues refer to product characteristics that cannot be changed or manipulated without also changing the physical characteristics of the product itself; extrinsic cues are characteristics that are not part of the product (Olson & Jacoby, 1972). Price and brand are thus considered to be extrinsic with respect to product quality.

This chapter examines the different definitions of quality and compares the different models and frameworks for software quality evaluation. This chapter will address both the topics of interest for the information systems community and the software engineering community. It will look at both historical and current literature. The chapter will give special attention to recent research on the Software Evaluation Framework, a framework for software evaluation, which gives the rationale for the choice of characteristics used in software quality evaluation, supplies the underpinning explanation for the multiple views of quality, and describes the areas of motivation behind software quality evaluation. The framework has its theoretical foundations on value-chain models, found in the disciplines of cognitive psychology and consumer research, and introduces the use of cognitive structures as a means of describing the many definitions of quality.

BACKGROUND

Software users today are demanding higher quality than ever before, and many of them are willing to pay a higher price for better quality software products. The issue of software quality has come to the forefront in Europe, the United Kingdom, the United States, and more recently Australia. The quality movement in software is not new. A search of the information systems literature has shown that attempts to achieve quality software have been on-going for many years. Software quality models include the product-based view (Boehm et al., 1976; Carpenter & Murine, 1984; Cavano & McCall, 1978; McCall et al., 1977; Kitchenham & Pfleeger, 1996; Kitchenham & Walker, 1986; Sunazuka et al., 1985), process focused models following a manufacturing-based view (Coallier, 1994; Dowson, 1993; Humphrey, 1988; Ould, 1992; Paulk, 1991), and more recently, techniques and tools to cater for the user-based view (Delen & Rijsenbrij, 1992; Erikkson & McFadden, 1993; Juliff, 1994; Kitchenham, 1987; Kitchenham & Pickard, 1987; Thompsett, 1993; Vidgen et al., 1994). However, the many models and approaches seem to contradict each other at times. Garvin (1984) tries to explain these contradictions by introducing different views of quality. He describes the models as transcendental-based view, product-based view, manufacturing-based view, economic-based view, and user-based view, which we will define later.

As the software market matures, users want to be assured of quality. They no longer accept the claims of the IT department at face value, but expect demonstrations of quality. There is a firm belief that an effective quality system leads to increased productivity and permanently reduced costs, because it enables management

to reduce defect correction costs by emphasizing prevention. A better-designed development process will accrue fewer lifetime costs, and higher productivity will therefore be possible. As such, many attempts at improving quality have been to focus on the development processes. However, productivity measurements are worthless unless they are substantiated by quality measurements. To develop software quickly, on time, and within budget is no good if the product developed is full of defects. It is the view of many that the costs of correcting defects in software late in development can be orders of magnitude greater than the cost of correcting them early (Kitchenham & Pfleeger, 1996; Pfleeger, 2001). Preventing defects in the first place can save even more. Productivity measurements require quality measurements.

The market for software is increasingly a global one, and organizations will find it increasingly difficult to succeed and compete in that market unless they produce and are seen to produce quality products and services. From simple issues concerning efforts required to develop software systems, there have been a myriad of developments and directions that have occurred since the beginning of the last decade concerning the process of software development. From all of these new directions, the consideration of the quality of the software produced is one of the major thrusts that have occurred. Software used to be a technical business, in which functionality was the key determinant of success. Today, one can no longer just adopt this view.

THE MEANING OF QUALITY

The Oxford English Dictionary (OED, 1990) states that quality is the "degree of excellence." While this definition of quality is found in an internationally accepted dictionary, it must be pointed out that there are many variations to this definition, which are not given here.

A formal definition of quality is provided by the International Standards Organization (ISO, 1986): *The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs.*

This standard definition associates quality with the products' or services' ability to fulfill its function. It recognizes that this is achieved through the features and characteristics of the product. Quality is associated both with having the required range of attributes and with achieving satisfactory performance in each attribute.

The best-known and most widely adopted definition of quality is simply "fitness for use" or some variant thereof. For example, Wimmer (1975) and Genth (1981) adopted the definition "fitness for use." Box (1984) defined quality as "the degree to which a product fulfils its functions, given the needs of the consumer," and Kotler (1984) speaks of "the rated ability of the brand to perform its functions as perceived by consumers." Kawlath (1969) defined perceived quality as "the fitness for certain goals."

Maynes (1976) proposed the following definition: "The quality of a specimen (a product/brand/seller/combination) consists of the extent to which the specimen provides the service characteristics that the individual consumer desires." A similar definition is suggested by Monroe and Krishnan (1985): "Perceived product quality is the perceived ability of a product to provide satisfaction relative to the available alternatives." Kuehn and Day (1962) stated that the quality of a product depends on how well it fits in with patterns of consumer preferences.

Kupsch and Hufschmied (1978) regarded quality as a bundle of need-satisfying attributes. A similar position was taken by Oxenfeldt (1950). Bockenhoff and Hamm (1983) defined quality as the composite of all product attributes irrespective of whether these attributes are in reality existent in the product and objectively measurable, and whether consumers are correct in their evaluations.

Thurstone (1985) suggested: "Quality is the index that reflects the extent to which the customer feels that his need, the product, and his expectations for that product overlap." He concluded that the relevant measure of quality does not reside in the product but in the customer. A similar position is taken by Wolff (1986) who argued that quality should be measured from the customer's perspective: "If the customer says it's good, it's good; if he says it's bad, it's bad."

Trenkle (1983) distinguished three manifestations of quality:

- neutral concept (i.e., "much quality" - "not much quality"), defined as the nature of a product, given by the whole of all the attributes which discriminates the product from the other products in the same category;
- evaluative concept ("good quality" - "bad quality"), defined as the fitness for use of a product, given by the whole of all the attributes that are relevant to the evaluation of the product;
- positive judgment ("quality products"), defined as superior or excellent with respect to all attributes.

Some researchers have followed Wittgenstein's (1953) linguistic approach that "the meaning of a word is its use in the language" and explored the word quality in terms of everyday use of the term. For instance, quality often means reliability, general approval, or general excellence in the eyes of the consumers (Holbrook & Corfman, 1983). The ordinary language approach, however, does not get one very far conceptually with respect to the meaning of perceived quality.

The Different Perspectives of Quality

There are numerous definitions and meanings given to quality. The previous section lists some of the popular definitions; however, they do not necessarily describe the perceptions of quality held by all individuals. David Garvin (1984) concluded in his paper "that quality is a complex and multifaceted concept." He described quality as being made up of five different perspectives — the transcendental-based view, the product-based view, the manufacturing-based view, the economics-based view, and the user-based view.

Transcendental-Based View

According to the transcendental-based view, quality is synonymous with "innate excellence." It is both absolute and universally recognizable, a mark of uncompromising standards and high achievement. Nevertheless, proponents of this view claim that quality cannot be defined precisely; rather, it is a simple, not analyzable property that we learn to recognize only through experience. This definition borrows heavily from Plato's discussion of beauty, where he argues that beauty is one of the "platonic forms," and, therefore, a term that cannot be defined. Like other such terms that philosophers consider to be "logically primitive," beauty (and perhaps quality as well) can be understood only after one is exposed to a succession of objects that display its characteristics.

Product-Based View

Product-based definitions are quite different; they view quality as a precise and measurable variable. According to this view, differences in quality reflect differences in the quantity of some ingredient or attribute possessed by a product. For example, high quality ice cream has high butterfat content, just as fine rugs have a large number of knots per square inch. This approach lends a vertical or hierarchical dimension to quality, for goods can be ranked according to the amount of the desired attribute they possess. However, an unambiguous ranking is possible only if virtually all buyers consider the attributes in question preferable.

Product-based definitions of quality first appeared in the economics literature, where they were quickly incorporated into theoretical models. In fact, the early economic research on quality focused almost exclusively on durability, simply because it was so easily translated in the above framework. Since durable goods provide a stream of services over time, increased durability implies a longer stream of services — in effect, more of the good. With regard to software, durability can be understood as the length of time a piece of software is used in a live environment. Of course, one recognizes that the software may need upgrades from time to time, as products can require repairs or modifications. Quality differences could, therefore, be treated as differences in quantity, considerably simplifying the mathematics.

There are two obvious corollaries to this approach. First, higher quality can only be obtained at higher cost. Because quality reflects the quantity of attributes that a product produces, and because attributes are considered to be costly to produce, higher quality goods will be more expensive. Second, quality is viewed as an inherent characteristic of goods, rather than as something ascribed to them. Because quality reflects the presence or absence of measurable product attributes, it can be assessed objectively and is based on more than preferences alone.

User-Based View

Garvin (1984) states that the user-based definitions start from the opposite premise that quality "lies in the eyes of the beholder." Individual consumers are assumed to have different wants or needs, and those goods that best satisfy their preferences are those that they regard as having the highest quality. This is an idiosyncratic and personal view of quality and one that is highly subjective. In the marketing literature, it has led to the notion of "ideal points:" precise combinations of product attributes that provide the greatest satisfaction to a specified consumer; in the economics literature, to the view that quality differences are captured by shifts in a product's demand curve and in the operations management literature, to the concepts of "fitness for use." There are two problems in each of these concepts. The first is practical — how to aggregate widely varying individual preferences so that it leads to meaningful definitions of quality at the market level. The second is more fundamental — how to distinguish those product attributes that connote quality from those that simply maximize consumer satisfaction.

A more basic problem with the user-based view is its equation of quality with maximum satisfaction. While the two are related, they are by no means identical. A product that maximizes satisfaction is certainly preferable to one that meets fewer needs, but is it necessarily better as well? The implied equivalence often breaks down in practice. A consumer may enjoy a particular brand because of its unusual features, yet may still regard some other brand as being of higher quality. In the latter assessment, the product's objective characteristics are also considered.

Even perfectly objective characteristics, however, are open to varying interpretations. Today, durability is regarded as an important element of quality. Long-lived products are generally preferred to those that wear out more quickly. This was not always true until the late nineteenth century; durable goods were primarily possessions of the poor, for only wealthy individuals could afford delicate products that required frequent replacement or repair. The result was a long-standing association between durability and inferior quality, a view that changed only with the mass production of luxury items made possible by the Industrial Revolution.

Manufacturing-Based View

Manufacturing-based definitions focus on the supply side of the equation and are primarily concerned with engineering and manufactur-

ing practice. Virtually, all manufacturing-based definitions identify quality as "conformance to requirements." Once a design or a specification has been established, any deviation implies that the specification may not be complete. Excellence is equated with meeting specifications and with "making it right the first time."

While this approach recognizes the consumer's interest in quality, its primary focus is internal. Quality is defined in a manner that simplifies engineering and production control. On the design side, this has led to an emphasis on reliability engineering (Boehm, 1963; Garvin, 1984); on the manufacturing side, to an emphasis on statistical quality control (Garvin, 1984; Juran & Gryna, 1980). Both techniques are designed to weed out deviations early: the former, by analyzing a product's basic components, identifying possible failure modes, and then proposing alternative designs to enhance reliability; and the latter, by employing statistical techniques to discover when a production process is performing outside acceptable limits.

Each of these techniques is focused on the same end: cost reduction. According to the manufacturing-based approach, improvements in quality (which are equivalent to reductions in the number of deviations) lead to lower costs, for preventing defects is viewed as less expensive than repairing or reworking them (Crosby, 1978; Erikkson & McFadden, 1993; Garvin, 1984). Organizations are, therefore, assumed to be performing suboptimally: were they only to increase their expenditures on prevention and inspection — testing prototypes more carefully or weeding out a larger number of defective components before they become part of fully assembled units — they would find their rework, scrap, and warranty expenses falling by an even greater amount.

Economic-Based View

Economic-based definitions define quality in terms of costs and prices. According to this view, a quality product is one that provides performance at an acceptable price or conformance at an acceptable cost. Under this approach, a $500 running shoe, no matter how well constructed, could not be a quality product, for it would find few buyers.

Garvin shows how each of these views can be used to define product quality. Engineers who believe a product has set characteristics often adopt the product-based view. These characteristics are used as the measure of quality. The manufacturing-based view is adopted when one believes the quality development process determines a quality product. And more recently, many organizations have been certified with the Capability Maturity Model, CMM, or SPICE (Paulk, 1991; Paulk et al., 1993). Certification assesses the manufacturing process and is awarded on successfully having a quality management system in place. Economists who believe that price has a correlation with quality adopt the economics-based view. And lastly, the user-based view is the one which emphasizes that each individual will have his or her own perception of quality. Garvin (1984) states that most existing definitions of quality fall into one of these categories, be they conformance to specification, meeting user requirements, or best development practice.

In a more recent paper, Braa and Ogrim (1994) also tried to expand the definition of quality. Rather than just focusing on technical properties, they considered the functional and organizational aspects of quality. Five aspects of quality are introduced.

Technical Quality

Technical quality refers to a system's structure and performance. The technical quality of a computer system is the basis of its functionality — the computer must perform expected operations. However, there is no need for a technically excellent computer system; it is more important that it suits the work tasks it is supposed to support.

Use Quality

By use quality we mean quality as experienced by the users when working with the computer-based system. Use quality is difficult to specify in advance. It is often necessary to experiment with design models, such as prototypes, to express needs and claims (Greenbaum & Kyng, 1991). Throughout the specification and design activities, users and developers will then learn about the limitations and the possibilities of the computer system in use. Consequently, learning is an important factor in the specification process, as well as in design.

Aesthetic Quality

In many other disciplines, such as the car industry, or building industry, aesthetic quality is used to evaluate the quality of artifacts. The idea of aesthetics is usually related to physical objects. However, aesthetics can also be applied to immaterial objects, for example, the notion of elegant proofs in mathematics. However, the aesthetic perspective is almost always neglected in software development (Dahlborn & Mathiassen, 1993; Stolterman, 1991). One possible exception is the design of user interfaces. An aspect of aesthetic quality is "elegance," introduced as a criterion by which to assess quality (Checkland & Scholes, 1990). An attempt to increase aesthetic quality and make it more visible is to raise questions such as: Is the transformation well designed? Is it aesthetically pleasant? Is it overcomplicated? Is it over- or under-engineered? These assessments allow the users' subjective experiences as well as the professionals' experience of similar systems to be included in the judgment of quality.

Symbolic Quality

Computer systems are not only artifacts, they are also used as symbols in the organization. This view is supported by Feldman and March's (1981) study of the use of information in organizations. They found that information is often used symbolically, e.g., signaling a well-driven organization, independent of whether the information is used or not. Symbolic, as well as aesthetic, aspects of computer systems are important factors in tailoring a system to the business philosophy and organization culture. For example, an organization wishing to have an innovative and modern image should have computer systems with graphical user interfaces and colors. Symbolic quality may be contradictory to the use quality. This can be illustrated by a hotel owner who wanted to install a computerized reception system. It turned out that the main purpose of the system was to create the impression of a modern, successfully run hotel and to recruit educated personnel. Not much attention was paid to the functionality of the system. If the motivation is of a symbolic character, as in this example, the use quality will probably be ignored.

Organizational Quality

When a computer system is well adapted to the organization, it can be said to be of high organizational quality. When assessing the quality of information systems, questions of economy, power and interests will arise sooner or later: Are the computer systems developed for the interests of individual users, for groups of users, or for the organization as a whole? Different user groups may have diverging, perhaps contradictory, ideas of what signifies good quality. A personnel control and wage system might represent good use quality for management and the personnel department, but not for the workers who are being controlled. A medical journal system in a hospital may be of good quality for the doctors, but at the same time decrease the influence of the nurses and use of their competence. This may, in turn, decrease the quality if their work. The different groups of users make different demands on functionality, in order to support their work. If these different

interests are not met, an intersection of several interests is developed and the result could be a computer system, which is not suited for anyone. This, in turn, may decrease the organizational quality of the system.

Hyatt and Rosenberg (1996) add another quality perspective, the project manager's view. The project manager views the software quality as "software that works well enough to serve its intended function and is available when needed". Project managers are usually concerned in producing software that is reliable, maintainable, and keeps customers satisfied. They usually face budget and/or time constraints, which will impede either one of their quality goals. The constraints will potentially sacrifice the testing (level, extent, and depth), walkthrough inspections, and documentation, which are some of the factors contributing to software quality. While Hyatt and Rosenberg focus only on the view of the project manager, it must be pointed out that many definitions of quality share this same view (Box, 1984; Genth, 1981; Wimmer, 1975).

The Software Perspective of Quality

Researchers in the software engineering area have tried different ways to define quality. In fact, they have moved progressively from the product-based view, to the manufacturing-based view, and more recently to the user-based view. They have also focused heavily on technical quality with a growing consideration for use quality and aesthetic quality. However, no models in software engineering seem to exist following the transcendental, economic, aesthetic, symbolic, or organizational based views of quality.

Table 1 presents some of the software quality models introduced in this chapter. There are other quality models; however, these are the most discussed models in the literature. These are only the most notable among many in research. At first glance, the list of software quality models gleaned from a review of literature is daunting. One might conclude that they are many and varied; however, many of these models are developments of earlier ones, and some are combinations of others with

Table 1. Various software quality models and their relationship to Garvin's and Braa's quality perspectives

Software quality models	Garvin's quality perspective	Braa's quality perspective
McCall	Product view	Technical Quality
Boehm	Product view	Technical Quality
Gilb	Product view	Technical Quality
Murine	Product view	Technical Quality
Schmitz	Product view	Technical Quality
Schweiggert	Product view	Technical Quality
Willmer	Product view	Technical Quality
Hausen	Product view	Technical Quality
ISO9000-3	User view	Technical Quality
ISO9126	User view, product view	Use Quality
Dromey	Product view	Technical Quality
COQUAMO	User, manufacturer, product view	Technical Quality, Use Quality
Squid	User, manufacturer, product view	Technical Quality, Use Quality
CMM	Manufacturer view	Technical Quality
Bootstrap	Manufacturer view	Technical Quality
Trillium	Manufacturer view	Technical Quality
MMM	Manufacturer view	Technical Quality
Scope	Manufacturer view	Technical Quality
Processus	Manufacturer view	Technical Quality
SPICE	Manufacturer view	Technical Quality
QFD	User, manufacturer, product view	Technical Quality, Use Quality

and without uniqueness. For example, ISO quality factors are very similar to those in the earlier McCall and Boehm models. Bootstrap, Trillium, and MMM are all evolutions of CMM. SPICE is an evolution of Bootstrap, CMM, trillium, and ISO9001. SCOPE and PROCESSUS are efforts at combining process and product through connecting ISO9001 and ISO9126. And SQUID uses elements of ISO9126 and McCall. In most cases, the reworking can be considered one researcher's improvements on another's. This leaves few originals. Shari Pfleeger (1997) states that since models illustrate the relationship between apparently separate events, the lack of real variety in models in software engineering is symptomatic of a lack of system focus.

The research on software quality indicates that it is divided into two main schools of thought. There is the Product school, where practitioners advocate that to clearly define, measure, and improve quality, one must measure characteristics of the software product which best influence quality. Then there is the Process school, where practitioners advocate that one may be unable to properly define quality, but states that a product's quality is improved when the software engineering process used to make the product is improved. A process model's quality factors refer to the product development life cycle as a way of expressing the product's quality.

According to this product-process division, Boehm's model, the COQUAMO model, McCall's model, Murine's SQM model, ISO9126, SQUID, and Dromey's model are all examples of product models: they focus on the final product. The process model includes ISO9001-3, CMM, SPICE, BOOTSTRAP, Trillium, and MMM. SCOPE and PROCESSUS are a composite of both product and process models.

The Product Quality models are basically those models that view quality as inherent in the product itself. These models decompose the product quality into a number of quality factors (some models describe it as quality features, criteria, or characteristics). They are decomposed further into lower levels. The lowest are the metrics to measure or indicate the criteria. The metrics are used to assess the quality of the software. The result of the assessment is compared against a set of acceptance criteria. This determines whether the software can be released. Many of the software quality models in this group do not suggest what metric values are acceptable and hence leave it to the software practitioner to decide.

The Process Quality models are those models that emphasize process improvement and development standards, with the aim that the process improvements will lead to improving the quality of the product. In the past, much software development was performed without proper process and management control. The software developers were more guided by intuitive approach or habit, rather than by using any particular standards. Rae et al. (1995) emphasizes the importance of standardizing the development process and states that it will reduce software development to a rational and well-defined series of tasks, which can be managed and controlled (Rae et al., 1995). Watts Humphrey (1988), in introducing the CMM, stated that the premise of the CMM lies in the relationship between the quality of the process and the quality of the product. The belief is that effective process management will be the key factor in producing high quality software. However, Gillies (1992), Paulk et al. (1993), and Rae et al. (1995) indicated that the adoption of a good quality process will not automatically guarantee a good quality product.

Schneidewind (1996) and Fenton (1996) debated whether the existence of a standard process is good enough to improve the quality of software. Schneidewind's view is that there is a need for a standard first, even though it is not perfect. Fenton argued that a good standard process is required to improve the quality of a product and a bad standard will not help at all. In my opinion, a standard is necessary as a guideline. However, as resources to set up the standards are scarce,

they have to be initially established as well as possible. The standards have to be constantly reviewed and updated, because the technology and software engineering methodology change. Fenton suggested that organizations should contribute to the resources.

Voas (1997) criticizes the software process movement in general and claims that the focus of software process movement and methods on improving quality give the impression that "clean pipes can produce only clean water." He states that we must not forget the part to be played by product oriented methods, with their focus on software metrics and system level testing that allow the developing product to be sampled for its internal quality. He claims that the preoccupation with process quality in the software industry is partly a result of the difficulty of reliable testing. And software metrics, he adds, as do others such as Gulezian (1995), brings its own set of problems related to what exactly measures should be, as well as how and when they are best measured. For example, although software reliability assessment has been accepted as a measure of software quality, most reliability models predict failure by looking at error history (Voas, 1997). Unfortunately, this data can be computed differently by different models, so the reliability results may not even be reliable. Gulezian states that structural metrics cannot capture the dynamics of software behavior where an input is transformed into an output by a set of instructions. Thus we see from this direction that much of the future of software quality lies linked to research in metrics, and providing some acknowledgment of the roles of both product and process measurement in the quality of a piece of software. While much more can be said of the process models, focus in the rest of the chapter will be given to product quality.

Criticism of Models

The product models, particularly McCall's and Boehm's models (and the later ISO9126), have had similar criticisms aimed at them from multiple directions (Rozman et al., 1977). These criticisms have all stated that there is a lack of clear criteria for selection of not only the higher-level quality attributes but also of the subcharacteristics. There is a lack of consensus between researchers on what level a quality characteristic should be placed and for the apparent arbitrary placement of these characteristics, for the lack of agreement on terminology, and for the paucity of discussion of the measurement aspect of each of the models. An additional complication and criticism of McCall's and Boehm's models is that each of the quality subcharacteristics is not related to a single quality factor. This complicates measurement and interpretation of results. For example, McCall's model shows the quality factor *flexibility* influenced by quality characteristics of *self-descriptiveness, expandability, generality, modularity.* To further confound interpretation of any measurements that might have been gathered, these same quality subcharacteristics also influence the quality factors *portability, reusability, interoperability* as well as *testability* and *maintainability.* In ISO9126 each subcharacteristic influences only one high level quality factor — the model is completely hierarchical.

Gillies (1992) and Kitchenham and Pfleeger (1996) stated that there are a number of issues with the hierarchical models:

1. The selection of quality factors and criteria seem to be made arbitrarily. It implies that there is no rationale for determining:
    - which factors should be included in the quality model and
    - which criteria relate to a particular quality factor.
2. The hierarchical models cannot be tested or validated, because they do not define the quality factors in a measurable way.
3. The measurement methods of both models

have a large degree of subjectivity, hence the software quality cannot be measured objectively.

4. In 1987, Perry Gillies (1992) pointed out that some quality factors or criteria conflict with each other due to inverse relationships. For example, there is an inverse relationship between usability and efficiency. The improvements in human-computer interface will increase the usability of the software; however, it will lead to reduction in efficiency as more coding will be required.

Despite the criticism, both models (especially McCall's) form the basis for most of the research in software quality today.

Kitchenham and Walker (1986) made a number of observations. They stated that there seemed to be little supporting rationale for including or excluding any particular quality factor or criteria. In fact, definitions of quality factors and criteria were not always consistent when comparing one model to the next. Different terminologies between the models exist, with little explanation for the meanings behind them. There was no rationale for deciding which criteria relate to a particular factor. Thus the selection of quality characteristics and subcharacteristics can seem arbitrary. The lack of rationale makes it impossible to determine whether the model is a complete or consistent definition of quality.

Though quality has been widely used and accepted, the differences between them highlights how one's view can dictate the perception of quality. The degree of subjectivity varies substantially from one question to another, even though all responses are treated equally. This variation makes combining metrics difficult, if not impossible. Moreover, when appropriate, the detail and complexity of the response should be reflected in a richer measurement scale. For example, while it is reasonable to expect a yes or no response to the question "Does this module have a single exit or entry point?", questions about documentation clarity probably require a multiple-point ordinal scale to reflect the variety of possible answers.

The models recommend direct measures, but most of them do not describe clearly how to conduct the measure. There appears to be no description of how the lowest level metrics are composed into an overall assessment of higher leveled characteristics. In particular, then, there is no means for verifying that the chosen metrics affect the observed behavior of a factor. That is, there is no attempt to measure factors at the top of the hierarchy, so the models are not testable.

More Recent Developments of Product Models

Dromey's Model

A more recent and promising model of product quality is found in Dromey (1995), who strongly pushes for a more comprehensive software quality model. He holds that software does not manifest quality attributes; rather it exhibits product characteristics that imply or contribute to quality attributes. He states that most models fail to deal with the product characteristics. Dromey's model expresses the same internal view as the classic Boehm and McCall models, because he specifically aims at quality of software code (although he claims this can be applied equally well to other artifacts of the development cycle such as requirements specification or user interfaces). He states that the requirements of any product model are that it provides an adaptable methodical system for building quality into software, and it classifies software characteristics and quality defects. He proposes a bottom up/top down approach. In bottom up one begins with a set of structural forms whose quality carrying properties are identified and must be satisfied. These properties impact certain higher leveled quality attributes. This allows a bottom up movement by ensuring particular product properties are satisfied, for example, for programmers. In his top down appraisal, designers

begin with the quality attribute and move to the quality carrying property. In this way, for example, designers may identify properties that need to be satisfied for each structural form.

Dromey's model is broad enough to be applied across languages and projects — as long as it evolves as new languages evolve. Dromey provides a substantial contribution to the software quality field in that he has provided a complex analysis, which established direct lines between tangible product characteristics and quality attributes. However, his complete disregard for characteristics that can be evaluated externally could be argued to result in an excellent product that does not do what the customer has requested. The model also needs to be broadened to include the same principles applied to other aspects of the design process such as requirements specification.

The Dromey model has been developed to address many of the problems of the earlier models. Dromey (1995) criticized the earlier quality models for failing to deal adequately with the product characteristic problems and to link the quality factors/attributes to product characteristics. He points out that hierarchical models that use top-down decomposition are usually rather vague in their definitions of lower levels. They thus offer little help to software developers who need to build quality products. To overcome this, he proposed specific quality-carrying properties and systematic classification of quality defects. The quality-carrying properties are:

1. Correctness properties (minimal generic requirements for correctness)
2. Structural properties (low level, intramodule design issues)
3. Modularity properties (high level, intermodule design issues)
4. Descriptive properties (various forms of specification/documentation)

Correctness properties fall broadly into three categories that deal with computability, completeness, and consistency. Structural properties have focused upon the way individual statements and statement components are implemented and the way statements and statement blocks are composed, related to one another, and utilized. The modularity properties address the high-level design issues associated with modules and how they interface with the rest of the system. And the descriptive properties reflect how well the software is described. Explicit comments are added to a program to document how the implementation realizes its desired functionality by manipulating variables with prescribed properties.

Dromey believes that it is impossible to build high-level quality attributes such as reliability or maintainability into products. Rather, software engineers must build components that exhibit a consistent, harmonious and complete set of product properties that result in the manifestations of quality attributes.

Dromey's approach is important because it allows us to verify models. It establishes a criterion for including a particular software property in a model and a means of establishing when the model is incomplete. For example, each quality-carrying property has quality impact, for example, reliability and quality defects. By identifying the defects, the software developer will take measures to prevent the defect from occurring.

Kitchenham and Pfleeger (1996) noted that Dromey's quality model provides a constructive approach to engineering software quality which can be refined and improved further. Dromey even developed tools to assist in the production of quality software, a static analyzer to detect quality defects, and a programming language aimed at preventing developers from building defects into their programs.

The criticism of Dromey's model is that he does not use the metric approach in measuring

the software quality. This is because he uses the concept of defect classification. However, as Kitchenham and Pfleeger (1996) pointed out, the model is proof that there are other means to measure quality.

Constructive Quality Model (COQUAMO)

Kitchenham shares Dromey's concern for the need for precision in measurement out of the quality models. She has developed various models in her search to define and measure software quality. COQUAMO (Gulezian, 1995) was developed to further this research. It was the REQUEST (Reliability and Quality for European Software Technology) project which adapted Boehm's COCOMO model to use the more precise measurement of Rome Laboratory Software Quality Framework. (RLSQF provided a framework for using certain metrics for measuring 29 software oriented attributes and relating them to 13 user-oriented quality factors). The beauty of COQUAMO was that Kitchenham saw the need for measurement tools to vary from one environment to another. This was a substantial improvement on all research to date, which had been trying to increase the objective measurements for specific characteristics, without producing any results.

Kitchenham introduced the concept of a "quality profile" and made a distinction between subjective and objective quality measures. It is an attempt to integrate the user, manufacturer, and product views in one software quality model (Kitchenham & Pfleeger, 1996). The COQUAMO model was developed based on Gilb's model and Garvin's views of quality (Gillies, 1992).

The COQUAMO model consists of the following components:

1. **Transcendent Properties:** these are qualitative measures that are hard to measure. People have different views and definitions. An example is usability.
2. **Quality Factors:** these are system characteristics, which are made up of measurable factors of quality metrics and quality attributes. The quality factors themselves could be subjective or objective characteristics such as reliability and flexibility.
3. **Merit Indices:** these define the system subjectively. They are measured subjectively by quality ratings.

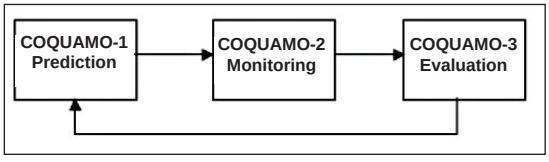
The objectives of COQUAMO and the relevant tools to assist the software developers are in the following table:

- to predict product quality
- to monitor progress toward quality
- to evaluate the results as feedback to improve prediction of the next project

The initial input to COQUAMO-l are estimates of the following quality drivers:

- product attributes such as quality requirements
- process attributes such as process maturity
- personnel attributes such as the developer's experience and motivation
- project attributes such the quality norm expected
- organizational attributes such as quality management

Figure 1. COQUAMO toolset diagram (Gillies, 1992)



The toolset of COQUAMO-2 consists of guidelines to monitor the progress toward quality product, while toolset COQUAMO-3 provides guidelines to assess the quality of the product and compare it against the prediction. The result of this will be input to COQUAMO-1 for the next project.

The distinctive characteristic of this model that differentiates it from previous models is that the evaluation and feedback processes are introduced here to improve the quality of software in the next project. There is a similarity with some aspects of Total Quality Management (TQM) in software quality modeling, that is, improvement of quality. This is featured by the COQUAMO-3 toolset.

The SQUID Product Model

The SQUID model, an ESPRIT 3 SQUID project (Kitchenham et al., 1997), was the result of a reassessment by Kitchenham of what constituted a quality model. It was developed after COQUAMO and after the results of the ESPRIT project concluded that there were no software product metrics that were likely to be good predictors of final product qualities. Kitchenham built SQUID based on these results and held firmly to the importance of monitoring and controlling internal measures.

Based on the McCall and ISO9126 models, SQUID aimed to use measurable properties to link internal properties and external characteristics to quality characteristics. A key principle to the SQUID model is that it must be customized for each project/product. Kitchenham states that the underlying principle of SQUID is that software quality requirements cannot be considered independently from a particular software product. Kitchenham acknowledges her alignment with Gilb (1996) on this point when he states that quality requirements arise from the operational properties required of the software, or of its support/maintenance environment. SQUID provides a concrete link between a concrete feature/component and the quality factor. The SQUID approach is to use measurement as the link between the product quality requirements and the quality characteristics (it uses ISO9126 characteristics as accepted results of research). In the application of the model to a product, concrete internal measures, with targets, are developed for the desired software properties, which reference quality characteristics. For example, the internal measures for the software property correctness were the number of faults detected, and the grouping into which the faults fell.

The SQUID model comes far closer than any in being able to balance the need for internal and external views and being able to consider the change in characteristics and metrics that are involved with each type of software in each context. The model's necessary broadness to suit this recognized need means that it takes a different form from the other models. There is no naming of quality factors and quality characteristics. The true strength of this model, as already stated, is its ability to be used in any project.

Other Approaches to Quality Evaluation

Quality Function Deployment (QFD) is a rigorous and high-structured process used in manufacturing. It provides a means of translating customer requirements into the appropriate technical requirements for each stage of product development and production. The process has been adopted in software engineering (Delen & Rijsenbrij, 1992; Erikkson & McFadden, 1993; Juliff, 1994; Thompsett, 1993). It required that before each software development project began, the appropriate characteristics along with the degree of importance of these characteristics had to be first identified by the stakeholders. The project quality would then be measured against these functions. Though QFD is very much a product definition process which helps businesses identify and integrate user needs and requirements to a product, Thompsett (1993) states that QFD is a tool which can help

define software quality. Erikkson and McFadden (1993) claim that this approach gives a good structure within which users' needs are taken into account and progressively refined to give appropriate product metrics. QFD has been described as a customer-driven system that attempts to get early coupling between the requirements of the customer and system designers. It has not only proven itself in manufacturing and service environments, but has been shown to be equally beneficial in software development (Erikkson & McFadden, 1993; Juliff, 1994; Thompsett, 1993). Erikkson & McFadden (1993) describe several benefits of QFD from their case study:

- QFD encourages focus on customer needs and helps to prioritize the requirements.
- QFD encourages developers to think in terms of defect prevention.
- QFD is a powerful communication vehicle that allows modeling of the discussions between customers, software designers, and other participants.
- QFD allows easy tracking of important customer requirements to the related software characteristics, product features, and product metrics.
- QFD provides an opportunity to follow the consequences in the process.
- As can be seen from this list, QFD is a powerful manufacturing process to ensure customer requirements are met. QFD differs from traditional models, in that its approach is to develop software products from the customers' perspective, not from the suppliers' perspective. While QFD supports the many views of quality, described by Garvin (1984), it does not try to understand the differences, nor does it try to identify similarities between customers.

Vidgen et al. (1994) proposed a framework to define quality based on the multiview development method (Wood, 1992; Wood-Harper & Avison, 1992). They believed that multiple perspectives of software quality are required if one is to assess product quality properly. The framework was based on customer satisfaction, relating the product with its use and the services provided to support it. The three views of IS quality provide the basis for the multiple perspective. It was not merely an exercise in looking at the same object from different angles, but it entailed different assumptions about what quality is. Though this framework was introduced, no further research followed.

Criticism of These Recent Models

Kitchenham acknowledges some problems with the SQUID model (Kitchenham et al., 1997). Using the ISO9126 model as a framework led her to make recommendations for on-going improvements to the ISO9126 standard. Among her recommendations is a request for a further broadening of the ISO9126 standard. Her reasons are good ones and may lead critics of the broadness of ISO9126 to rethink their reasons for seeking further specificity. She says that insofar as software products do not have a standard set of functional requirements, then it seems that rather than give a fixed set of quality requirements, the standards should specify how the quality characteristics should be defined by those involved in the project. From this generic outline, the development team would have guidance to develop their product model. Kitchenham also states that the lower level properties should be clearly classified as external or internal characteristics. She finally recommends that research should focus on ways to validate quality models. She acknowledges the contribution Dromey's (1995) research has made in providing a criterion for including a software property in a model. Dromey supports this need for a broad model to be developed by those involved in the project and states that it would involve the project team specifying the subcharacteristics of each quality characteristic of each component, establishing the

metrics that correlate to the characteristics of the software product and to the environment in which it is functioning, and to provide a rating level that expresses a degree of user satisfaction. Since each set of users and developers needs is different, each rating level will change. This is a complex task that would require much guidance.

Vollman (1993) provides additional comment on this issue of model broadness: not only is the subdivision of quality characteristics influenced by the evaluator's point of view and the product's context, but when this occurs, the algorithms applied when evaluating will need to differ. In 1993, the ISO9000 standards included a publication that was the precursor to this need for the measure to be applied by multiple evaluators in multiple situations, and stated that an organization should use whatever metrics it deems appropriate to its applications as long as the ISO9000 metrics methodology is followed. Schneidewind (1993) states that this broadness allows metrics to be used by developers, managers, quality assurance organizations, maintainers, and users. Clearly the newest metric for software quality measurement will receive much heated discussion as a result of the move to balance internal measures with measures of external product attributes.

From this section, it can be concluded that there have been vast improvements in defining software quality. Yet overall, software product quality research is still relatively immature, and today it is still difficult for a user to compare software quality across products. Researchers are still not clear as to what is a good measure of software quality because of the variety of interpretations of the meaning of quality, of the meanings of terms to describe its aspects, of criteria for including or excluding aspects in a model of software, and of the degree to which software development procedures should be included in the definition (Comerford, 1993). Although many of the criticisms of product models are valid, Voas (1997) proposes that because product-oriented methods are more focused on achieving quality than assessing it, they play a critical role in software quality research. He contends firmly that software behavior, irrespective of the software's development history, is what defines software quality. He sees organizations and research caught in a bind with software testing measuring the right thing but with insufficient precision and process measurement more accurately measuring the wrong thing.

MEASURING QUALITY WITH THE SOFTWARE EVALUATION FRAMEWORK

There have been many studies on the topic of software quality, yet little empirical studies on what influences the perception of quality for different people. Earlier research of Wong (Wong, 1998; Wong & Jeffery, 1995, 1996) concluded that different groups of people view quality in different ways and that it was possible to group people with similar definitions of quality and similar choices of characteristics in their quality assessment process.

During the past 30 years there have been many studies on the topic of software quality; however, there have been none on a framework for software quality, which considers the motivation behind the evaluation process, other than the earlier version of this framework introduced by Wong and Jeffery (2001). This framework is based on the notion that software evaluators are influenced by their job roles. This is supported by earlier studies (Wong, 1998; Wong & Jeffery, 1995, 1996) where stakeholders with different job roles were found to focus on different sets of software characteristics when evaluating software quality. What motivates these differences is found within the broader context of value, where studies have shown that values are a powerful force in influencing the behavior of individuals (Rokeach, 1968; Yankelovich, 1981).

The theoretical basis for developing such a framework was based on the theory found in cognitive psychology and adopted by Gutman's Means-End Chain Model (Bagozzi, 1997; Bagozzi & Dabholkar, 2000; Gutman, 1982, 1997; Valette-Florence, 1998), which posits that linkages between product characteristics, consequences produced through usage, and personal values of users underlie the decision-making process or, in this case, the software quality evaluation process. It is the aim of the framework to not only show the relationships between the characteristics and software quality, but also show that there are relationships between the characteristics and the desired consequences, and between the characteristics and the sought-after values.

Personal values are beliefs people have about important aspects of themselves and the goals toward which they are striving. Personal values are the penultimate consequences of behavior for people: their feelings of self-esteem, belonging, security, or other value orientations. As such, personal values are part of the central core of a person. That is, personal values determine which consequences are desired and which are undesired.

Values have been shown to be a powerful force in influencing the behaviors of individuals in all aspects of their lives (Rokeach, 1968; Yankelovich, 1981). It is proposed in this research, that their use in software quality evaluation shows the behavior of software evaluators and the relationship between the software characteristics, the desired consequences, and the values sought. Several attempts in consumer research have been made to provide a theoretical and conceptual structure connecting consumers' values to their behavior (Gutman, 1982; Howard, 1977; Vinson et al., 1977; Young & Feigin, 1975). The basis by which the study is performed is via adopting Gutman's means-end chain model (Bagozzi, 1997; Bagozzi & Dabholkar, 2000; Gutman, 1982, 1997; Valette-Florence 1998), which posits that linkages between product characteristics, consequences produced through usage, and personal values of users underlie the decision-making process or, in our case, the software quality evaluation process. The term *means* refers to the software product and services, and *ends* refers to the personal values important to the user.

Gutman's model is able to give a complete representation of the means-end chain, representing linkages from the characteristics to the values, along with the capability of explicating the chain. Earlier models (Howard, 1977; Vinson et al., 1977; Young & Feigin, 1975) failed to consider the means-end chain. They focused only on the consequences and values without considering how they relate to the characteristics or attributes. The means-end chain model seeks to explain how a person's choice of a product or service enables him or her to achieve his or her desired result. Such a framework consists of elements that represent the major usage processes that link personal values to behavior. Two assumptions underlie this model:

- all user actions have consequences; and
- all users learn to associate particular consequences from the usage of the product or service.

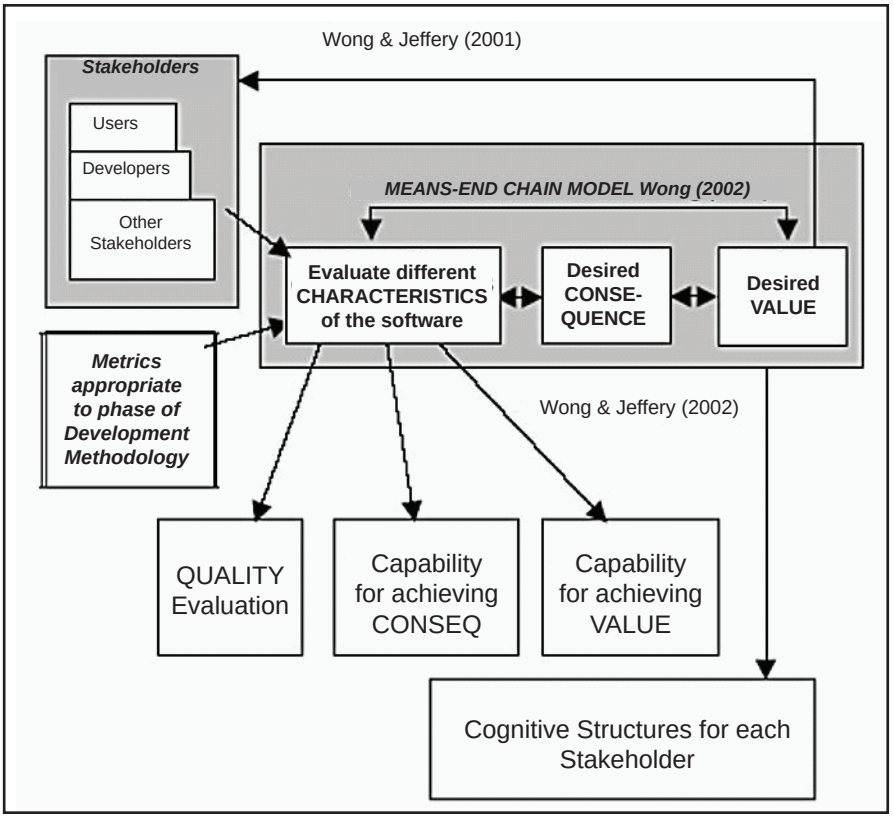
Consequences may be desirable or undesirable; they may stem directly from usage or the act of usage, or occur indirectly at a later point in time or from others' reaction to one's consumption behavior. The central aspect of the model is that users choose actions that produce desired consequences and minimize undesired consequences. Of course, because it is the characteristics that produce consequences, consideration for the characteristics that the software product possesses must be made. Therefore, it is important to make aware the characteristic-consequence relations. Overall, the characteristic-consequence-value interrelations are the focus of the model. Values provide the overall direction, consequences determine the selection of specific behavior in specific situations, and the characteristics are what is in

the actual software product that produces the consequences. It is knowledge of this structure that permits us to examine the underlying meaning of quality.

As highlighted in the literature, the benefit of utilizing Gutman's model in the framework is that it shows how the desired values influence the behaviors of individuals in all aspects of their lives (Gutman, 1982; Rokeach, 1968; Yankelovich, 1981). Gutman's model suggests that the desired consequences and the values sought are the motivators behind the choice of characteristic for software evaluation. In addition to this, the framework also highlights the significance of this relationship through the relationships between characteristics and consequences and also between the characteristics and value. It is through these relationships that the possibility of using the characteristics to evaluate each consequence and value becomes apparent.

The framework shown in Figure 2 is based on Gutman's Means-End Chain Model. As can be seen in this diagram, the framework consists of a number of boxes describing the three elements of Gutman's model, the stakeholders who evaluate the software quality, the outcome for the quality evaluation, and the arrows linking these elements, while also describing the direction of the influence. The Means-End Chain Model has been placed in the main box, as it is proposed, in this framework, to be the central influence for the choice of characteristics used in software evaluation, and the influence for the differences found between stakeholders. The framework not

Figure 2. SEF: Software evaluation framework



only shows the related elements, which influence software quality evaluation, but also introduces a way of describing the relationships between the characteristics, the consequences, and the values. These diagrams are called cognitive structures. Cognitive structures can be drawn for each stakeholder, allowing stakeholder differences to be described through a diagram (Wong, 2002, 2003a; Wong & Jeffery, 2001).

The recent studies of Wong and Jeffery (Wong, 2002, 2002a; Wong & Jeffery, 2001, 2002) provide the premise to this framework. An exploratory study by Wong and Jeffery (2001), utilized a qualitative approach to explore the influence of value on the choice of characteristics, and to determine whether cognitive structures could be used as a tool to represent the links between the characteristics, consequences, and values. The results of the study not only gave strong support for the influence of value on the choice of characteristic in software evaluation, but also supported earlier pilot studies on stakeholder differences (Wong, 1998; Wong & Jeffery, 1995, 1996), identifying different cognitive structures for users and developers.

A more recent paper by Wong (2002a) reported on a large quantitative study, which tested the appropriateness of utilizing Gutman's Means-End Chain Model (Bagozzi, 1997; Bagozzi & Dabholkar, 2000; Gutman, 1982, 1997; Valette-Florence, 1998) in software evaluation. Unlike previous studies of Gutman's Model, this study showed strong support for the Means-End Chain Model in a software evaluation setting. The results showed strong correlations between the characteristics, the consequences and the values, and supported the results of the qualitative study (Wong & Jeffery, 2001) described earlier.

The Software Evaluation Framework has also been applied to the requirements phase of the software development project (Wong, 2003a, 2003b, 2004). The results showed support for the use of the framework. The results showed that the measurements between the different phases are not the same, though the motivation behind the choice of these measurements is the same for a stakeholder group. The study also finds that the two groups of stakeholders are very similar in the measurements they choose for evaluating requirements documents; however, the motivation behind their choice of these measurements differs between the stakeholder groups. These results are a contrast to that of the implementation phase. More recently, studies have also been conducted on the metrics, which could be used to measure the characteristics of the software product (Wong, 2003a, 2003c, 2004, 2004a).

CONCLUSION

An exhaustive and critical review of the research concerned specifically with software product quality is reviewed in this chapter. It is clear that some models have provided more progress toward refining the definition of software quality while others have contributed to a better understanding of the process for measuring software quality. While the attempts of many of the models have been admirable, the development of the Software Evaluation Framework has succeeded in introducing a framework which fills the gaps found in many of the models.

Several broad conclusions can be drawn from the literature.

First, it is understood that there are different views of quality and that an understanding of these views is required; however, it is difficult to adopt all views at the same time when evaluating software. Though a number of assertions have been made in the literature regarding the multiple views of quality, there have not been many studies to support this, nor are there any frameworks or models which give the rationale for the different views.

Second, we know little about what rationale is used for selecting a characteristic for software evaluation. There is a need to consider what

motivates the characteristics used in software evaluation for EACH different view.

Third, the Software Evaluation Framework, introduced in this chapter, showed support for applying Gutman's Means-End Chain Model in software quality evaluation. The results endorse the links between characteristics, consequences and values, which have been regarded as rich constructs for understanding software evaluator behavior (Izard, 1972; Rokeach, 1973; Tolman, 1951). The results of the study provided further evidence of a connection between patterns of characteristics, consequences, and values and gives valuable support for the use of cognitive structures as a means of describing this relationship. Knowing this relationship is valuable for marketing and promoting the software, which is important for software acceptance. Software developers have often been challenged with conducting appropriate software demonstrations. For many developers, there is a lack of understanding of what is important to the user. The Software Evaluation Framework introduces the motivation for user acceptance of the software and allows the developer to have a better understanding of the desired consequences and how they are related to the software. The framework is valuable for understanding the differences in views of quality among the many stakeholders in a software development project. The cognitive structures introduced a way of describing these differences, showing clearly the characteristics of the software product, which are significant in their evaluation of software quality, along with the desired consequences and sought-after values associated with these characteristics.

Fourth, the use of the cognitive structures, introduced in the Software Evaluation Framework, can better the understanding of the characteristics selected for quality evaluation and how they are related to what the evaluators seek as desired consequences from the use of the software. The cognitive structures not only showed that users and developers differ in their choice of characteristics, but show differences in the consequences sought and the values desired. Software Quality has been described as a combination of characteristics, with ISO9126 being adopted as the current international standard. However, though support for ISO9126 appeared in some of the results, evidence suggests that non-ISO9126 characteristics are also important when evaluating software quality. This is supported by the cognitive structures and the results of earlier studies of Wong and Jeffery (Wong, 1998; Wong & Jeffery, 1995, 1996). The results found that ISO9126 characteristics, usability, and functionality strongly affect software quality for both developers and users, while technical characteristics, like portability and maintainability, were only significant for the developers. However, the results surprisingly found operational characteristics, like efficiency and reliability, to minimally affect software quality. This result was not expected since many of the measurements for software quality, like defects and failures, focus on reliability. As to the non-ISO9126 characteristics, support was found to be important for both users and developers, while the economic and institutional characteristics were only relevant for users. It is evident that further work is required to identify whether the non-ISO9126 characteristics should be part of the ISO9126 set of characteristics. The use of the cognitive structures can also help to identify the problems that may be occurring in current evaluation practices. Software Quality is seldom confined to just one characteristic, though very often, reliability seems to be the only focus. Defects and failures have often been the only metrics collected. While reliability, an operational characteristic, is important, the results of the research show that other metrics need to be collected if a more generally applicable set of measure for Software Quality is desired for different stakeholders.

REFERENCES

Bagozzi, R. (1997, September 1997). Goal-directed behaviors in marketing: Cognitive and emotional. *Psychology & Marketing, 14*, 539-543.

Bagozzi, R., & Dabholkar, P. (2000, July). Discursive psychology: An alternative conceptual foundation to means-end chain theory. *Psychology & Marketing, 17*, 535-586.

Bockenhoff, E., & Hamm, U. (1983). Perspektiven des Marktes für Alternativ erzeugte Nahrungsmittel. *Berichte über Landwirtschaft, 61*, 341-381. (Cited in Steenkamp, 1989).

Boehm, G. (1963 April). Reliability engineering. *Fortune*, pp. 124-127.

Boehm, B., Brown, J., & Lipow, M. (1976). Quantitative evaluation of chance of receiving desired consequences. *Proceedings of the Second International Conference on Software Engineering* (pp. 592-605).

Box, J. (1984). Product quality assessment by consumers – the role of product information. *Proceedings of the XIth International Research Seminar in Marketing*, Aix-en-Provence (pp. 176-197).

Braa, K., & Ogrim, L. (1994). Critical view of the application of the ISO standard for quality assurance. *Journal of Information Systems, 5*, 253-269.

Carpenter, S., & Murine, G. (1984, May). Measuring software product quality. *Quality Progress*, 16-20.

Cavano, J., & McCall, J. (1978, November). A framework for the measurement of chance of receiving desired consequences. *Proceedings of the ACM SQA Workshop* (pp. 133-139).

Checkland, P., & Scholes, J. (1990). *Soft systems methodology in action*. Chichester, UK: Wiley.

Coallier, F. (1994, January). *How ISO9001 fits into the software world*. IEEE Software.

Comerford, R. (1993, January). Software. *IEEE Spectrum*, pp. 30-33.

Crosby, P. (1978). *Quality is free*. Maidenhead: McGraw-Hill.

Dahlborn, B., & Mathiassen, L. (1993). *Computers in context. The philosophy and practice of systems design*. Cambridge, MA: NCC Blackwell.

Delen, G., & Rijsenbrij, D. (1992). The specification, engineering, and measurement of information systems quality. *Journal of Systems and Software, 17*(3), 205-217.

Dowson, M. (1993). Software process themes and issues. *Proceedings of the 2nd International Conference on the Software Process: Continuous Software Process Improvement* (pp. 54-60).

Dromey, R. (1995, February). A model for software product quality. *IEEE Transactions on Software Engineering, 21*(2), 146-162.

Erikkson, I., & McFadden, F. (1993). Quality function deployment: A tool to improve software quality. *Information and Software Technology, 35*(9), 491-498.

Feldman, M.S., & March, J.G. (1981). Information in organizations as signal and symbol. *Administrative Science Quarterly, 26*, 171-186.

Fenton, N. (1996, January). Do standards improve quality: A counterpoint. *IEEE Software, 13*(1), 23-24.

Garvin, D. (1984). What does "product quality" really mean? *Sloan Management Review, 24*.

Genth, M. (1981). Qualität und Automobile – Eine Untersuchung am Beispiel des deutschen Automobilmarktes 1974-1977. Frankfurt: Lang, in Steenkamp Product Quality. (Cited in Steenkamp, 1989).

Gilb, T. (1996, January). Level 6: Why we can't get there from here. *IEEE Software*, 97-103.

Gillies, A. (1992). *Software quality, theory and management* (1st ed.). London: Chapman & Hall.

Greenbaum, J., & Kyng, M. (1991). *Design at work: Cooperative design of computer systems.* NJ: Lawrence Erlbaum.

Gulezian, R. (1995). Software quality measurement and modeling, maturity, control and improvement. *IEEE COMPCON 1995 Proceedings* (pp. 52-60).

Gutman, J. (1982). A means-end chain model based on consumer categorization processes. *Journal of Marketing, 46*(Spring), 60-72.

Gutman, J. (1997). Means-end chains as goal hierarchies. *Psychology & Marketing, 14*(6), 545-560.

Hatton, L. (1993). *The automation of software process and product quality, chance of receiving desired consequences Management I.* Computational Mechanics Publications.

Holbrook, M.B., & Corfman, K.P. (1983). Quality and other types of value in the consumption experience: Phaedrus rides again. In Jacoby & Olson (Eds.), *Perceived quality* (pp. 31-57). Lexington.

Howard, J.A. (1977). *Consumer behaviour: Application of theory.* New York: McGraw-Hill.

Humphrey, W. (1988, March). Characterising the software process: A maturity framework. *IEEE Software, 5*(2), 73-79.

Hyatt, L., & Rosenberg, L. (1996, April). A software quality model and metrics for identifying project risks and assessing software quality. *Proceedings of the 8th Annual Software Technology Conference*, Utah.

ISO. (1986). ISO8402 Quality-Vocabulary, International Organization for Standardization, Geneva.

Izard, C. (1972). *Human emotions.* New York: Plenum Press.

Juliff, P. (1994). *Chance of receiving desired consequences Function Deployment, Chance of receiving desired consequences Management II Vol 1.* Computational Mechanics Publications.

Juran, J.M., & Gryna, F.M. (1980). *Quality planning and analysis.* New York: McGraw-Hill.

Kawlath, A. (1969). *Theoretische Grundlagen der Qualitätspolitik*. Wiesbaden, Germany: Gabler GmbH, in Steenkamp Product Quality. (Cited in Steenkamp, 1989).

Kitchenham, B. (1987, July). Towards a constructive quality model. Part 1: Chance of receiving desired consequences modeling, measurement and prediction. *Software Engineering Journal.*

Kitchenham, B., Linkman, S., Pasquini, A., & Nanni, V. (1997). The SQUID approach to defining a quality model. *Software Quality Journal, 6*, 211-233.

Kitchenham, B., & Pfleeger, S. (1996, January). *Software quality: The elusive target.* IEEE Software.

Kitchenham, B., & Pickard, L. (1987, July). Towards a constructive quality model. Part 2: Statistical techniques for modeling chance of receiving desired consequences in the ESPRIT REQUEST project. *Software Engineering Journal.*

Kitchenham, B., & Walker, J. (1986, September). *The meaning of quality.* Software Engineering 86: Proceedings of BCS-IEE Software Engineering 86 Conference, Southampton, England.

Kotler, P. (1984). *Marketing management: Analysis, planning and control* (5th ed.). Englewood Cliffs, NJ: Prentice Hall.

Kuehn, A., & Day, R. (1962). Strategy of product quality. *Harvard Business Review, 40,* 100-110.

Kupsch, P., & Hufschmied, P. (1978). Die Struktur von Qualitätsurteilen und das Informationverhalten von Konsumenten beim Kauf langlebiger Gebrauchsgüter. Oplagen: Westdeutscher Verlag. (Cited in Steenkamp, 1989).

Maynes, E.S. (1976b). *Decision-making for consumers.* New York: MacMillan Publishing.

McCall, J., Richards, P., & Walters, G. (1977, November). Factors in chance of receiving desired consequences, Vol 1, 2, & 3.

Monroe, K., & Krishnan, R. (1985). The effect of price and subjective product evaluations. In J. Jacoby & J.C. Olson (Eds.), *Perceived quality* (pp. 209-232). Lexington: Lexington Books.

Myers, W. (1993, March). Debating the many ways to achieve quality. *IEEE Software.*

OED. (1990). *Oxford English Dictionary.*

Olson, J.C., & Jacoby, J. (1972). Cue utilization in the quality perception process. *Proceedings of the Third Annual Conference of the Association for Consumer Research,* Iowa City, Iowa (pp. 167-179).

Ould, M. (1992). Chance of receiving desired consequences. Improvement through Process Assessment - A view from the UK. *Proceedings of the IEEE Colloquium on Chance of receiving desired consequences.*

Oxenfeldt, A.R. (1950). Consumer knowledge: Its measurement and extent. *Review of Economics and Statistics, 32,* 300-316.

Paulk, M.C. (1991). Capability maturity model for software (Report CMU/SEI-91-TR-24). SEI, Carnegie Mellon University.

Paulk, M.C., Curtis, B., Chrissis, M.B., & Weber, C.V. (1993, July). The capability maturity model, Version 1.1. *IEEE Software, 10*(4), 18-27.

Pfleeger, S.L. (2001). *Software engineering: Theory and practice* (2nd ed.). Prentice Hall.

Pfleeger, S.L., Jeffery, R., Curtis, B., & Kitchenham, B. (1997, March/April). Status report on software measurement. *IEEE Software,* 34-43.

Rae, A., Robert, P., & Hausen, H.L. (1995). *Software evaluation for certification. Principles, practice and legal liability.* UK: McGraw-Hill.

Rokeach, M. (1968). *Beliefs, attitudes and values.* San Francisco: Jossey-Bass.

Rokeach, M. (1973). *The nature of human values.* New York: Free Press.

Rombach, D., & Basili, V. (1990). Practical benefits of goal-oriented measurement. *Proceedings of the Annual Workshop of the Centre for Software Reliability: Reliability and Measurement,* Garmisch-Partenkirchen, Germany.

Rozman, I., Horvat, R., Gyorkos, J., & Hericko, M. (1977). PROCESSUS - Integration of SEI CMM and ISO quality models. *Software Quality Journal, 6,* 37-63.

Schneidewind, N.F. (1993, April). New software quality metrics methodology standards fill measurement needs. *IEEE Computer,* 105-196.

Schneidewind, N.F. (1996, January). Do standards improve quality: A point. *IEEE Software,* 13(1), 22-24.

Stolterman, E. (1991). Designarbetets dolda rationalitet. PhD thesis, Research Report No. 14:91. Information Processing and Computer Science, Institutionen för Informationsbehandling, Administrativ Databehandling University of Umeå. (Cited in Braa et al., 1994).

Sunazuka, T., Azuma, M., & Yamagishi, N. (1985). Chance of receiving desired consequences. Assessment technology. *Proceedings of the IEEE 8th International Conference on Sofware Engineering.*

Thompsett, R. (1993). Quality agreements for quality systems. *Proceedings of Australian Computer Society,* Victorian Branch, Annual Conference.

Thurstone, W.R. (1985). Quality is between the customer's ears. *Across the Board*, pp. 29-32.

Tolman, E. (1951). A psychological model. In T. Parsons & E. Shils (Eds.), *Towards a general theory of reasoned action* (pp. 148-163). Cambridge: Harvard University Press.

Trenkle, K. (1983). Lebensmittelqualität und Verbraucherschutz. *AID-verbrauchersdienst,* pp. 211-216. (Cited in Steenkamp, 1989).

Valette-Florence, P. (1998, June). A causal analysis of means-end hierarchies in a cross-cultural context: Metholodogical refinements. *Journal of Business Research, 42*(2), 161-166.

Vidgen, R., Wood, J., & Wood-Harper, A. (1994). *Customer satisfaction: The need for multiple perspectives of information system quality. Chance of receiving desired consequences Management II Vol 1.* Computaional Mechanics Publications.

Vinson, D.E., Scott, J.E., & Lamont, L.M. (1977, April). The role of personal values in marketing and consumer behaviour. *Journal of Marketing, 41,* 44-50.

Voas, J.M. (1997, July/August). Can clean pipes produce dirty water? *IEEE Software*, pp. 93-95.

Vollman, T.E. (1993, June). Software quality assessment and standards. *Computer, 6*(6), 118-120.

Wimmer, F. (1975). Das Qualitatsurteil des Konsumenten: Theoretische Grundlagen und Empirische Ergebnisse. Frankfurt: Lang. (Cited in Steenkamp, 1989).

Wittgenstein, L. (1953). *Philosophical investigations.* New York: MacMillan.

Wolff, M.F. (1986). Quality/process control: What R and D can do. *Research Management,* pp. 9-11.

Wong, B. (1998). Factors influencing software quality judgment (Tech. Rep.). CSIRO.

Wong, B. (2002, May 25). *Comprehending software quality: The role of cognitive structures.* Proceedings of the International Workshop on Software Quality (in conjunction with ICSE 2002).

Wong, B. (2002a). *The appropriateness of Gutman's means-end chain model in software evaluation.* Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE 2002).

Wong, B. (2003). *Measurements used in software quality evaluation.* Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP 2003).

Wong, B. (2003a). *Applying the software evaluation framework "SEF" to the software development life cycle*. Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE 2003).

Wong, B. (2003b). *Measuring the quality of the requirements specification document for an e-commerce project.* Proceedings of the 2003 International Business Information Management Conference (IBIM 2003).

Wong, B. (2003c). *A study of the metrics applied to the software evaluation framework "SEF".* Proceedings of the Third International Conference on Quality Software (QSIC 2003).

Wong, B. (2004). *A study of the metrics for measuring the quality of the requirements specification document.* Proceedings of the 2004 International Conference on Software Engineering Research and Practice (SERP 2004).

Wong, B. (2004a, December). The software evaluation framework "SEF" extended. *Information and Software Technology Journal, 46*(15), 1037-1047.

Wong, B., & Jeffery, R. (1995, November 22-24). *Quality metrics: ISO9126 and stakeholder perceptions.* Proceedings of the Second Australian Conference on Software Metrics (ACOSM'95), Sydney (pp. 54-65).

Wong, B., & Jeffery, R. (1996). A pilot study of stakeholder perceptions of quality (Tech. Rep.), CSIRO.

Wong, B., & Jeffery, R. (2001). *Cognitive structures of software evaluation: A means-end chain analysis of quality.* Proceedings of the Third International Conference on Product Focused Software Process Improvement (PROFES 2001).

Wong, B., & Jeffery, R. (2002). *A framework on software quality.* Proceedings of the Fourth International Conference on Product Focused Software Process Improvement (PROFES 2002).

Wood, J.R. (1992). Linking soft systems methodology (SSM) and information systems (IS). *Systemist - Information Systems Special Edition, 14*(3), 133-135.

Wood-Harper, A.T., & Avison, D. (1992). Reflections from the experience of using multiview: Through the lens of soft systems methodology. *Systemist, 14*(3), 136-145.

Yankelovich, D. (1981, April). New rules in American life: Search for self-fulfilment in a world turned upside down. *Psychology Today,* pp. 60.

Young, D.E., & Feigin, E. (1975, July). Using the benefit chain for improved strategy formulation. *Journal of Marketing,* 39, 72-74.

This work was previously published in Measuring Information Systems Delivery Quality, edited by E. Duggan and J. Reichgelt, pp. 55-89, copyright 2006 by Idea Group Publishing (an imprint of IGI Global).

Chapter 1.22
Software Configuration Management in Agile Development

Lars Bendix
Lund Institute of Technology, Sweden

Torbjörn Ekman
Lund Institute of Technology, Sweden

ABSTRACT

Software configuration management (SCM) is an essential part of any software project and its importance is even greater on agile projects because of the frequency of changes. In this chapter, we argue that SCM needs to be done differently and cover more aspects on agile projects. We also explain how SCM processes and tools contribute both directly and indirectly to quality assurance. We give a brief introduction to central SCM principles and define a number of typical agile activities related to SCM. Subsequently, we show that there are general SCM guidelines for how to support and strengthen these typical agile activities. Furthermore, we establish a set of requirements that an agile method must satisfy to benefit the most from SCM. Following our general guidelines, an agile project can adapt the SCM processes and tools to its specific agile method and its particular context.

INTRODUCTION

In traditional software development organisations, software configuration management (SCM) is often pushed onto the projects by the quality assurance (QA) organisation. This is done because SCM in part can implement some QA measures and in part can support the developers in their work and therefore helps them to produce better quality. The same holds true for agile methods—SCM can directly and in-directly contribute to better QA on agile projects.

Software configuration management (SCM) is a set of processes for managing changes and modifications to software systems during their

entire life cycle. Agile methods embrace change and focus on how to respond rapidly to changes in the requirements and the environment (Beck, 1999a). So it seems obvious that SCM should be an even more important part of agile methods than it is of traditional development methods. However, SCM is often associated with heavily process-oriented software development and the way it is commonly carried out might not transfer directly to an agile setting. We believe there is a need for SCM in agile development but that ist should be carried out in a different way. There is a need for the general values and principles of SCM, which we consider universal for all development methods, and there is a need for the general techniques and processes, which we are certain will be of even greater help to agile developers than they are to traditional developers.

There are some major differences in agile projects compared to traditional projects that heavily influence the way SCM can—and should—be carried out. Agile methods shift the focus from the relation between a project's management and the customer to the relation between developers and the customer. While traditional SCM focuses on the projects and company layers in Figure 1, there is a need to support developers as well when using SCM in agile projects. Shorter iterations, more frequent releases, and closer collaboration within a development team contribute to a much greater stress on SCM processes and tools.

Agile methods are people-oriented rather than process-oriented and put the developer and the customer in focus. As a consequence, SCM has to shift its primary focus from control activities to that of service and support activities. The main focus on audits and control needs to be replaced by a main focus on supporting the highly iterative way of working of both the team and the developers, as seen in Figure 2. From a QA point of view, the control measures are moved down to the developers themselves with the purpose of shortening the feedback loop in agile methods. So SCM does not directly contribute to the QA on an agile project, this is the task of the processes that the agile method in question prescribes. However, by supporting said processes and making them easier and safer to practice SCM indirectly is a big factor in QA on agile projects.

The traditional process-oriented view of SCM has also lead to several misconceptions of agile methods from an SCM point of view. The lack of explicit use of SCM and its terminology has lead quite a few people to conclude that agile methods are not safe due to an apparent lack of rigorous change management. However, a lot of

Figure 1. The different layers of SCM

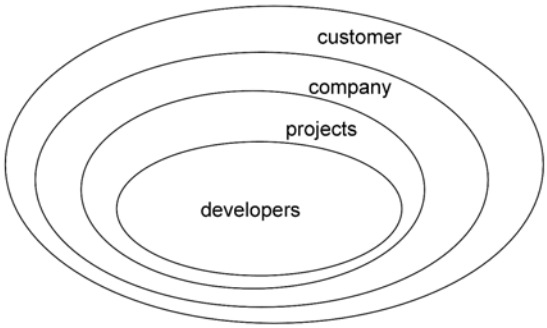
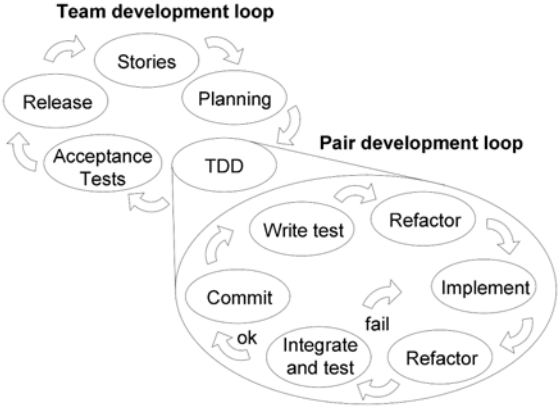


Figure 2. The main development loops in agile



SCM activities are actually carried out in agile methods although they are not mentioned explicitly. Bendix and Hedin (2002) identify a need for better support from SCM, in particular for refactoring in order for this practice to be viable. Koskela (2003) reviews agile methods in general from an SCM point of view and concludes that only a few of the existing agile methods take SCM explicitly into account. He also notices that most methods highly value SCM tool support but that SCM planning has been completely forgotten. There is thus a need to provide guidance for using SCM or for implementing SCM in agile. The SCM literature mostly takes the control-oriented view of SCM (Berlack, 1992; Buckley, 1993; Hass, 2003; Leon, 2005) and there is very little written about team- and developer-oriented support activities (Babich, 1986; Mikkelsen & Pherigo, 1997; Bendix & Vinter, 2001; Berczuk & Appleton, 2003). These activities are the ones that can benefit agile methods the most and should therefore be emphasized more when used in an agile setting. However, it is important to stress that agile methods need the whole range of SCM support from developer through to customer.

In the next section, we provide background information about SCM for those who are not so familiar with SCM, and describe and define a number of SCM-related agile activities to establish a terminology. In Section 3, we give general guidelines for how these agile activities can be supported by SCM and how agile methods could benefit from adopting more SCM principles. We also provide pointers to literature where more details can be found. Future trends for SCM in the agile context are described in Section 4, and in Section 5 we draw our conclusions.

BACKGROUND

This section gives an introduction to the concepts and terminology in SCM that serve as a background for the analysis in following sections. We also define and describe activities in agile methods that are related to SCM or affected by SCM in one way or the other.

SCM Activities

SCM is a method for controlling the development and modifications of software systems and products during their entire life cycle (Crnkovic, Asklund, & Persson Dahlqvist, 2003). From this viewpoint, SCM is traditionally divided into the following activities: configuration identification, configuration control, configuration status accounting, and configuration audit (Leon, 2005). These activities reflect mostly the part of a development project with relations to the customer. However, since agile methods are often more developer centric, there is also a need for a more developer-oriented view of SCM than the traditional control-oriented view above. Typical developer-oriented aspects of SCM include: version control, build management, workspace management, concurrency control, change management, and release management (Bendix & Vinter, 2001). We present each activity from a general perspective and explain both its purpose and what is included in the activity. After this introduction, the reader should be familiar with these basic SCM concepts and their purpose, so we can use them for our analysis in the next section.

Configuration Identification

Configuration identification is the activity where a system is divided into uniquely identifiable components, called configuration items, for the purpose of software configuration management. The physical and functional characteristics of each configuration item are documented including its interfaces and change history. Each configuration item is given a unique identifier and version to distinguish it from other items and other versions of the same item. This allows us to reason about

a system in a consistent way both regarding its structure and history. Each item can be either a single unit or a collection (configuration) of lower level items allowing hierarchical composition. During configuration identification a project baseline and its contents are also defined, which helps to control change as all changes apply to this uniquely defined baseline.

Configuration Control

Software is very different from hardware as it can be changed quickly and easily, but doing so in an uncontrolled manner often leads to chaos. Configuration control is about enabling this flexibility in a controlled way through formal change control procedures including the following steps: evaluation, coordination, approval or disapproval, and implementation of changes. A proposed change request typically originates from requests for new features, enhancement of existing features, bug reports, etc. A request is first evaluated by a Change Control Board (CCB) that approves or disapproves the request. An impact analysis is performed by the CCB to determine how the change would affect the system if implemented. If a request is approved, the proposed change is assigned to a developer for implementation. This implementation then needs to be verified through testing to ensure that the change has been implemented as agreed upon before the CCB can finally close the change request.

Configuration Status Accounting

Developers are able to track the current status of changes by formalizing the recording and reporting of established configuration items, status of proposed changes, and implementation of approved changes. Configuration status accounting is the task to provide all kinds of information related to configuration items and the activities that affect them. This also includes change logs, progress reports, and transaction logs. Configuration status accounting enables tracking of the complete history of a software product at any time during its life cycle and also allows changes to be tracked compared to a particular baseline.

Configuration Audits

The process of determining whether a configuration item, for instance a release, conforms to its configuration documents is called configuration audit. There are several kinds of audits each with its own purpose but with the common goal to ensure that development plans and processes are followed. A functional configuration audit is a formal evaluation that a configuration item has achieved the performance characteristics and functions defined in its configuration document. This process often involves testing of various kinds. A physical configuration audit determines the conformity between the actual produced configuration item and the configuration according to the configuration documents. A typical example is to ensure that all items identified during configuration identification are included in a product baseline prior to shipping. An in-process audit ensures that the defined SCM activities are being properly applied and controlled and is typically carried out by a QA team.

Version Control

A version control system is an invaluable tool in providing history tracking for configuration items. Items are stored, versions are created, and their historical development is registered and conveniently accessible. A fundamental invariant is that versions are immutable. This means that as soon as a configuration item is given a version number, we are assured that it is unique and its contents cannot be changed unless we create a new version. We can therefore recreate any version at any point in time. Version control systems typically support configuration status accounting by providing automatic support for history track-

ing of configuration items. Furthermore, changes between individual versions of a configuration item can be compared automatically and various logs are typically attached to versions of a configuration item.

Build Management

Build management handles the problem of putting together modules in order to build a running system. The description of dependencies and information about how to compile items are given in a system model, which is used to derive object code and to link it together. Multiple variants of the same system can be described in a single system model and the build management tool will derive different configurations, effectively building a tailored system for each platform or product variant. The build process is most often automated, ranging from simple build scripts to compilation in heterogeneous environments with support for parallel compilation. Incremental builds, that only compile and link what has changed, can be used during development for fast turn around times, while a full build, rebuilding the entire system from scratch, is normally used during system integration and release.

Workspace Management

The different versions of configuration items in a project are usually kept in a repository by the version control tool. Because these versions must be immutable, developers cannot be allowed to work directly within this repository. Instead, they have to take out a copy, modify it, and add the modified copy to the repository. This also allows developers to work in a controlled environment where they are protected from other people's changes and where they can test their own changes prior to releasing them to the repository. Workspace management must provide functionality to create a workspace from a selected set of files in the repository. At the termination of that workspace, all changes performed in the workspace need to be added to the repository. While working in the workspace, a developer needs to update his workspace, in a controlled fashion, with changes that other people may have added to the repository.

Concurrency Control

When multiple developers work on the same system at the same time, they need a way to synchronize their work; otherwise it may happen that more than one developer make changes to the same set of files or modules. If this situation is not detected or avoided, the last developer to add his or her changes to the repository will effectively erase the changes made by others. The standard way to avoid this situation is to provide a locking mechanism, such that only the developer who has the lock can change the file. A more flexible solution is to allow people to work in parallel and then to provide a merge facility that can combine changes made to the same file. Compatible changes can be merged automatically while incompatible changes will result in a merge conflict that has to be resolved manually. It is worth noticing that conflicts are resolved in the workspace of the developer that triggered the conflict, who is the proper person to resolve it.

Change Management

There are multiple and complex reasons for changes and change management needs to cover all types of changes to a system. Change management includes tools and processes that support the organization and tracking of changes from the origin of the change to the approval of the implemented source code. Various tools are used to collect data during the process of handling a change request. It is important to keep traceability between a change request and its actual implementation, but also to allow each piece of code to be associated to an explicit change request. Change management is also used to provide valuable metrics about the progress of project execution.

Release Management

Release management deals with both the formal aspects of the company releasing to the customer and the more informal aspects of the developers releasing to the project. For a customer release, we need to carry out both a physical and a functional configuration audit before the actual release. In order to be able to later re-create a release, we can use a bill-of-material that records what went into the release and how it was built. Releasing changes to the project is a matter of how to integrate changes from the developers. We need to decide on when and how that is done, and in particular on the "quality" of the changes before they may be released.

Agile Activities

This section identifies a set of agile activities that either implement SCM activities or are directly affected by SCM activities. The presentation builds on our view of agile methods as being incremental, cooperative, and adaptive. Incremental in that they stress continuous delivery with short release cycles. Cooperative in that they rely on teams of motivated individuals working towards a common goal. Adaptive in that they welcome changing requirements and reflect on how to become more effective. While all activities presented in this section may not be available in every agile method, we consider them representative for the agile way of developing software.

Parallel Work

Most projects contain some kind of parallel work, either by partitioning a project into sub-projects that are developed in parallel, or by implementing multiple features in parallel.

Traditional projects often try to split projects into disjoint sub-projects that are later combined into a whole. The incremental and adaptive nature of agile methods requires integration to be done continuously since new features are added as their need is discovered. Agile methods will therefore inevitably lead to cooperative work on the same, shared code base, which needs to be coordinated. To become effective, the developers need support to work on new features in isolation and then merge their features into the shared code base.

Continuous Integration

Continuous integration means that members of a team integrate their changes frequently. This allows all developers to benefit from a change as soon as possible, and enables early testing of changes in their real context. Continuous integration also implies that each member should integrate changes from the rest of the team for early detection of incompatible changes. The frequent integration decreases the overall integration cost since incompatible changes are detected and resolved early, in turn reducing the complex integration problems that are common in traditional projects that integrate less often.

Regular Builds

Agile projects value frequent releases of software to the customer and rapid feedback. This implies more frequent builds than in traditional projects. Releases, providing added value to the customer, need to be built regularly, perhaps on a weekly or monthly basis. Internal builds, used by the team only, have to be extremely quick to enable rapid feedback during continuous integration and test-driven development. This requires builds to be automated to a large extent to be feasible in practice.

Refactoring

The incremental nature of agile methods requires continuous Refactoring of code to maintain high quality. Refactorings need to be carried out as

a series of steps that are reversible, so one can always back out if a refactoring does not work. This practice relies heavily on automated testing to ensure that a change does not break the system. In practice, this also means that it requires quick builds when verifying behavioural preservation of each step.

Test-Driven Development

Test-driven development is the practice that test drives the design and implementation of new features. Implementation of tests and production code is interleaved to provide rapid feedback on implementation and design decisions. Automated testing builds a foundation for many of the presented practices and requires extremely quick builds to enable a short feedback loop.

Planning Game

The planning game handles scheduling of an XP project. While not all agile methods have an explicit planning game, they surely have some kind of lightweight iterative planning. We emphasize planning activities such as what features to implement, how to manage changes, and how to assign team resources. This kind of planning shares many characteristics with the handling of change requests in traditional projects.

SCM IN AN AGILE CONTEXT

In the previous section, we defined some agile activities that are related to SCM and we also outlined and described the activities that make up the field of SCM. In this section, we will show how SCM can provide support for such agile activities so they succeed and also how agile methods can gain even more value from SCM. It was demonstrated in Asklund, Bendix, and Ekman (2004) that agile methods, in this case exemplified by XP, do not go against the fundamental principles of SCM. However, it also showed that, in general, agile methods do not provide explicit nor complete guidance for using or implementing SCM. Furthermore, the focus of SCM also needs to shift from control to service and support (Angstadt, 2000) when used in agile. SCM does not require compliance from agile, but has a lot of good advice that you can adapt to your project if you feel the need for it—and thus value people and interactions before tools and processes (Agile Manifesto, 2001).

In this section, we first look at how SCM can support and service the agile activities we defined in the previous section. Next, we look at how agile methods could add new activities and processes from SCM and in this way obtain the full benefit of support from SCM.

How Can SCM Techniques Support Agile?

SCM is not just about control and stopping changes. It actually provides a whole range of techniques and processes that can service and support also agile development teams. Agile methods may tell you what you should do in order to be agile or lean, but in most cases, they are also very lean in actually giving advice on how to carry out these agile processes. In this sub-section, we show how SCM techniques can be used to support and strengthen the following SCM-related agile activities: parallel work, continuous integration, regular builds, refactoring, test-driven development, and planning game.

Parallel Work

Agile teams will be working in parallel on the same system. Not only on different parts of the system leading to shared data, but also on the same parts of the system, leading to simultaneous update and double maintenance. Babich (1986) explains all the possible problems there are when coordinating a team working in parallel—and

also the solutions.

The most common way of letting people work in parallel is not to have collective code ownership, but private code ownership and locking of files that need to be changed. This leads to a "split and combine" strategy where only one person owns some specific code and is allowed to change it. Industry believes that this solves the problem, but the "shared data" problem (Babich, 1986) shows that even this apparently safe practice has its problems (e.g., combining the splits). These problems are obviously present if you practise parallel work as well. In addition, we have to solve the "simultaneous update" problem and the "double maintenance" problem, when people actually work on the same file(s) in parallel.

The "shared data" problem is fairly simple to solve—if the problem is sharing, then isolate yourself. Create a physical or virtual workspace that contains all of the code and use that to work in splendid isolation from other people's changes. Obviously you cannot ignore that other people make changes, but having your own workspace, you are in command of when to "take in" those changes and will be perfectly aware of what is happening.

The "simultaneous update" problem only occurs for collective code ownership where more people make changes to the same code at the same time. Again, the solution is fairly simple, you must be able to detect that the latest version, commonly found in the central repository, is not the version that you used for making your changes. If that is not the case, it means that someone has worked in parallel and has put a new version into the repository. If you add your version to the repository, it will "shadow" the previous version and effectively undo the changes done in that version. If you do not have versioning, the new version will simply overwrite and permanently erase the other person's changes. Instead you must "integrate" the parallel changes and put the resulting combined change into the repository or file system. There are tools that can help you in performing this merge.

The "double maintenance" problem is a consequence of the "protection" from the "shared data" problem. In the multiple workspaces, we will have multiple copies of every file and according to Babich (1986) they will soon cease to be identical. When we make a change to a file in one workspace, we will have to make the same change to the same file in all the other workspaces to keep the file identical in all copies. It sounds complicated but is really simple, even though it requires some discipline. Once you have made a change, you put it in the repository and—sooner or later—the other people will take it in from the repository and integrate it if they have made changes in parallel (see the "simultaneous update" problem).

A special case of parallel work is distributed development where the developers are physically separated. This situation is well known in the SCM community and the described solutions (Bellagio & Milligan 2005) are equally applicable to distributed development as to parallel work. There are solutions that make tools scale to this setting as well. Distributed development is thus not different from parallel work from an SCM perspective, as long as the development process that SCM supports scales to distributed development.

In summary, we need a repository where we can store all the shared files and a workspace where we can change the files. The most important aspect of the repository is that it can detect parallel work and that it can help us in sorting out such parallel work. Also it should be easy and simple to create whole workspaces. Most version control tools are able to do that and there is no need to use locking, which prevents real parallel work, since optimistic sharing works well. We must thus choose a tool that can implement the copy-merge work model (Feiler, 1991).

Continuous Integration

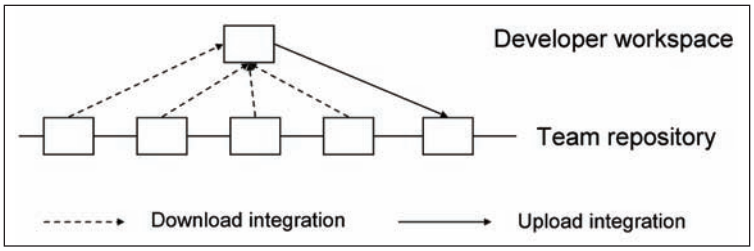
In traditional projects, the integration of the contributions of many people is always a painful process that can take days or even weeks. Therefore, continuous integration seems like a mission impossible, but this is actually not the case. The reason why integration is painful can be found in the "double maintenance" problem (Babich, 1986)—the longer we carry on the double maintenance without integrating changes, the greater the task of integration will be. So there are good reasons for integrating as often as possible, for instance after each added feature.

Integrating your change into the team's shared repository is often a two-step process. The reason is that tools usually cannot solve merge conflicts and re-run automated tests to check the quality in one step. First, you have to carry out a "download" (or subscription) integration where you take all the changes that have been added to the repository since you last integrated and integrate them into your workspace, as shown in Figure 3, where a box represents a new version of the configuration. If nothing new has happened, you are safe and can do the "upload" (or publication) integration, which simply adds your changes as the latest versions in the repository. If something has changed in the repository, it can be either new versions of files that you have not changed—these can simply be copied into your workspace—or files that you have changed where there may be conflicting changes. In the latter case you have to merge the repository changes into your own changes. At this point, all other people's changes have been integrated with your changes and your workspace is up-to-date, so you could just add the result to the repository. However, you should check that the integration actually produced a viable result and check the quality of it. This can be done by running a set of quality tests (e.g., unit tests, acceptance tests), and if everything works well, then you can add the result to the repository—if your workspace is still up-to-date. Otherwise, you have to continue to do "download" integrations and quality checks until you finally succeed and can do the "upload" integration, as shown in Figure 3.

This way of working (except for the upload quality control) is implemented in the strict long transactions work model (Feiler, 1991). You will notice that in this process, the upload integration is a simple copy of a consistent and quality assured workspace. All the work is performed in the download integration. Following the advice of Babich (1986), this burden can be lessened if it is carried out often as the changes you have to integrate are smaller. So for your own sake you should download integrate as often as possible. Moreover, for the sake of the team you should upload (publish) immediately when you have finished a task or story so other people get the possibility to synchronize their work with yours.

What we have described here is the commonality between the slightly different models and approaches presented in Aiello (2003), Appleton,

Figure 3. Download and upload integration



Berczuk, and Konieczka (2003a, 2003b, 2004a), Appleton, Berczuk, and Cowham (2005), Farah (2004), Fowler and Foemmel (2006), Moreira (2004) and Sayko (2004). If you are interested in the details about how you can vary your approach to continuous integration depending on your context, you can consult the references.

Continuous integration leads to an increased velocity of change compared to traditional development. This puts additional strains on the integration process but is not a performance problem on the actual integration per se. However, there may be performance issues when the integration is combined with a quality gate mechanism used to determine whether changes are of sufficient quality to be integrated in the common repository or not. Even if this quality gate process is fully automated, it will be much slower than the actual merge and upload operation and may become a bottleneck in the integration process. It may therefore not always be possible to be true to the ideal that developers should carefully test their code before uploading their changes in which case you could use a more complex model for continuous integration (Fowler & Foemmel, 2006) that we will describe next under regular builds.

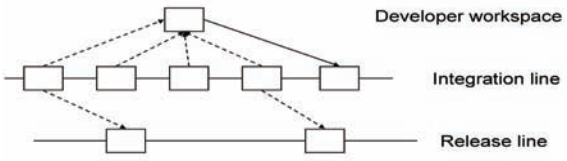
Regular Builds

When releases become frequent it also becomes important to be able to build and release in a lean way. If not, much time will be "wasted" in producing these releases that are needed to get customer feedback. Making it lean can be done in three ways: having always releasable code in the repository, performing a less formal release process, and automation of the build and release processes.

Before you can even think about releasing your code, you have to assure that the code you have is of good quality. In traditional development methods this is often done by a separate team that integrates the code and does QA. In agile, this is done by the developers as they go. The ideal situation is that the code in the repository is always of the highest quality and releasable at any time. This is not always possible and you can then use a mix between the traditional way and the agile ideal by having multiple development lines. The developers use an integration line to check in high quality code and to stay in sync with the rest of the developers. The QA-team uses a separate line to pull in changes from the integration line and does a proper and formal QA before they "promote" the approved change to the release line, as seen in Figure 4.

In agile methods, there is a general tendency to move the focus of QA from coming late in the development process, just before release, to being a centre of attention as early as possible in the development process. This means that agile can do with a less formal release process than traditional projects because much of the work has already been done. However, there is still a need to do physical and functional audits and to work with bill-of-materials such that earlier releases can be re-created again if needed. In agile methods, functional audits can be carried out by running the

Figure 4. Working with integration and release lines



acceptance tests. They are the specification of the requirements that should be implemented. To be really sure that we have implemented everything we claim, we should check the list of acceptance tests against the list of requirements we claim have been implemented in this release. We also need to check whether all the files that should be in the release (e.g., configuration files, manual, documentation, etc.) are actually there.

When releasing becomes a frequent action, there is a much greater need to automate it. The actual creation of the release can be automated by using build tools; acceptance tests and the verification that all files are there can be automated by writing simple scripts.

More information about regular builds can be found in Appleton and Cowham (2004b).

Refactoring

Refactoring is an important part of agile methods but also to some extent in traditional methods. The purpose of a Refactoring is not to implement new functionality, but rather to simplify the code and design.

In general, there are two different situations where you do refactorings: as part of a story to simplify the code before and/or after the implementation of the story's functionality; and architectural refactorings that are needed to implement a whole new set of features. In both cases, the two main problems are that a refactoring may touch large parts of the code and that the refactoring should be traceable and possible to undo. The latter means that there is the need for version control tool to keep track of the steps of each refactoring and make it possible to back out of a refactoring if it turns out that is does not work.

The fact that refactorings tend to be "global" possibly affecting large parts of the code, puts even greater strains on the continuous integration since there are more possibilities of merge conflicts. The recommendation for successful application of continuous integration is to integrate very often to reduce the risk of merge conflicts. The same goes for refactorings that should be split up into many small steps that are integrated immediately when they are done.

If you need to refactor code to facilitate the ease of implementing a story, then this refactoring should be seen as a separate step and integrated separately—the same goes if you need to refactor after the implementation of the story. For the architectural refactorings, we need to split the refactoring up into smaller tasks such that there will be as little time as possible between integrations to lower the risk of merge conflicts. Larger refactorings should also be planned and analysed for impact such that it is possible to coordinate the work to keep down the parallel work, or at least to make people aware of the fact that it is going on.

For a more specific treatment of the problems architectural refactorings can cause to SCM tools and the continuous integration process and how these problems can be dealt with, we refer the reader to Ekman and Asklund (2004) and Dig, Nguyen, and Johnson (2006).

Test-Driven Development

The short version of test-driven development is design a little—where you design and write tests, code a little, and finally run the tests to get feedback. Here the crucial part is to get feedback on what you have just changed or added. If that cannot happen very quickly, test-driven development breaks down with respect to doing it in small increments. If you want to run your tests after writing a little code, you must be able to re-build the application you want to test very quickly—if you have to wait too long you are tempted to not follow the process as it is intended.

So what is needed is extremely quick re-builds, a matter of a few minutes or less, and the good news is that SCM can provide that. There are techniques for doing minimal, incremental builds that will give you a fast turn-around time, so you

can run your tests often without having to wait too long. Make (Feldman, 1979) is the ancestor of all minimal, incremental build tools, but there exists a lot of research on how to trade "consistency" of a build for time (Adams, Weinert, & Tichy, 1989; Schwanke & Kaiser, 1988). For the small pair development loop in Figure 2, we might be satisfied with less than 100% consistency of the build as long as it is blisteringly fast. For the big team development loop in Figure 2 (i.e., integrating with others), speed might not be that important while consistency of the build is crucial. A properly set up SCM system will allow developers to have flexible build strategies that are tailored to specific parts of their development cycle.

Another aspect of test-driven development is that if we get an unexpected result of a test-run, then we have to go bug hunting. What is it that has caused the malfunction? If you run tests often, it means that you introduced the bug in the code that you wrote most recently—or as Babich puts it "an ounce of derivation is worth a pound of analysis" (Babich, 1986)—meaning that if we can tell the difference in the code between now and before, we are well under way with finding the bug. Version control tools provide functionality for showing the difference between two versions of the same file and some tools can even show the structural differences between two versions of a configuration.

Planning Game

Agile methods use stories, or similar lightweight specification techniques, as the way that customers specify the requirements of the system, and acceptance tests to specify the detailed functionality. These stories specify changes to the system and correspond to change requests when analyzed from an SCM perspective. The stories, or change requests, have to be estimated for implementation cost by the developers and then prioritised and scheduled by the customer during the planning game. For someone coming from SCM this sounds very much like the traditional way of handling change requests: an impact analysis has to be carried out to provide sufficient information for the Change Control Board to be able to make its decision whether to implement the change request, defer it, or reject it. So we can see that the parallel to estimation in agile is impact analysis (Bohner & Arnold, 1996) in traditional SCM. Likewise, the parallel to the customer prioritising is the chair of the Change Control Board taking decisions (Daniels, 1985). For the planning game to work properly, it is important that everyone is aware of what his or her role is—and that they seek information that will allow them to fill that role well. It is also important to be aware of the fact that the traditional formal change request handling process can indeed—and should—be scaled to fit the agility and informality that is needed in an agile method.

How Can SCM Add More Value to Agile?

In agile methods, there is very much focus on the developers and the production process. In the previous sub-section, we have seen how many of these processes can be well supported by techniques and principles from SCM. However, agile methods often overlook the aspects of SCM that deal with the relation to the customer and where traditional SCM has special emphasis. In the following, we look at the traditional SCM as represented by the four activities of configuration identification, configuration control, configuration status accounting, and configuration audit (Leon, 2005). For each activity, we describe what new activities and processes could be added to agile methods to help provide a more covering support for the development team.

Configuration Identification

The most relevant part of configuration identification for agile methods is the identification

and organisation of configuration items. Some artefacts are so important for a project that they become configuration items and go into the shared repository. Other artefacts (e.g., sketches, experiments, notes, etc.) have a more private nature and they should not be shared in order not to confuse other people. However, it may still be convenient to save and version some of the private artefacts to benefit from versioning even though they are not configuration items. They can be put into the repository but it is very important that the artefacts, configuration items and not, are structured in such a way that it is absolutely clear what a configuration item is and what a private artefact is. Structuring of the repository is an activity that is also important when it contains only configuration items.

Configuration identification is an SCM activity that traditionally is done up-front, which goes against the agile philosophy. However, there can be some reason in actually trying to follow the experience that SCM provides. Rules for identifying configuration items should be agreed upon, such that they can be put into the repository and information about them shared as early as possible. More importantly, though, is that the structuring of configuration items should not be allowed to just grow as the project proceeds, because most SCM tools do not support name space versioning (Milligan, 2003) (i.e., handling structural changes to the repository while retaining the change history).

Configuration Control

The part of configuration control that deals with the handling of change requests is taken care of by a planning game or similar activity. However, two important aspects of configuration control are neglected by most agile methods: tracking and traceability.

In traditional SCM, change requests are tracked through their entire lifetime from conception to completion. At any given point in time, it is important to know the current state of the change request and who has been assigned responsibility for it. This can benefit agile methods too as they also need to manage changes and coordinate the work of different people. In some agile methods there is an explicit tracker role (chromatic, 2003) that is responsible for this activity.

Traceability is an important property of traditional SCM and is sometimes claimed to be the main reason for having SCM. It should be possible to trace changes made to a file back to the specific change request they implement. Likewise, it should be possible to trace the files that were changed when implementing a certain change request. The files that are related to a change request are not just source code files, but all files that are affected by that change (e.g., test cases, documentation, etc). Another aspect of traceability is to be able to know exactly what went into a specific build or release—and what configurations contain a specific version of a specific file. The main advantage of having good traceability is to allow for a better impact analysis so we can be informed of the consequences of changes and improve the coordination between people on the team.

Configuration Status Accounting

This activity should be seen as a service to everyone involved in a project including developers and the customer, even though it traditionally has been used primarily by management and in particular project managers. Configuration status accounting can be looked at as simple data mining where you collect and present data of interest. Many agile methods are very code centred and the repository is the place where we keep the configuration items that are important for the project, so it is natural to place the meta-data to mine in the same repository. Configuration status accounting does not need to be an upfront activity like configuration identification, but can be added as you discover the need. However, you should be aware that the

later you start collecting data to mine, the less data and history you get to mine. Usually this is seen as an activity that benefits only managers, but there can be much support for the developers too—all you have to do it to say what kind of meta-data you want collected and how you want it to be presented. If you do not document changes in writing, then it is important that you can get hold of the person that did a change; when you have shared code, then it is important to see who is currently working on what.

Configuration Audit

Configuration audit can be looked at as a verification activity. The actual work, considered as a QA activity, has been done elsewhere as part of other processes, but during the configuration audits, it gets verified that it has actually been carried out. The functional configuration audits verify that we have taken care of and properly closed all change requests scheduled for a specific build or release. The physical configuration audit is a "sanity check" that covers the physical aspects (e.g., that all components/files are there—CD, box, manual) and that it can actually be installed. Even though configuration audit is not directly a QA activity, it contributes to the quality of the product by verifying that certain SCM and QA activities have actually been carried out as agreed upon. Configuration audits are needed not because we mistrust people, but because from time to time people can be careless and forget something. The basis for automating the functional configuration audit in agile is there through the use of unit and acceptance tests.

SCM Plans and Roles

You definitely need to plan and design your SCM activities and processes very carefully on an agile project. Moreover, they have to be carried out differently from how they are done on traditional projects and the developers will need to know more about SCM because they are doing more of it on an agile project.

This does not imply that you should write big detailed SCM plans the same way as it is being done for traditional projects. The agile manifesto (Agile Manifesto, 2001) values working software over comprehensive documentation. The same goes for SCM where you should value working SCM processes over comprehensive SCM plans. In general, what needs to be documented are processes and activities that are either complex or carried out rarely. The documentation needs to be kept alive and used—otherwise it will not be up-to-date and should be discarded. We can rely on face-to-face conversation to convey information within a team when working in small groups and maybe even in pairs. However, if the team grows or we have a high turnover of personnel, that might call for more documentation. If possible, processes should be automated, in which case they are also documented.

In general, agile projects do not have the same specialization in roles as on traditional projects. Everyone participates in all aspects of the project and should be able to carry out everything—at least in theory. This means that all developers should have sufficient knowledge about SCM to be able to carry out SCM-related activities by themselves. There will, for instance, not be a dedicated SCM-person to do daily or weekly builds or releases on an agile project. However, to do the design of the SCM-related work processes, even an agile team will need the help of an SCM expert who should work in close collaboration with the team such that individuals and interaction are valued over processes and tools (Agile Manifesto, 2001).

SCM Tools

In general, SCM is very process centric and could, in theory, be carried out by following these

processes manually. However, agile methods try to automate the most frequently used processes and have tools take care of them (e.g., repository tools, build tools, automated merge tools, etc). Fortunately, the requirements that agile methods have to SCM tooling are not very demanding and can, more or less, easily be satisfied by most tools. For this reason, we do not want to give any tool recommendations or discuss specific tools, but rather focus on the general requirements and a couple of things to look out for. Furthermore, most often, you just use the tool that is given or the selection is based on political issues.

Using parallel work, we would need a tool that works without locking and thus has powerful merge capabilities to get as painless an integration as possible. Using test-driven development, we need to build very often so a fast build tool is very helpful—and preferably it will be flexible such that we can sometimes choose to trade speed for consistency. Working always against baselines, it would be nice if the repository tool would automatically handle bound configurations (Asklund, Bendix, Christensen, & Magnusson, 1999) so we should not do that manually.

However, a couple of things should be taken into account about SCM tooling. Because of refactoring and the fact that the architecture is grown organically, there will be changes to the structure of the files in the repository. This means that if the tool does not support name space versioning (Milligan, 2003), we will have a harder time because we lose history information and have no support for merging differing structures. However, this can be handled manually and by not carrying out structural changes in parallel with other work. It is much more problematic to actually change your repository tool in the middle of a project. Often you can migrate the code and the versions but you lose the meta-data that is equally as valuable for your work as the actual code. Therefore, if possible, you should try to anticipate the possible success and growth of the project and make sure that the tool will scale to match future requirements.

FUTURE TRENDS

While resisting the temptation to predict the future, we can safely assume that the increased use and awareness of SCM in agile development will result in a body of best practices and increased interaction between agile and SCM activities. Furthermore, we expect to see progress in tool support, including better merge support and increased traceability to name a few.

Continuous integration is an activity that has already received much attention and is quite mature and well understood. Many other SCM-related activities require continuous integration and we expect to see them mature accordingly when that foundation is now set. This will result in new best practices and perhaps specific SCM-related sub-practices to make these best practices explicit. A first attempt to specify SCM sub-practices for an agile setting is presented in Asklund, Bendix, and Ekman. (2004) and we expect them to mature and more sub-practices to follow.

SCM tools provide invaluable support and we envision two future trends. There is a trend to integrate various SCM-related tools into suites that support the entire line of SCM activities. These tools can be configured to adhere to pretty much any desired development process. They may, however, be somewhat heavyweight for an agile setting and as a contrast, we see the use of more lightweight tools. Most SCM activities described in this chapter can be supported by simple merge tools with concurrency detection.

Parallel work with collective code ownership can benefit from improved merge support. Current merge tools often operate on plain text at the file level and could be improved by using more fine-grained merge control, perhaps even with syntactic and partially semantics aware merge. An alternative approach is to use very fine-grained merge combined with support for increased awareness to lower the risk of merge conflicts. The increased use of SCM will also require merge support for

other artefacts than source files.

The use of SCM in agile development will enable better support for traceability and tracking of changes. A little extra effort can provide bi-directional traceability between requirements, defects, and implementation. However, more experience is needed to determine actual benefits in an agile context before one can motivate and justify this extra "overhead."

SCM is being used more and more in agile methods, despite not being mentioned explicitly. However, it is often carried out in the same way as in traditional projects, but can benefit from being adapted to this new setting. The practices presented in this chapter adapt SCM for agile methods but more widespread use will lead to even more tailored SCM. In particular, SCM practices will be further refined to fit an agile environment and probably lead to more agile SCM. Some of these ideas may indeed transfer to traditional projects, providing more lightweight SCM in that setting as well.

CONCLUSION

SCM provides valuable activities that enhance the QA for agile development. The main quality enhancement does not stem directly from SCM but indirectly by supporting other quality enhancing activities. Traceability is, for instance, crucial to evaluate any kind of quality work, and configuration audits verify that SCM and QA activities have been carried out.

We have shown how typical agile activities can be supported directly by SCM techniques while retaining their agile properties. For instance, continuous integration demands support from SCM tools and processes to succeed while build and release management can help to streamline the release process to enable frequent releases. SCM can thus be used to support and strengthen such developer-oriented activities.

SCM is traditionally very strong in aspects that deal with the relation to the customer. Agile methods can benefit from these activities as well. Configuration control allows precise tracking of progress and traceability for each change request. Lightweight SCM plans simplify coordination within a team and help in effective use of other SCM-related activities. These are areas that are often not mentioned explicitly in agile literature.

There is, in general, no conflict between agile methods and SCM—quite the contrary. Agile methods and SCM blend well together and enhance each other's strengths. Safe SCM with rigorous change management can indeed be carried out in an agile project and be tailored to agile requirements.

SCM tools provide help in automating many agile activities, but we must stress that what is important are the SCM processes and not so much a particular set of tools. There are also many agile activities that could be supported even better by enhanced tool support. For instance, current merge tools are often fairly poor at handling structural merges such as refactorings; often this results in loss of version history and traceability, and incomprehensible merge conflicts.

Many agile teams already benefit from SCM, but we believe that a more complete set of SCM activities can be offered to the agile community. Tailored processes and tools will add even more value and may indeed result in SCM activities that are themselves agile, which may even have an impact on more traditional software development methods.

REFERENCES

Adams, R., Weinert, A., & Tichy, W. (1989). Software change dynamics or half of all ADA compilations are redundant. *Proceedings of the 2nd European Software Engineering Conference*,Coventry, UK.

Agile Manifesto (2001). *Manifesto for agile software development.* Retrieved June 1, 2006, from

http://agilemanifesto.org/

Aiello, B. (2003). Behaviorally speaking: Continuous integration: Managing chaos for quality! *CM Journal,* September.

Angstadt, B. (2000). SCM: More than support and control. *Crosstalk: The Journal of Defence Software Engineering,* March.

Appleton, B., & Cowham, R. (2004b). Release management: Making it lean and agile. *CM Journal,* August.

Appleton, B., Berczuk, S., & Konieczka, S. (2003a). Continuous integration: Just another buzz word? *CM Journal,* September.

Appleton, B., Berczuk, S., & Konieczka, S. (2003b). Codeline merging and locking: Continuous updates and two-phased commits. *CM Journal,* November.

Appleton, B., Berczuk, S., & Konieczka, S. (2004a). Continuous staging: Scaling continuous integration to multiple component teams. *CM Journal,* March.

Appleton, B., Berczuk, S., & Cowham, R. (2005). Branching and merging: An agile perspective. *CM Journal,* July.

Asklund, U., Bendix L., Christensen H. B., & Magnusson, B. (1999, September 5-7). The unified extensional versioning model. *Proceedings of the 9th International Symposium on System Configuration Management,* Toulouse, France.

Asklund, U., Bendix, L., & Ekman, T. (2004, August 17-19). Software configuration management practices for extreme programming teams. *Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques,* Turku, Finland.

Babich, W. A. (1986). *Software configuration management: Coordination for team productivity.* Addison-Wesley.

Beck, K. (1999a). Embracing change with extreme programming. *IEEE Computer, 32*(10), 70-77.

Beck, K. (1999b). *Extreme programming explained: Embrace change.* Addison-Wesley.

Bellagio, D. E., & Milligan, T. J. (2005). *Software configuration management strategies and IBM Rational ClearCase.* IBM Press.

Bendix, L., & Vinter, O. (2001, November 19-23). Configuration management from a developer's perspective. *Proceedings of the EuroSTAR 2001 Conference,* Stockholm, Sweden.

Bendix, L., & Hedin, G. (2002). Summary of the subworkshop on extreme programming. *Nordic Journal of Computing, 9*(3), 261-266.

Berczuk, S., & Appleton, S. (2003). *Software configuration management patterns: Effective teamwork, Practical Integration.* Addison-Wesley.

Berlack, H. R. (1992). *Software configuration management.* John Wiley & Sons.

Bohner, S. A., & Arnold, R. S. (1996). *Software change impact analysis.* IEEE Computer Society Press.

Buckley, F. J. (1993). *Implementing configuration management: Hardware, software, and firmware.* IEEE Computer Society Press.

Chromatic. (2003). *Chromatic: Extreme programming pocket guide.* O'Reilly & Associates.

Crnkovic, I., Asklund, U., & Persson Dahlqvist, A. (2003). *Implementing and integrating product data management and software configuration management.* Artech House.

Daniels, M. A. (1985). *Principles of configuration management.* Advanced Applications Consultants, Inc.

Dig, D., Nguyen, T. N., & Johnson, R. (2006). *Refactoring-aware software configuration management* (Tech. Rep. UIUCDCS-R-2006-2710).

Department of Computer Science, University of Illinois at Urbana-Champaign.

Ekman, T., & Asklund, U. (2004). *Refactoring-aware versioning in eclipse*. Electronic Notes in Theoretical Computer Science, 107, 57-69.

Farah, J. (2004). Making incremental integration work for you. *CM Journal*, November.

Feiler, P. H. (1991). *Configuration management models in commercial environments* (Tech. Rep. CMU/SEI-91-TR-7). Carnegie-Mellon University/ Software Engineering Institute.

Feldman, S. I. (1979). Make—A program for maintaining computer programs. *Software—Practice and Experience*, *9*(3), 255-265.

Fowler, M., & Foemmel, M. (2006). *Continuous integration*. Retrieved June 1, 2006, from http://www.martinfowler.com/articles/continuousIntegration.html

Hass, A. M. (2003). *Configuration management principles and practice*. Addison-Wesley.

Koskela, J. (2003). *Software configuration management in agile methods*. VTT publications: 514, VTT Tietopalvelu.

Leon, A. (2005). *Software configuration management handbook*. Artech House.

Mikkelsen, T., & Pherigo, S. (1997). *Practical software configuration management: The Late-night developer's handbook*. Prentice Hall.

Milligan, T. (2003). *Better software configuration management means better business: the seven keys to improving business value*. IBM Rational white paper.

Moreira, M. (2004). Approaching continuous integration. *CM Journal*, November.

Sayko, M. (2004). The role of incremental integration in a parallel development environment. *CM Journal*, November.

Schwanke, R. W., & Kaiser, G. E. (1988, January 27-29). Living with inconsistency in large systems. *Proceedings of the International Workshop on Software Version and Configuration Control*, Grassau, Germany.

This work was previously published in Agile Software Development Quality Assurance, edited by I. Stamelos and P. Sfetsos, pp. 136-153, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 1.23
Governance of Software Development:
The Transition to Agile Scenario

Yael Dubinsky
IBM Haifa Research Lab, Israel
Technion–Israel Institute of Technology, Israel

Avi Yaeli
IBM Haifa Research Lab, Israel

Yishai Feldman
IBM Haifa Research Lab, Israel

Emmanuel Zarpas
IBM Haifa Research Lab, Israel

Gil Nechushtai
IBM Haifa Research Lab, Israel

ABSTRACT

Governance is the exercise of control and direction over a subject such as a society, an organization, processes, or artifacts, by using laws and policies that are defined, deployed, and executed. In this chapter we develop this definition into a formal conceptual model that can be applied to a variety of governance domains. At the heart of this model lies the concept of the governance solution and its lifecycle. The governance solution embodies the set of mechanisms—decision rights, policies, controls, and measurements—applied to a governance scope in order to achieve some governance goals. As part of the lifecycle, the effectiveness of the governance solution is measured, and corrections and alignments are made as necessary. We demonstrate how this model can be applied to multiple governance domains by providing examples from IT governance as well as software-development governance. We conclude by providing a detailed scenario in the

software-development governance space, which looks at large software organizations undergoing transition to agile development methodology. We further demonstrate how the governance model is instantiated and evolved in the context of this scenario.

INTRODUCTION

The field of information technology (IT) governance has garnered an increased amount of attention in recent years. However, it is still struggling to provide a universally agreed-upon definition and a complete model for IT governance, along with the required tools and techniques.

The definitions of IT governance that can be found in the literature from Broadbent (1998), Chulani, Clay, Yaeli, Wegman, and Cantor (2006), Van Grembergen and De Haes (2004), Weill and Ross (2004), and Williams (2005) and they all share common ideas, such as the need to increase the value of IT to the organization while reducing risk. For example, *Weill and Ross (2004) focus on decision rights and define IT governance as* "specifying the decision rights and accountability framework to encourage desirable behavior in the use of IT" (p. 8). Van Grembergen and De Haes (2004) address the alignment of the IT organization with the business needs, and define IT governance as "the leadership and organizational structures, processes, and relational mechanisms that ensure that the organization's IT sustains and extends the organization's strategy and objectives" (p. 1).

Chulani et al. (2006) include both decision rights and the alignment with business needs: "Within IBM, a widely accepted definition for IT governance is:

- Governance that pertains to an organization's information technology activities and the way those activities support the goals of the business
- Decision making rights associated with IT as well as the mechanisms and policies used to measure and control the way IT decisions are made and carried out within the organization" (p. 10).

In recent years, several IT governance and control frameworks, such as CobiT[1], ITIL[2], ISO-17799[3] have been developed. These frameworks help business management, IT management, quality practitioners, and auditors understand what needs to be done; yet they are far from being complete. Dahlberg and Kivijärvi (2006) outline the limitations of CobiT as a process-centric framework and suggest a new framework that takes an integrated process and structural approach, and links into corporate governance.

Another limitation stems from the fact that CobiT is a high-level framework targeted at IT organizations that support a business unit or a business organization. CobiT considers software development activities only within the context of providing a supporting service in a value chain for another business unit, rather than as a central business activity in itself. Software development activities are briefly described in CobiT as part of the high-level control objective AI2, "Acquire and Maintain Application Software." CobiT thus lacks a description of governance mechanisms that are appropriate for organizations with a large focus on software development. To that end, organizations need to refer to other standards and frameworks that focus more on software development and control of software development activities.

This chapter is aimed at bridging the gap between high-level IT governance and software development governance. We first present a model for governance in general, and then use the model to describe IT and software development domain-specific governance. The model is built based on a review of the literature and a set of scenarios, as explained in the next section. We use the process of transition to agile software development (Beck & Andres, 2004; Dubinsky, Hazzan, Talby, &

Keren, 2006; Highsmith, 2002) to demonstrate the domain-specific governance schemes.

The agile approach to software development has emerged over the last decade, becoming mainstream as more and more organizations adopt agile practices (Barnett, 2006). The approach is based on a manifesto[4] that emphasizes the individuals involved in the software development, collaboration with the customer, and the need to provide testable working software. Several principles and methods (Highsmith, 2002) are used by software teams in different capacities (Ambler, 2007). As agile software development becomes more common, software organizations are becoming more interested in governing the transition to an agile approach.

We present data from the first author's involvement in two software-development projects that were carried out at two different organizations that underwent the transition to agile. The first project was developed by the IT department of a financial organization. The second project was developed by the software group of an international product provider. The action research method (Lewin, 1948) is used in this field of transition to agile processes, where the researchers plan an action, execute it while collecting data, analyze the data and reflect on it, and then define the next action by refining their role. The data emerged from their participation in planning sessions, guiding retrospective processes, involvement in refining the process and product measures, and consulting to higher management on the adoption of agile methodologies into the work procedures.

GOVERNANCE MODEL

This section presents a model for governance. The purpose of the model is to uniformly represent the main concepts involved in a governing process and their interrelationships. The development of this model started with a literature review, and built upon existing work using a set of scenarios that relate to project management, software engineering, and development processes. The literature review included CobiT; Val IT[5]; ISO 17799; OCEG Foundation[6]; CMMI[7]; SWEBOK[8]; Weill and Ross (2004); and Abrams, von Känel, Mueller, Pfitzmann, and Ruschka-Taylor (2006). The set of scenarios is based on our field experience as well as experiences of other practitioners and researchers with whom we collaborate. The model attempts to abstract the elements of governance found in the various references and domains. We therefore start with the dictionary definitions of the word governance rather than with one of the many domain governance definitions that exist in previous works.

We begin by examining the general meaning of governance and incrementally introduce elements of the model that stem from that basic definition. The model reflects our view of how governance and governance processes should be organized and may require modifications to describe some existing situations.

The word govern is defined[9] as "to exercise continuous sovereign authority over; especially to control and direct the making and administration of policy in" and also "to control, direct, or strongly influence the actions and conduct of." The first part of the definition implies that

- Governance is an ongoing process;
- There must exist an entity with legitimate rights to exercise authority over the things that are subject to governance;
- Governance is concerned with controlling the way policies and laws are established.

The second part of the definition suggests that the purpose of governance is to influence or affect the activities, state, or behavior of the subjects being governed. Hence, governance affects and regulates its subjects through the administration of policies.

Scoping Authority over Subjects

The definition of governance specifies that governance exercises legitimate authority over the subjects being administered. We need a way to describe the boundaries of the subjects and activities being governed, as well as the boundaries of the area of jurisdiction over which the governing entity will have legitimate authority. Furthermore, we know that people are subject to multiple governing bodies, such as national and local governments, as well as the organizations where they work. It is therefore necessary to describe multiple authority hierarchies and the relationships across these levels.

A governance scope represents a set of entities and relationships that are subject to acts of governance. Governance scope is hierarchically decomposable so that it can capture the hierarchical nature of society and business organizations. However, in order to represent multiple overlapping hierarchies, a governance scope can belong to more than a single hierarchy. In principle, a scope can identify organizations, suborganizations, processes, activities, roles, and artifacts; it can then establish the boundaries over the entities that are governed. In the context of corporate governance, the scope would be the entire organization and its activities. In the case of IT governance, the scope would be the IT organization, processes, activities, roles, and resources. According to Cantor and Sanders (2007), it is often useful to express the scope of governance in terms of processes within organizations, since there are many existing standards that consistently decompose the entire activities of organizations into processes and activities. Examples of such standards include CobiT and ITIL.

A governing body, sometime referred to as the government, represents the set of roles that have the right to exercise authority over the governance scope. Within social and business organizations, it is common to find multiple governing bodies, each of which focuses on different kinds of scopes and is concerned with different governance needs. It is therefore useful to think about the arrangement of governing bodies in hierarchies and to align them with the hierarchies of governance scopes. By doing so, we can express the delegation of legitimate authority between governing bodies across the organization hierarchy, and the fact that legislation enacted by one governing body needs to conform, or at least not conflict with, legislation done by another governing body higher in the governing hierarchy chain. For example, a local government cannot create laws that violate national and federal laws. Within business organizations, it is common to find a hierarchy of governing bodies based on an organizational structure. We also consider process owners as governing bodies who are given authority to exercise control and legislation within the scope of their processes. Figure 1 illustrates the hierarchical nature of governance scopes and the relationships between these and the governing bodies.

There is a large body of political science literature that talks about types of governments (e.g., democracy, monarchy), how they are established,

Figure 1. Modeling governance scopes and governing bodies

and the accountability of the governors to their constituents. The current version of our model does not address those elements of political models of governments.

Goals of Governance

According to the second part of the definition, the purpose of governance is to influence the activities, state, or behavior of the subjects being governed. The need to influence the subjects in the first place often stems from external forces that place constraints or requirements on the activities within the governance scope. For example, state government regulations place constraints on organizations that do business within the jurisdiction of the state. Another example is the need to establish or update service delivery policies based on new security policies established by the larger organization. A final example is an IT organization that needs to control costs or improve performance based on business needs.

Hence, the context represents the overall situation and set of internal and external relationships in which a governance scope exists and in which its activities take place. The context sometimes acts as the driver or source of requirements for the act of governance.

A governance goal represents the desired state that the initiative or act of governance is trying to achieve within the governance scope. A goal needs to be measurable and provide a clear indication of how success and failure will be assessed. Governance goals are hierarchically decomposable, allowing the nesting of sub-goals. In this case, the success criteria of a high-level goal can be expressed as functions of the success criteria of the sub-goals. An example of a governance goal in business organizations is "ensuring that the organization performs effectively and efficiently against the requirements and imperatives coming from its context, and to ensure the delivery of the expected outcome." It is useful to express the governance goals in the terminology of the context; this enhances the communication between different stakeholders by providing a common vocabulary.

Governance Mechanisms

Based on our governance definition, governance requires the means to "control, direct, or strongly influence the actions and conduct" of the governed subjects. A governance mechanism represents an abstraction of the possible mechanisms that can be used to regulate, influence, or control the actions and conduct of elements described within the governance scope in order to achieve some governance goal. There are many kinds of governance mechanisms that have been suggested and categorized by academia, industry standards, and vendors. Weill and Ross (2004) highlight three categories of mechanisms: decision-making structures, process alignment, and communication mechanisms. CobiT focuses on mechanisms to control processes, and identifies policies, procedures, practices, and organizational structures as means of control. IBM identifies two major groups of mechanisms that are established in the governance process (Chulani et al., 2006; Ericsson, 2007):

- **Static mechanisms:** Chains of responsibility, authority, and communication (decision rights);
- **Dynamic mechanisms:** Measurement, policy, standards, and control mechanisms.

All these definitions are compatible and cover more or less the same types of mechanisms, although the organization and focus are sometimes different. The following are several examples of these mechanisms and how they influence the governance scope:

- Decision rights mechanisms are the means through which an organization can establish, charter, and communicate the roles and

responsibilities for particular management and decision-making processes. Typically, the decision rights are documented and communicated in a policy, such as a spending policy that allows a first-line manager to approve spending up to $3000 without a senior manager's signature. A RACI matrix (Hallows, 2001) is an example of a structured way to describe decision rights.

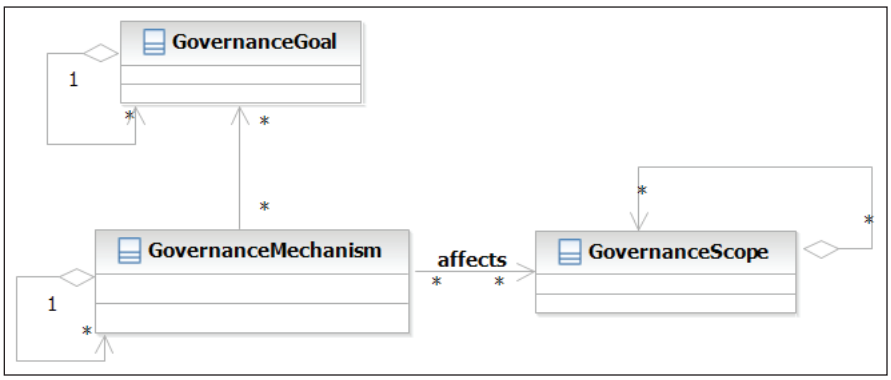
- Policies, procedures, guidelines, practices, and standards mechanisms all instruct the subjects under governance at varying level of formalism and strictness of the desired behavior or how to conduct their activities. Controls, measurements, and decision authority are often documented and communicated in policies and procedures.
- Control and measurement mechanisms provide the means for people with decision-making rights to control and monitor the activities for which they are responsible. Decision checkpoints, incentives, and policy assertions are examples of controls. For example, a project funding approval checkpoint is a control in the project funding process. An ROI measurement is a mechanism used to measure the return of investment in an asset. Another example is the measurement of estimated versus actual development time for software development tasks. Note that measurement has a dual role: it enables monitoring but also acts as an influencing mechanism that drives the behavior of the subjects.

The governance mechanism should provide a clear statement of its desired effect on the governance scope via one or more governance goals. Furthermore, governance mechanisms can be hierarchical, allowing governance goals to be met by a hierarchy of governance mechanisms. Figure 2 shows the part of the model that describes governance goals, scopes, and mechanisms. A governance mechanism affects a governance scope to realize a governance goal. In addition, a hierarchy of governance mechanisms can realize a hierarchy of governance goals.

Governance Points and Observables

In order for a governance mechanism to control and monitor an activity within the governance scope, it is necessary to identify the exact situation in the governance scope and the exact condition under which the governance mechanism should operate. This identification also serves as the specification for how to implement and deploy the governance mechanism. A governance point represents a specified location and situation within

Figure 2. Modeling governance scopes, mechanisms, and goals



the governance scope to which a governance mechanism should be applied. For example, a policy that enables a first-line manager to approve vacations that do not exceed two consecutive weeks creates a governance point. This point is the set of situations in which first-line managers in the governance scope should decide upon vacation approvals.

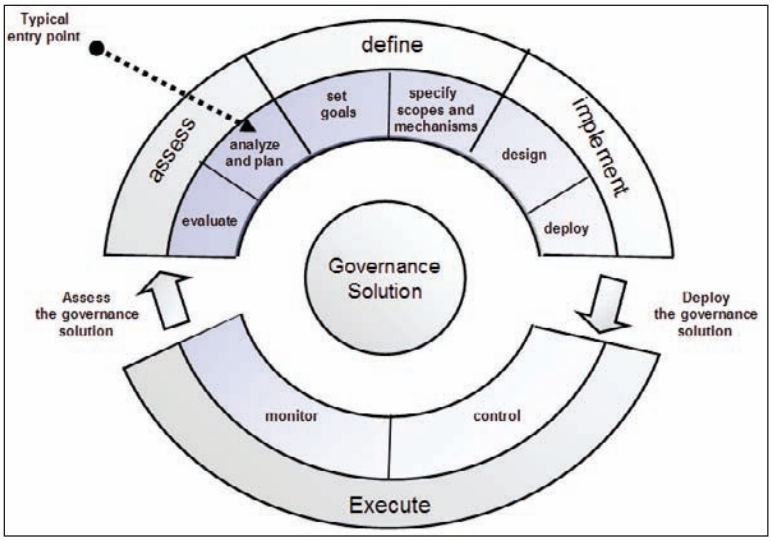
From an operational perspective, it is useful to express governance points in the context of artifact lifecycles, where events, activities, and state transitions of the artifacts act as potential points to which governance mechanisms can be applied. This creates a common structure for the definition of governance points. It also supports the implementation and integration of the governance mechanism into the processes and software automation of the activities described in the governance scope.

A governance observable represents a metric, event, piece of information, or artifact metadata that can be observed by a governance mechanism at a governance point. This provides the means to characterize the behavior of the governance scope by identifying observable information that could be used to help achieve the governance goals. It also allows the identification of specific properties within the governance scope that are relevant to achieving a governance goal. For example, such properties may include an event or attribute that are used for calculating a metric.

Governance Solutions and the Governance Process

So far we have shown how governance mechanisms can be associated with governance scopes to achieve governance goals. Often, sets of mechanisms, scopes, and goals collectively have some significance from the perspective of an organization, process, or initiative. In such cases, it is useful to refer to them as a group. A governance solution represents the collection of governance mechanisms applied to a set of governance scopes to achieve a set of related governance goals. For example, an IT governance solution is the set of governance mechanisms that are applied in the scope of an IT organization, its processes, and activities, to achieve the IT governance goals. Note that the term governance solution is commonly

Figure 3. The governance lifecycle (adapted from Cantor & Sanders, 2007; used with permission)



used (e.g., Cantor & Sanders, 2007) to denote the specification of the mechanisms, scopes, and goals. However, as we discuss next, a governance solution has its own lifecycle and it is necessary to discuss the state of the solution at specification time as well as at other phases of its lifecycle.

As presented in the definition, governance is an ongoing process. In fact, it is an iterative process through which the governance solution is established and evolved. In Figure 3, we present the lifecycle of that process and typical activities that are likely to take place in each phase of the lifecycle. Among different lifecycle views, such as the plan–do–study–act (PDSA) cycle (Tague, 2004), we have chosen to adapt the approach of Cantor and Sanders (2007) to governance lifecycle. This approach emphasizes the separation between activities to establish and evolve a governance solution, and those relating to the execution of the governance solution.

The governance process has four major phases:

1. **Assess:** The current governance solution is evaluated and new requirements for the governance solution are analyzed and planned. This includes measuring governance effectiveness metrics, assessing key performance indicators against previously defined governance goals, and planning how to address new governance needs arising from the context, such as new regulations.
2. **Define:** The governance solution is defined. The governance goals are captured and governance effectiveness measurements are defined. The scopes to bring under governance are determined and the governance mechanisms are specified.
3. **Implement:** This phase includes all the steps necessary to realize the defined governance specification and prepare it for execution by the organization. This phase could include process reengineering, automation and tool support, education, infrastructure deployment, policy announcement, and so forth. The governance model presented here does not focus on the artifacts generated in this phase.
4. **Execute:** The solution has already been deployed in the organization and management is expected to execute the governance solution. Managers and other specified roles are exercising their decision rights and playing a role in controlling and monitoring the scopes under their responsibility.

The governance lifecycle shows a clear separation between activities done to establish and evolve a governance solution and those that are done while executing a governance solution. This observation can be useful for understanding the relationships between the roles of governors and managers. It can be said that governors are responsible for establishing a governance solution while managers are responsible for executing the governance solution. Moreover, a governing body will sometimes assign decision rights to itself; in those cases, the governing body is also an actor in the execution of the governance solution. Similarly, some managers may sit in governing bodies; in those cases, they assume multiple roles of both governor and manager.

The governance solution can be viewed as having states that correspond to the governance process phases. It is interesting to note that there is always some version of a governance solution that is in the state of "executing" for any given scope. In each iteration of the lifecycle, the governance process can modify an executing governance solution by defining, implementing, and deploying a new version of that solution. Furthermore, some steps of the Assess phase of the governance lifecycle may also be running continuously by monitoring the executing governance solution.

A governance execution result represents the result of applying a governance mechanism at a particular time. It is a measurement that relates to the governance scope, but is used in the context

of the governance assessment phase. Examples of such measurements might be compliance records/status or governance performance indicators.

Figure 4 illustrates the model of the governance solution. A governance mechanism can be used to affect some behavior within a governance scope to realize some governance goal. The governance mechanism can be applied at specific governance points within the governance scope to affect or observe some behavior within the scope. The result of applying the mechanisms is stored in a governance execution result that is produced by the mechanism.

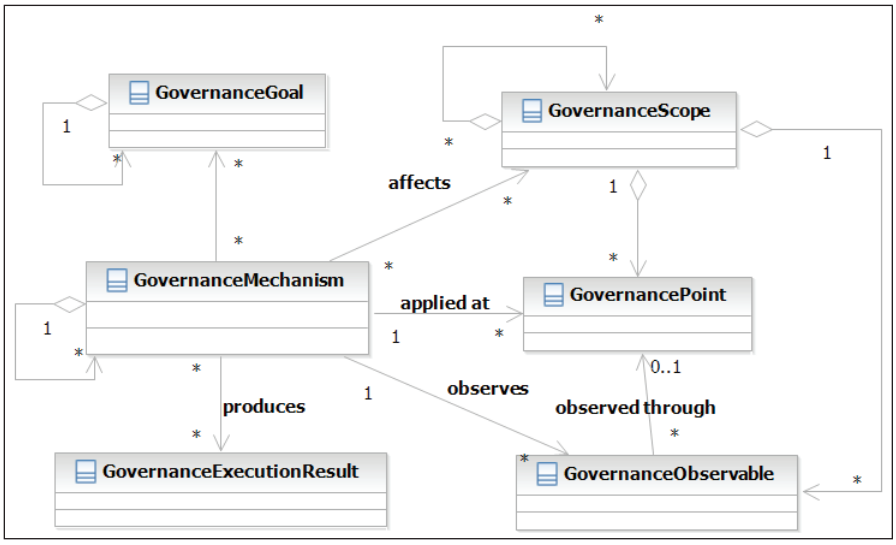
Systems of Governance Solutions

We are seeing a proliferation of governance solutions established by multiple governing bodies to cover a wide range of governance scopes. How are governance solutions related and how can the solutions be orchestrated to scale up when governing a large organization?

To answer this question, we make the following observations:

- There are multiple governance processes that are executed asynchronously by different governing bodies. Each has its own lifecycle and the governance solution of different governing bodies can be in different states. Furthermore, the cycle time may not be the same in all governance processes. Some processes may have a one-year cycle, while others have a quarterly cycle, depending on how adaptive and responsive the governance should be to the changing scope and context.
- Governance solutions have relationships. For example, governance decisions made by the large organization will have an effect on the governance solutions established for smaller organization scopes. In fact, the former can be viewed as part of the context of the latter. For example, a larger organization can define a policy stating that all sub-organizations should be ISO-certified within two years. This imposes a requirement for each organization to initiate a governance solution focusing on ISO certification.

Figure 4. Modeling the governance solution



- Governance solutions can be defined for varying granularities of scope. For example, governance solutions that are established by the board of directors and apply to an entire organization may coexist with a governance solution that focuses on development policies for a 30-person agile project.

To summarize, while the governance solution can autonomously execute for any given scope and goal, it can also link to other governance solutions either through the context or by establishing governance mechanisms that affect other governance solutions. These two characteristics ensure the scalability of the governance model.

TRANSITION TO AGILE SCENARIO: A GOVERNANCE PERSPECTIVE

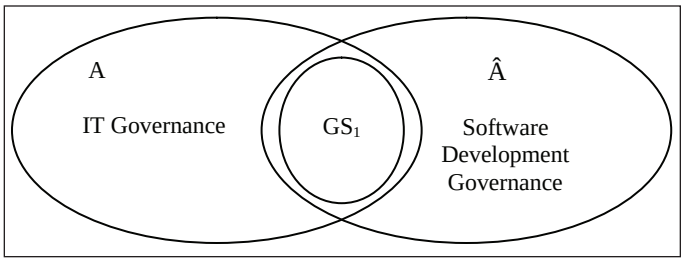
In this section, we present some of the data that was gathered in two different organizations that underwent a transition to agile development. The first (denoted by A) is a financial organization using software developed by an agile team, which is part of the organization's IT department. The team actually works for another organization (denoted by Â) and is outsourced to organization A to develop and maintain its software products. The second organization (denoted by B) is a global company that manufactures devices with embedded software. Several distributed teams, one of which has started to use agile methods, develop the software. The data was gathered by observations in planning sessions and development activities, consultation to higher management, and by guiding the retrospective process of the agile team.

We use the governance concepts described in the previous section to present the case of software development governance and analyze the transition-to-agile processes in the two organizations.

Governance scope. In both organizations, the transition to agile development began with one project. The management of organization Â decided to improve the delivery time of software projects and selected the project in organization A as an experiment for the agile approach. The intention of the management of organization Â was to extend use of the agile approach to more of its software projects. Examining the governance scope in this case, we denote by GS_1 the scope that contains the project team and its activities. Figure 5 schematically shows GS_1 as an inner scope in the two different hierarchies of organizations A and Â. In organization A, GS_1 is part of the IT governance; in organization Â, GS_1 is part of the software development governance. The teammates are the employees of organization Â outsourced to A.

In organization B, the project team and its activities are defined as the governance scope, which is within the hierarchy of the software development governance in this organization. In this case, the project manager together with his

Figure 5. Hierarchy of governance scopes in organizations A and Â



team leaders learned about the agile approach and decided to start the transition in their jurisdiction. In parallel, they started the process of convincing their upper management that it was worthwhile. This was different from organization Â, where the direction of governance is top-down.

Governing body. The governing body in the case of GS_1 included governors from both organizations A and Â. The governors from organization A were the CIO and the domain expert that served as the onsite customer. This expert was responsible for directing the team according to the release vision, giving them the requirements for each iteration, and prioritizing the requirements. The CIO and the domain expert could delegate authority to others in organization A, such as the people responsible for the infrastructure and those who tested the system as end users. The governors from organization Â continuously monitored this process in order to learn about its benefits and drawbacks. They worked together with the governors of organization A, who agreed to cooperate in implementing a new approach to software development.

In organization B, the project manager and team leaders served as the governing body. As part of the transition process, the role of methodologist was suggested. This person was in charge of maintaining and enhancing the methodology (Dubinsky & Hazzan, 2006). Although other roles that are concerned with the transition (e.g., tracker) were rotated among teammates, the person who performed this role did it consistently and continued to learn about agile methods. She also took upon herself to educate other teams in the organization about agile concepts. As a result, she naturally joined the governing body and is involved in every decision this body makes.

Governance goals. The governance goals of organization A are different from the governance goals of organization Â. Though they share the same scope, the context is different. The governing body of organization A is concerned with the needs of the IT users inside the organization and who use the software to support the business activities of this organization. In contrast, the governing body of organization Â is concerned with the kind of contracts it signs with its customers (in this case, organization A) and how these can be improved to increase business benefits as well as provide high quality products. This difference in governance goals on the same scope requires special attention when dealing with the mechanisms that should be used in this compound solution.

In organization B, the governance goals were derived from previous experience of the governing body in running software projects in this and other organizations. The goals refer on the one hand to the professional aspects of software production and on the other hand to the way the project is viewed in the context of the organization. Hence, one goal is to improve the way software requirements are dealt with and to shorten the time-to-market. Another goal is to follow the software development process used in this organization and published in an internal handbook. The decision to implement the agile approach to cope with the first goal implied that the governing body should communicate to higher management the changes caused by the agile implementation, especially these that do not fit existing procedures. They would also need to reconcile the governance solutions used inside the team and in its relations with the rest of the organization.

Governance mechanisms. The adoption of an agile approach implied several governance mechanisms. The most conspicuous was the work procedure that required short development iterations. In both organizations A and B, a release contains about eight iterations of two weeks each. When uncertainty increased, usually towards the end of the release, the iterations were shortened to one week. An iteration of two weeks was composed of one day of presentations and planning, and nine days of development. During the day of presentations and planning, the team presents the artifacts of the previous iteration to the customer and, together with the customer, plans the

functionality to be developed in the next iteration. During the nine days of development, the team develops the required functionality and no new requirements are accommodated.

Another example of a mechanism that was derived from the agile approach is the policy that states who the customer is and the definition of this role. In both organizations A and B, before the transition-to-agile process, the project manager played the role of the customer in setting the functionality developed by the team, along with the priorities and schedule. This behavior caused an on-going negotiation between the project manager and the real customer. The project manager generally refused to change requests that the real customer prioritized as important and continued to distribute work to the team as seemed necessary. The agile approach set the rules clearly: the customer is the one who provides the software requirements, decides what the team will develop in every iteration, and the priorities of these requirements. Of course, the customer can and should receive all the professional advice the team and the project manager can give. The customers should commit themselves to the process of development, be available during the iteration, particularly in the presentations where they gave feedback and in planning sessions where they reviewed the progress. The project manager should manage the development process and people so that a high quality software product will be developed.

The agile metrics are used in both organizations as the measurements of the process and product. Each team contains a tracker who is responsible for collecting the data required by the metrics and for communicating them at the end of the presentation of the iteration artifacts. The agile metrics can be used as governance observables. One measurement that was used by both teams is the calculation of estimated time of completed tasks versus the actual time that was invested to develop them. Next, we present governance observables that illustrate this measurement.

Governance points and observables as part of the governance lifecycle. The notions of governance scope, governing body, governance goals, and mechanisms are part of the Define and Implement phases of the governance lifecycle (see Figure 3). The governance points and observables exist in the Execute phase. The following are two kinds of governance mechanisms that are found in both organizations and that provide observables data:

- The measurement of estimated time of daily-completed tasks versus the actual time that was invested to develop them.
- The policy of conducting one-hour retrospectives at every iteration (Talby, Hazzan, Dubinsky & Keren, 2006).

These mechanisms belong to specific goals that are common in processes of transition to agile in general and were specifically set as goals also in our two cases.

- The first goal was to shorten delivery time. One of the mechanisms that serve this goal is measuring team velocity, which can be perceived as the amount of productive work units per iteration (Beck & Fowler, 2000). The agile approach recommends small releases of two to three months, each of which consists of short iterations of two to four weeks. Measuring team velocity per iteration enables ongoing visibility of the progress information as well as the ability to make decisions on how to continue.
- The second goal was to continuously improve the process by gradually adopting agile practices. One of the mechanisms that serve this goal is when teammates reflect on their activities (Hazzan, 2002; Kerth, 2001; Schön, 1983) after every iteration, before they start planning the next iteration. This is done as part of a retrospective process that enables individuals and teams to share ideas

about the major issues that emerge in their software development environments, think about ways to improve, and make decisions to support these improvements.

The data on team velocity and retrospective processes can be presented as governance observables that emerged in the Execute phase and are used as part of the Assess phase of the governance lifecycle. We present these observables in the following sections. We note that there are further issues that are relevant to the transition to agile scenario, such as testing and quality assurance, simplicity as a concept, and the role scheme, which are not addressed here.

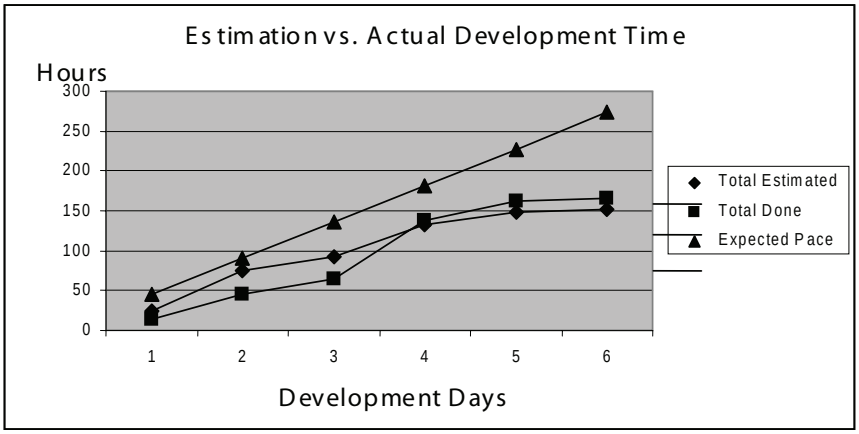
Team Velocity

The first governance observable is the graph of estimated time of completed tasks versus the actual time that was invested to develop them. One associated governance point occurs every development day when the tracker displays the graph during the stand-up-meeting that starts the team's workday. This is done in order to trace the team's daily progress. Another governance point that uses the same observable occurs every two weeks in the presentation of the iteration artifacts. This is done in order to trace the team's iteration progress. The daily progress is measured against the iteration commitment. The iteration progress is measured against the release commitment. We used this data to better understand the agile process. From the governance perspective, we can decide if this mechanism can assist us in the Assess phase to follow up on the performance of the governance solution.

Each day, in each of the teams, the tracker added two new points to the graph. The "total estimated" point represented the cumulative estimations of all tasks that were completed by the previous day and the "total done" point represented the cumulative actual time devoted to those tasks. A completed task was counted when the developer in charge completed the coding, unit testing, and integration with the entire developed system.

Figure 6 shows the graphs of estimations versus actual time in the third iteration for the team in organization A. The third (linear) line provides the expected pace according to available time. As can be observed, there is a significant difference between the time that is available for development and the time that is actually used for development. This phenomenon also happened in the case of the team in organization B. There are several reasons that may cause this behavior. Firstly, only

Figure 6. Estimation vs. actual development time in iteration 3 (team A)

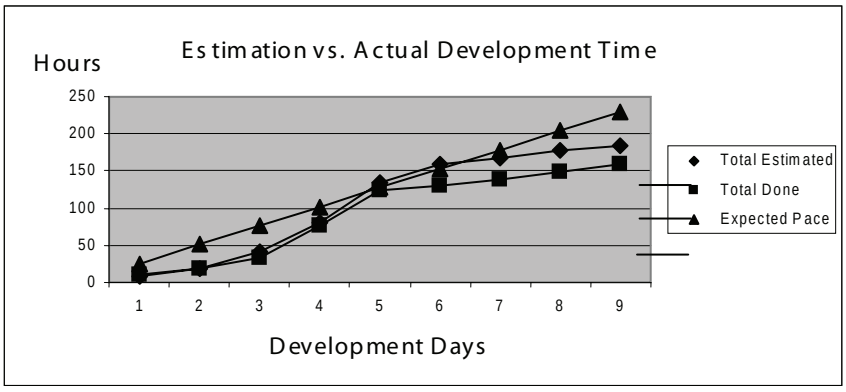


completed tasks are calculated. Although some teammates had invested time in other tasks, this time was not taken into consideration if the task was not complete before the end of the iteration. Secondly, there is often time invested in urgent tasks that come up, such as support service for end users who work with modules that are in production. Thirdly, some developers, whose time was taken into account, may have been absent.

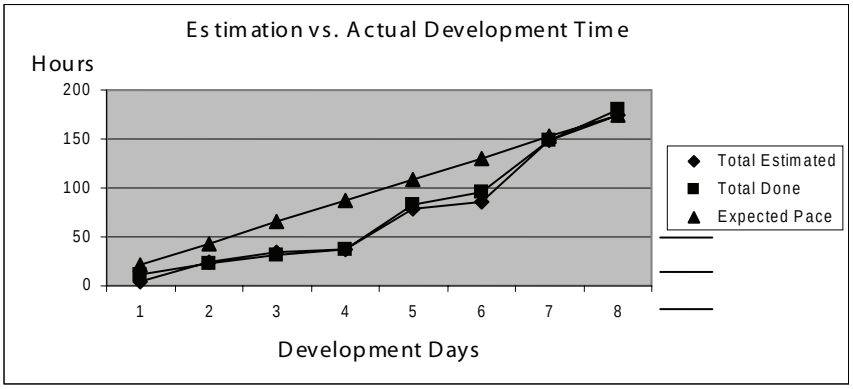
Figure 7 shows the graphs of estimations versus actual time in the second (Figure 7a) and forth (Figure 7b) iterations of the team in organization B. The Expected Pace line is the ratio between the total working hours of all teammates in the iteration and the number of days in the iteration.

We observe that at the end of the second iteration (Figure 7a), the amount of time actually spent on completed tasks (159 hours) was 30% less than the available time according to the expected pace (229 hours). This can be explained by the fact that tasks in progress that were not completed in this iteration were not counted. In the fourth iteration (Figure 7b), however, all tasks were completed. In such cases the total-expected point unites with the expected pace point since this was the amount of hours that was considered in the work planning.

Figure 7. Estimation vs. actual development time in iterations 2 and 4 (team B)

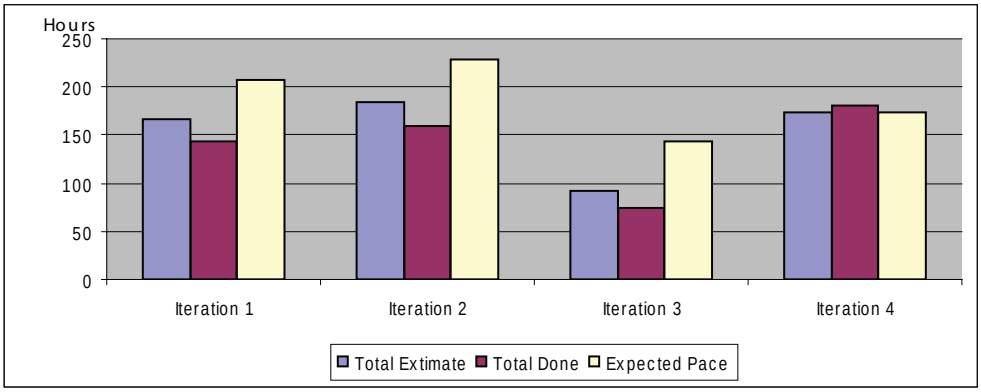


a) Iteration 2



(b) Iteration 4

Figure 8. Estimation vs. actual development time in the first 4 iterations (team B)



We further observe that in both iterations, the distribution into tasks was reasonably good. Too many dependencies between tasks can delay their completion, causing a nearly flat total-done line until close to the end of the iteration and then a sudden increase when many tasks are completed together.

Figure 8 summarizes the estimation versus actual times and the expected pace for each of the first four iterations for this team. We observe that estimates are too high in the first three iterations: 14% in the first (167 hours estimate vs. 144 actual) and second iterations (184 hours estimate vs. 159 actual) and 20% in the third (92 hours estimate vs. 74 actual). In the fourth iteration, in contrast, estimates were low by 4% (174 hours estimate vs. 180 actual).

In the three first iterations, the total-done time was less than the expected pace. In the first two iterations, the gap was 30% (144 hours vs. 207 and 159 hours vs. 229), and in the third iteration the gap was 48% (74 hours vs. 144). In the fourth iteration, the gap was –4% (180 hours vs. 174), which implies a certain improvement in the quality of the estimations.

Another issue that affected these measurements was the variability in team size. The reasons for the drop in the expected pace in the third iteration are personnel changes, travel of several teammates, and sick leave for two teammates.

Retrospective Process

The second governance point that we present concerns the retrospective process that guides the software development process. The agile approach use reflection sessions and a retrospective process as an internal practice (Beck & Andres, 2004; Salo, 2005; Talby et al., 2006). The teams in organizations A and B adopted the practice of one-hour reflection between the presentation of the artifacts of the previous iteration and the planning of the next iteration (Talby et al., 2006).

A reflection session was conducted in organization A between the first two releases of the transition-to-agile process. Teammates filled out an anonymous individual questionnaire containing questions mainly about the characteristics of the previous release and the expectations from the next one. We review some of the data that was collected during the discussion that followed. Reflecting on "the best thing that happened in the first release," these were the teammates' comments:

- "Focus on the goal with the customer's global view"
- "Cooperation"
- "It takes less time to develop tasks compared with the previous method"
- "End to end [development] shortens processes and integration"
- "A platform to talk about problem was created, sit together and hear problems"
- "Good communication between team and customers, there is someone to approach with questions"
- "People started new things, like modules"
- "Setting priorities gives focus"

Reflecting on "what can be changed in the next release," teammates generated a list and then voted for the following as the most important:

- Improve the integration process
- Adopt the methodology into the QA process
- Perform better follow-up

One of the first reflection sessions in organization B was dedicated to the relationships between the team and the organization. Teammates filled out an anonymous individual questionnaire containing questions about the organization's goals and values, existing policies, and the way the team could contribute to the organization and vice versa. One of the questions asked them to rate their level of agreement with several statements on the organization's existing policy on software quality. Table 1 shows the teammates' answers; the number in each cell represents the number of people who gave that answer. Based on this and similar data, we suggest that the reflection sessions can be considered as a governance mechanism that is activated in the Execute phase and feeds the Assess phase of the governance lifecycle.

In a broader perspective, the data that emerges from the retrospective process, together with other governance observables, can assist the governance follow-up in the Assess phase. During the Assess phase, a complete perspective on the governance solution lifecycle is gained. Specifically, we can follow up the transition-to-agile process and continuously steer it according to the governance information.

CONCLUSION AND FUTURE DIRECTIONS

Starting from a comprehensive definition of governance and its components, we presented a high-level governance model, which includes the main components and the relationships between them. To validate the model framework, we showed how to conceptually instantiate it for both IT governance and software development governance, in the context of transition-to-agile software development in large organizations.

Table 1. Teammates reflection on the organizational policy on software quality (majority is marked with grey)

Statement	1 Strongly disagree	2	3	4	5 Strongly agree	No answer
I'm familiar with the policy		2	1	4	5	
I follow the policy in my project	1		6	3	2	
Most projects in the organization follow the policy		3	2	4		3

We are now developing a governance lifecycle platform based on the model presented here. It should serve as a single point of administration for the governance of software development activities. The main parts of the governance solution platform are the governance module, the data module, the scheduler, and the user interface.

In brief, the governance module manages the governance lifecycle by supporting the governing body and relevant roles. The data module contains a data adapter that mediates between the application and the database. The database includes all the information from the different data sources available in software development environments:

- Software development artifacts such as code, test, specifications, models, and bug list
- Software management artifacts such as task plans and estimation graphs;
- Activity indicators that capture the state of the activities and tasks being performed; and
- Governance observables such as measures, policies, decision rights, and roles

The scheduler is responsible for scheduling tasks for governance mechanisms that are used within the governance solution. The user interface presents views appropriate to each of the different roles that are involved in the governance process.

In light of the development of the governance lifecycle platform, we suggest extending the development component of existing governance tools according to our model. This way, the model presented in this chapter can be used to augment existing IT governance models (e.g., CobiT and ITIL) to close the gap between the IT governance model and the governance of software development activities.

ACKNOWLEDGMENT

We would like to thank Clive Gee and Duncan Clark for their significant contribution to the UML model. We would like to thank Murray Cantor, Duncan Clark, Clive Gee, John Falkl, Steve Graham, Christine Draper, Greg Rader, and Calvin Powers for numerous discussions and ideas.

Orit Hazzan participated in consulting for one of the organizations. We would like to thank her for fruitful discussions.

REFERENCES

Abrams, C., von Känel, J., Mueller, S., Pfitzmann, B., & Ruschka-Taylor, S. (2006). Optimized enterprise risk management (Research Report RZ3657). IBM. Retrieved May 14, 2008, from http://domino.research.ibm.com/library/cyberdig.nsf/papers/0C4791FC96DEF130852571D0003F5F15/$File/rz3657.pdf

Ambler, S. (2007). Agile adoption rate survey: March 2007. Retrieved May 14, 2008, from http://www.ambysoft.com/surveys/agileMarch2007.htm

Barnett, L. (2006). And the agile survey says. Agile Journal March 6, 2006. Retrieved May 14, 2008, from http://www.agilejournal.com/articles/from-the-editor/and-the-agile-survey-says%85.html

Beck, K., & Andres, C. (2004). Extreme programming explained (2nd ed.). Boston, Massachusetts: Addison-Wesley.

Beck, K., & Fowler, M. (2000). Planning extreme programming. Boston, Massachusetts: Addison-Wesley.

Broadbent, M. (1998). Leading governance, business and IT processes: The organizational

fabric of business and IT partnership. Findings Gartner Group, 31 December 1998, document #FIND-19981231-01.

Cantor, M., & Sanders, J. (2007). Operational IT governance. Retrieved May 14, 2008, from http://www.ibm.com/developerworks/rational/library/may07/cantor_sanders/index.html

Chulani, S., Clay, W., Yaeli, A., Wegman,, M. N., & Cantor, M. (2006). Understanding IT governance: Definitions, contexts, and concerns (Research Report RC24064). IBM. Retrieved May 14, 2008, from http://domino.research.ibm.com/library/cyberdig.nsf/papers/38905EEA124CDDFB852571FE00569CCE/$File/rc24064.pdf

Dahlberg, T., & Kivijärvi, H. (2006). An integrated framework for IT governance and the development and validation of an assessment instrument. In Proceedings of the 39th Hawaii International Conference on Systems Sciences, Kauai, Hawaii.

Dubinsky, Y., & Hazzan, O. (2006). Using a role scheme to derive software project metrics. Journal of Systems Architecture, 52, 693–699.

Dubinsky, Y., Hazzan, O., Talby, D., & Keren, A. (2006). System analysis and design in a large-scale software project: the case of transition to agile development. In Proceedigns of the the Eigth International Conference on Enterprise Information Systems, Paphos, Cyprus.

Ericsson, M. (2007). The governance landscape: Steering and measuring development organizations to align with business strategy. Retrieved May 14, 2008, from http://www.ibm.com/developerworks/rational/library/feb07/ericsson/

Hallows, J. E. (2001). The project management office toolkit. Newton Square, Pennsylvania: Project Management Institute.

Hazzan, O. (2002). The reflective practitioner perspective in software engineering education. The Journal of Systems and Software, 63(3), 161-171.

Highsmith, J. (2002). Agile software development ecosystems. Boston: Addison-Wesley.

Kerth, N. L. (2001). Project retrospectives: A handbook for team reviews. New York: Dorset House Publishing Company.

Lewin, K. (1948). Resolving social conflicts; Selected papers on group dynamics. New York: Harper & Row.

Salo, O. (2005). Systematical validation of learning in agile software development environment. In Proceedings of the the Seventh International Workshop on Learning Software Organizations, Kaiserslautern, Germany.

Schön, D. A. (1983). The reflective practitioner. New York: Basic Books.

Tague, N. R. (2004). The quality toolbox (2nd ed.). Milwaukee, WI: ASQ Quality Press.

Talby, D., Hazzan, O., Dubinsky, Y., & Keren, A. (2006). Reflections on reflection in agile software development. In Proceedings of the Agile 2006 Conference, Minneapolis, Minnesota.

Van Grembergen, W., & De Haes, S. (2004). IT governance and its mechanisms. Information Systems Control Journal, 1. Retrieved May 14, 2008, from http://www.isaca.org/Template.cfm?Section=Home&Template=/ContentManagement/ContentDisplay.cfm&ContentID=16771

Weill, P., & Ross, J. W. (2004). *IT governance*. Watertown, MA: Harvard Business School Press.

Williams, P. A. (2005). *The buck stops here. IT Leadership, 1.* Retrieved May 14, 2008, from http://www.the-itleader.com/features/feature258/

ENDNOTES

1. Control Objectives for Information and related Technology (CobiT®); see http://www.itgi.org.
2. Details on ITIL® can be found at: http://www.itil.co.uk.
3. Information on ISO-17799 security controls can be found at: http://www.iso-17799.com,
4. The Agile Manifest at www.agilemanifesto.org.
5. Val-IT Governance Framework: www.isaca.org/valit.
6. Open Compliance & Ethics Group (OCEG) Foundation redbook: http://www.oceg.org/view/Foundation.
7. Capability Maturity Model Integration (CMMI): http://www.sei.cmu.edu/cmmi.
8. The Software Engineering Body of Knowledge (SWEBOK): http://www.swebok.org.
9. Merriam-Webster Online Dictionary definition at http://www.merriam-webster.com. Other dictionaries offer similar definitions.

This work was previously published in Information Technology Governance and Service Management: Frameworks and Adaptations, edited by A. Cater-Steel, pp. 266-284, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.24
Domain-Specific Language for Describing Grid Applications

Enis Afgan
University of Alabama at Birmingham, USA

Purushotham Bangalore
University of Alabama at Birmingham, USA

Jeff Gray
University of Alabama at Birmingham, USA

ABSTRACT

Grid computing environments are dynamic and heterogeneous in nature. In order to realize application-specific Quality of Service agreements within a grid, specifications at the level of an application are required. This chapter introduces an XML-based schema language (called the Application Specification Language, ASL) and a corresponding modeling tool that can be used to describe applications in grid computing environments. Such application descriptions allow derivation of guided and autonomic service developments for installation and invocation routines throughout the grid. In order to promote the language and ease the application description process, a domain-specific tool is also introduced. Based on our experience, the ASL in combination with higher level models improves, simplifies and promotes the grid application deployment process while simultaneously minimizing tedious and error-prone tasks such as manual application description composition.

INTRODUCTION

Grid computing (Foster, Kesselman, & Tuecke, 2001) has gained popularity as the emerging architecture for next-generation high performance distributed computing. Grid computing provides ubiquitous access to distributed high performance computing (HPC) resources that are shared between multiple organizations through virtualization and aggregation. This is realized through a layer of software (e.g., grid middleware), thus

making grid applications extremely software-intensive systems where the value of software is equivalent to the value of the underlying infrastructure. Grid middleware provides a standard set of services for authentication, authorization, resource allocation and management, job scheduling, submission, monitoring, and data transfer and management (Berman, Hey, & Fox, 2003b). Software packages and tools based on open-source/open-standard approaches such as Globus Toolkit (Foster & Kesselman, 1997) have enabled the deployment of "production quality" computational grids. Several domain-specific grids are currently operational, for example, Grid Physics Network (GriPhyN) (GriPhyN, 2006), Network for Earthquake Engineering Simulation (NEES-Grid) (NEES, 2006), International Virtual Data Grid Laboratory (IVDGL) (IVDGL, 2006), Open Science Grid (Grid, 2007), and Particle Physics Data Grid Collaboratory Pilot (Grid2003) (PPDG, 2006). Grid computing has offered researchers enormous computing and data storage capabilities by providing seamless access to geographically distributed resources through the creation of virtual organizations (VOs).

Despite the many benefits of grid computing, the grid itself does not provide a novel programming paradigm for developing new applications. Furthermore, no formal methodology exists for porting existing legacy applications to the grid. Most of the applications developed for the grid are based on traditional HPC or distributed computing principles. Typical HPC applications are developed using implicit parallel programming techniques (e.g., compiler-based automatic parallelization and directive-based parallelization) or explicit parallel programming techniques (e.g., threads and message-passing). After an HPC application is developed and tested on local resources, it is then deployed; that is, the entire application (source code, dataset, scripts) is transferred to a remote site, compiled on a remote host, and made available for execution.

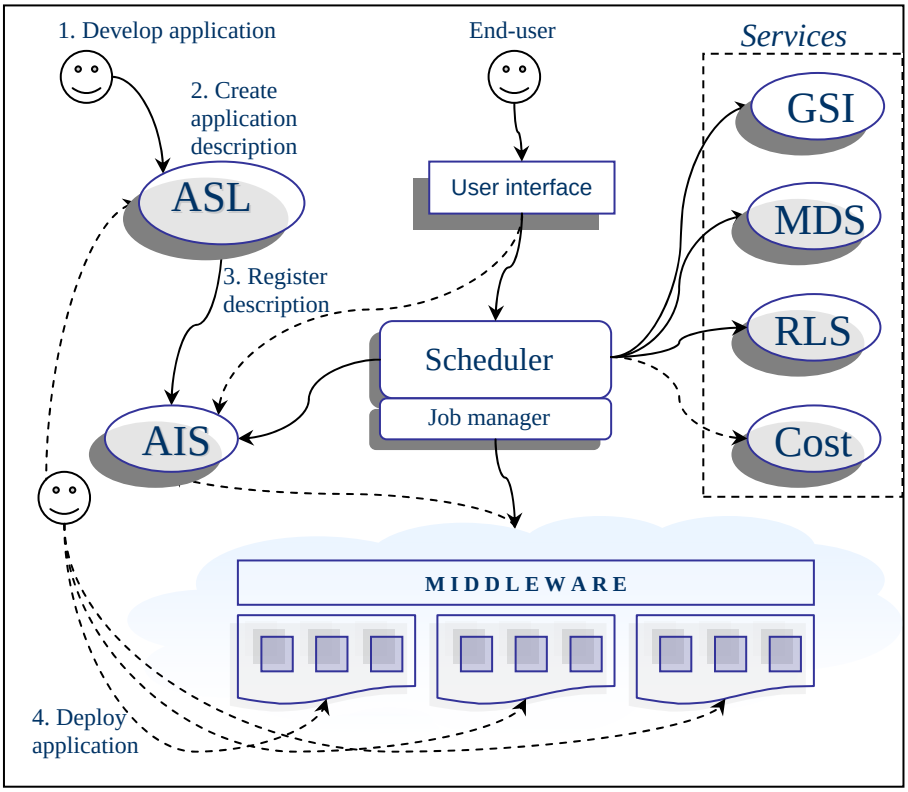
Deploying an application on the grid requires additional steps that involve user intervention, great insight into the internal structure of the application, as well as familiarity with the various grid computing technologies and toolkits. In addition, the progressive steps of application execution and job submission may involve many additional steps required from the end-user, not necessarily found in a typical application. Due to this inherent complexity and difficulty using the grid, several approaches have attempted to simplify grid deployment and configuration by developing technologies such as Web portals (Gannon et al., 2003), workflow systems (Aalst & Hee, 2002), and component assembly (Armstrong et al., 1999). The ultimate goal of such efforts is to enable the adoption of grid technologies and applications to a wider group of end-users who are not familiar with programming languages and the lower level grid infrastructure. The potential impact for improving grid accessibility to such users is significant (e.g., applied science researchers, distributed organizations, and organizations with variable computational requirements).

To tailor the complexities of a user's view of the vast amount of grid software by addressing the goals of end-user accessibility, there is a need to standardize and simplify the process of application deployment on the grid. As part of the contribution of this chapter, we present a new language called the Application Specification Language (ASL) (Afgan & Bangalore, 2007) that can be used by application developers and end-users to describe details of a given application. The ASL allows an application to be represented in the heterogeneous world of the grid by capturing its functionality, options and differences as compared with other applications found in the grid. Through the use of ASL, application descriptions can be made available for immediate use or further advancements among applications such as job schedulers, automated interface generators and application-specific on-demand help provisioning. The ASL

can also be used to describe how an application is combined with other matching services and software. The ability to specify the composition of services can facilitate the creation of new and added functionality, as well as enable further advancement of existing tools that can take advantage of the provided information.

To ease the process of collecting application description data and the description generation, a domain-specific modeling language (DSML) has been designed to assist in automating the grid application description and deployment processes. We believe that higher level models will improve, simplify and promote the grid application description and deployment process while minimizing tedious and error-prone tasks such as manual composition of application descriptions. The process of using the ASL and DSML within the context of grid deployment is shown in Figure 1, which outlines the major steps required for grid application development, publication and deployment. This figure highlights the global interaction between grid components and summarizes the ideas presented in this chapter. Upon development of an application, the developer can provide an application-specific description of a new application in the language described in this chapter. The application is registered by a service that can subsequently be queried by users, schedulers, or other tools. Information in the application description provides necessary data to enable automated integration with other available tools and services (e.g., Grid Security Infrastructure (GSI) (Foster, Kesselman, Tsudik,

Figure 1. Architecture using existing grid middleware technologies coupled with the technologies introduced throughout this chapter to enable improved application description and deployment



& Tuecke, 1998), Monitoring and Discovery Service (MDS) (Czajkowski et al., 1998), Resource Specification Language (RSL) (Czajkowski et al., 1998), Application Information Service (AIS)) as well as services still in development (e.g., cost). The focus of this chapter is on step 2 in the figure, whose addition to the global picture provides missing but beneficial functionality in the overall infrastructure. The dotted lines in the figure indicate possible interaction with a set of external events.

The rest of this chapter is organized as follows. The next section introduces grid computing concepts and provides a taxonomy of grid users. This is followed by a section that builds on grid technologies and provides examples of grid application development and deployment through a sample scenario outlining some of the difficulties with the current grid technologies. These issues are addressed in the section entitled *Application Specification Language*, which provides an introduction of a new language. The construction of this language is eased by a support tool based on metamodeling technology introduced and described in the *Modeling Wizards to Generate ASL* section. The end of the paper presents future trends and conclusions.

AN OVERVIEW OF GRID COMPUTING

Grid computing represents one of the new frontiers of computing. As the grid matures, many existing technologies will have to be revised and adopted to incorporate new ideas and concepts required by this evolving environment. Furthermore, many new technologies will arise as the result of emerging opportunities. This section introduces basic concepts of grid computing and presents a user categorization within this paradigm.

Grid Computing Background

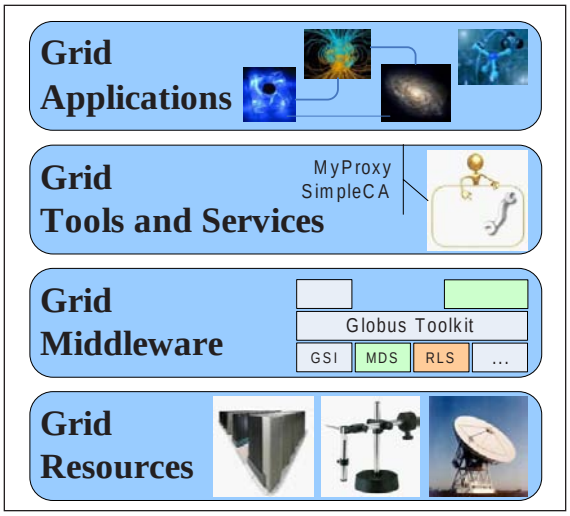
The grid community has witnessed increased adoption over the past several years as the emerging architecture for next-generation high performance distributed computing. Grid computing is a culmination of distributed computing (Tanenbaum & Steen, 2002) and high-performance computing (Kumar, Grama, Gupta, & Karypis, 1994). The grid integrates networking, communication, computation and information to provide a virtual platform for theoretically unlimited compute power and data management (Berman, Fox, & Hey, 2003a). It is designed to provide ubiquitous access to the vast number of resources and enables sharing between multiple organizations providing desired resource virtualization and aggregation. Virtual Organizations (VO) (Foster et al., 2001) represent aggregations of communities that may share national and international boundaries but have common objectives. Through VOs, grid computing is capable of aggregating heterogeneous resources that belong to different administrative domains to create a unique and valuable resource for the scientific community.

The VOs that are available in the grid are enabled through the use of middleware that provides a standard set of services for authentication, authorization, resource allocation and management, job scheduling, submission, and monitoring. Through this model, individual resources within an organization are virtualized into a common pool of accessible resources. The Globus Toolkit (GT) (Foster & Kesselman, 1997) is the most popular grid infrastructure framework for enabling the creation of grid applications. The GT offers all the basic technologies required for setting up a grid environment to support user creation, job submission and monitoring that span traditional boundaries (e.g., single institution environment setup). The technologies associated with the GT include:

- Grid Security Infrastructure (GSI) (Foster et al., 1998) is used for user authentication;
- Global Resource Allocation Manager (GRAM) (Czajkowski et al., 1998) assists in remote resource job management;
- Monitoring and Discovery System (MDS) (Czajkowski, Fitzgerald, Foster, & Kesselman, 2001) offers information discovery and state of resources; and
- GridFTP (Allcock et al., 2001) provides file transfer capabilities in the grid environment.

On top of the low-level grid middleware tools, an additional layer of grid system software packages provides an abstraction layer that hides complexities such as authentication, resource discovery, and file transfer. These intermediate layers offer direct interaction with the grid middleware allowing communication at a higher level of abstraction. Examples of such packages are portals (Gannon et al., 2003; Novotny, Russell, & Wehrens, 2004) and application schedulers (Afgan, 2004; Buyya, Abramson, & Giddy, 2000; Huedo, Montero, & Llorente, 2004) that provide a solid layer for using the grid as well as many options to develop grid applications.

Figure 2. Layered structure of grid computing



The top-most layer found in grid computing is the application layer that interacts directly with an end-user. A grid application is considered to be any application that is capable of exploiting grid resources through the use of grid middleware. The possibilities for using the grid are restricted only by the usability of the system as a whole. The end-result is a highly software-intensive system where, on top of the middleware functionality, lays the unlimited layer of application software offering a myriad of options and combinations to be explored by application developers as well as end-users. Figure 2 shows the overall layered structure of grid computing.

Because the grid is still an emerging technology, applications that were originally developed to run in more traditional environments may be adopted to take advantage of the benefits offered by grid technologies. The process of converting an existing legacy application to make use of the grid is called *application deployment* or *grid enablement*. The primary benefit an application gains, once grid-enabled, is the ability to use compute power more efficiently. Another benefit is that long running jobs can be broken up to execute in parallel on multiple resources to decrease turnaround time or increase efficiency. Additionally, larger data sets can be stored as well as processed without putting the load on any single system. By evolving a legacy application, the heterogeneity of the grid can be exploited to benefit the application as a whole by executing different application segments on separate (i.e., heterogeneous) systems making each segment take maximum advantage offered by underlying hardware, and in the end, gain maximum benefit for the application as a whole (Chase, 2005). With the diverse pool of available resources and the wider pool of available applications, ensuring mutual compatibility and maintainability becomes a major challenge. Some of the difficulties arise between the involved parties because individual requirements are not

stated in a standard, clear, and concise way and thus provide no opportunities for automation or standardized reliability.

GRID USER CATEGORY CLASSIFICATION

This section provides an overview of the roles and goals of individual grid users. The section identifies the specific requirements that are imposed by each user category, as well as the provisioning tools that accommodate their needs. The categories of users introduced in this section are middleware developer, application deployer, application developer, resource owner, and end-user. In some cases, a single person may play multiple roles (e.g., application developer and deployer).

Middleware Developer

As the interconnecting fiber of the grid, middleware represents the mechanism that facilitates the management of underlying resources as well as development of grid applications. Grid middleware is capable of delivering and sharing compute and data resources over secure channels while hiding intrinsic platform differences among the available resources. The middleware developers are key participants in the grid ecosystem and are closely connected to all other user categories. Middleware developers consider feature requests from user groups and participate in the design and implementation of the middleware standards that shape the way the grid operates. Examples of grid middleware include: the Globus Toolkit, Message Passing Interface (MPI) (Forum, 1998), Condor (Litzkow, Livny, & Mutka, 1988), Storage Resource Broker (SRB) (Baru, Morre, Rajasekar, & Wan, 1998), Web portals (GridSphere (Novotny et al., 2004), and the Open Grid Computing Environment (OGCE) (Gannon et al., 2003).

Application Deployer

An application deployer has the responsibility of performing the necessary work to grid-enable a legacy application. Due to the well-known possibility of high costs and challenges of application modification, the community focus is not on modifying the source code, but rather adopting the application through creation of wrappers that convert the entire application into a grid-enabled solution (Afgan, Sathyanarayana, & Bangalore, 2006; McConnell, 1996). The deployment process and wrapper creation for one of the five strategies is performed using a combination of the following techniques:

- Create script interfaces for command-line tools.
- Program directly to the Globus Toolkit using an available C API.
- Use a commodity toolkit such as Java CoG kit (Laszewski, Foster, Gawor, & Lane, 2001).
- Use the Grid Application Toolkit (GAT) (Allen et al., 2005).
- Use Grid Services (Sotomayor & Childers, 2005).

Because the application deployer is working with an existing application that may have several peculiarities, as well as the heterogeneity of grid resources, many issues may arise offsetting expected application performance or even correct execution. One of the causes of the evolution problem is the lack of standardized application information (Ulrich, 2002). The application deployer must placate the existing user base and should not modify the original application requirements to address the grid-related concerns. If any modifications are required during the deployment process, to better accommodate needs of either end-user or the resource owner, currently, there

is no effective way to convey the modifications that took place (as compared to the descriptive information pertinent to the original application) to either of those users categories.

Application Developer

As the grid gains acceptance, more applications will be custom tailored to take advantage of the benefits offered by the grid. Traditional programming techniques and methodologies (i.e., sequential or parallel) have their own set of challenges. Incorporating the grid into a legacy application poses a new set of accidental complexities. Most of the programming techniques and application types available when developing grid applications come from the HPC area or traditional distributed computing.

Grid services, which are based on the Service-Oriented Architecture (SOA) (Foster, Kesselman, Nick, & Tuecke, 2002a), represent the future of grid application development. However, grid services are hard to develop and deploy, as evidenced by the following steps that are required to develop a single, standalone service (Sotomayor & Childers, 2005):

1. Define and specify the service's interface so clients know how to invoke it.
2. Implement the given interface for the service; implement any related libraries.
3. Define the deployment parameters to indicate the specifics of the service.
4. Deploy the service to a grid service container.
5. Register the service so it can be discovered by other services.

In addition to coding difficulties, other issues arise when developing grid applications. These issues include security, licensing, job management, and synchronization. Another major step in application development is aggregation of newly created services or already existing ones through library calls and advanced programming techniques. Application creation will need to be tailored to extract maximum benefit from the grid for a wider community of users. Examples of such use include advertising application requirements (e.g., service name, installation instructions, license requirements), as well as minimum hardware requirements (e.g., CPU speed, underlying network infrastructure, memory consumption). Unfortunately, current methods do not provide an adequate solution to address these issues.

Resource Owner

Resource owners and providers assist in running the most basic layer of the grid multitier architecture, which consists of the resources (e.g., networks, computer, storage nodes), the middle-tier (e.g., middleware, certificate authority, portals), and the clients (e.g., GT clients, GUIs). Resource owners are responsible for maintaining their respective resources by meeting hardware requirements for individual applications, installing those applications, obtaining relevant libraries and compilers, maintaining applications, and enforcing policies. Beyond the initial cost of installing the grid environment, there is the natural cost of running individual clusters, as well as keeping licenses, subscriptions, and libraries up to date. The resource owners must also be cognizant of the Quality of Service (QoS) agreements between participating users (e.g., compatibility, application versioning, standardization). In order for grid computing to move further into the mainstream, resource providers must be able to reduce their operating costs, increase resource utilization and find a benefit in sharing their resources with the wider community.

End-User

The role of the resource owners is aligned to the bottom of Figure 2. At the top of Figure 2 are end-users who rely upon the results of a grid

application. In order for the grid to become a commodity computing technology, there is a need to attract a wide variety of users. Some of the issues preventing widespread adoption are the complexities and dynamics involved in job submission and job management, which are not yet tailored for lay-persons (i.e., those who are not computer savvy). Consider the following scenario that exemplifies the manual steps needed to submit a job on the grid:

1. Select the resources needed by an application.
2. Create a proxy certificate (e.g., grid-proxy-init).
3. Copy the necessary source code and input files to a remote host (e.g., globus-url-copy).
4. Create a Resource Specification Language (RSL) (Czajkowski et al., 1998) string.
5. Submit the job for execution (e.g., globus-job-run, globus-job-submit).
6. Copy output files generated from the remote host to local machine (e.g., globus-url-copy).

Although the general concept of the above steps exists in many distributed contexts, the introduction of the grid manifests as an additional obstacle toward application usage. As such, the grid requires a level of computer expertise that is not within the skill sets of general end-users. The complexities of manual job submission, typified by the above six steps, can be ameliorated by the following user-friendly approaches:

- Write a client using a commodity toolkit (e.g., Java CoG Kit (Laszewski et al., 2001)).
- Write an interface to a grid service.
- Use a portal (e.g., OGCE (Gannon et al., 2003) or a grid-port (Thomas, Mock, Dahan, Mueller, Sutton, & Boisseau, 2001)).
- Use a workflow systems (e.g., Chimera (Foster, Voeckler, Wilde, & Zhao, 2002b)).
- Use a component framework (e.g., Armstrong et al., 1999).

All of the above techniques are viable choices and present gradual improvements in usability, customization and acceptance. Whichever layer of abstraction the above techniques provide, one common goal is to simplify job submission by abstracting grid resources from the end-user.

From the grid user classification and description, there are numerous requirements imposed by each of the user groups. This point is further complicated when user roles are combined and intertwined with a need to provide an effective way to address individual user concerns from application development and registration to application usage. By helping to solve this problem, users will be empowered to create and use grid applications while improving overall grid acceptance and resource utilization.

GRID APPLICATION DEPLOYMENT SCENARIO

Software engineering researchers have introduced many helpful principles (e.g., information hiding (Parnas, 1972)) and best practices (e.g., design patterns (Gamma, Helm, Johnson, & Vlissides, 1994)) with the goal of simplifying the development process. Despite the improvements offered by traditional software engineering principles, there still remain many open questions when it comes to new technologies such as the grid. For example, developing applications to execute in parallel computing environments changes some of the underlying assumptions; this is due to the focus of performance as the primary concern, rather than maintainability, evolution, and changeability. Parallel computing (Kumar et al., 1994) is based on a set of models primarily governed by the communication patterns between algorithmic iterations within an application. The following categories represent various types of parallel programs based on communication patterns (full

details and descriptions of the individual sections are provided in the section entitled *Application name and description*):

1. Sequential applications
2. Parametric Sweep applications
3. Master-Worker applications
4. All-Worker applications
5. Loosely coupled parallel applications
6. Tightly couple parallel applications
7. Workflow applications

Development of parallel applications entails coordination between computation and communication as limited by the underlying hardware architecture. Depending on the problem type, one communication pattern may be a better choice over another based on the constraints of a specific application context. Once developed, the application is deployed on a targeted architecture. Occasionally, hardware-specific optimizations are performed to customize the code for optimal execution. Developing applications for parallel computing environments is guided by the concept of overlapping communication with computation, thus maximizing the use of resources. Although the optimal solution of any application is rarely achieved, the actuality of the application executing on a single, dedicated resource results in a reasonably understood platform to build upon. On the other hand, developing the applications for the grid under the constraints of parallel computing introduces new obstacles and complications. The execution resource becomes a virtual entity occasionally available and with a high failure rate. The communication between iterations and application modules is not only exponentially slower but also highly unstable. The development of such applications raises the complexity considerations to a higher level and introduces a whole new field of application deployment as a significant part in the application lifecycle.

Most current grid environments are based on the concepts embodied in Service-Oriented Architecture (SOA) (Foster et al., 2002a) and Web Services (Gottschalk, Graham, Kreger, & Snell, 2002). Applications deployed on existing grids are submitted as grid services, which are Web services with extended functionality to include support for stateful behavior, lifecycle management, and notifications (Foster et al., 2002a). The development of a grid service proceeds in several steps requiring expertise in many fields (Sotomayor & Childers, 2005). When developing traditional parallel applications, it is required for individual subtasks to work toward a common goal. The same idea transfers to the context of the grid, but additional difficulties arise because many applications that already exist and conform to this subtask model are not grid-aware, which requires major reengineering. The bottom line is that many existing software applications developed by the parallel applications community will not be rewritten to comply with the grid model, but will rather be adopted by the use of wrappers. This introduces new obstacles because different user categories must interact to achieve the common goal. Originally, an application developer was able to develop an application and deploy it on a resource because they possessed the required expertise. By adopting the grid, the requirements related to application deployment have risen and require new expertise, often not readily and easily available due to the involved complexities. This implies the need for a new domain expert (i.e., a grid specialist) who possesses deep understanding of the technology and is able to transform traditional applications into grid applications. The drawback to this solution is that additional communication is necessary at the development level, which prolongs the application development. Furthermore, the delivery process is perplexed with new possibilities to introduce errors due to miscommunication or misunderstanding.

The application deployment process on a grid is a nontrivial task. The user must first determine what resources are available and then decide the most suitable resource for a particular applica-

tion. It is important to differentiate the application development process from the application deployment process. As stated earlier, typical HPC applications are developed using a specific programming language and parallel programming paradigm (e.g., compiler directive-based, threads, message-passing, combination of threads and message-passing) and often the programming paradigm chosen decides the application deployment platform. If the application uses a shared-memory programming paradigm then the application can be deployed only on a shared memory system, whereas an application developed using the message-passing paradigm can be deployed on both distributed memory and shared memory systems. Furthermore, applications might have specific requirements (e.g., processor architecture, amount of memory, disk space, interconnection network, libraries, operating system) to deliver desired performance and scalability. An application developer could describe these requirements using application deployment descriptors, or hints could be added about various performance implications and space/time tradeoffs. For commercial software packages, information about licensing and subscription could also be provided by the deployment descriptors. An application scheduler may use these descriptors to select a suitable resource and schedule the application for execution.

As a concrete example showing the full process of application deployment, we will outline the options and necessary steps required to enable an application such as the Basic Local Alignment Search Tool (BLAST) (Altschul, Gish, Miller, Myers, & Lipman, 1990), a frequently used bioinformatics application to perform sequence similarity searches, to execute on a local campus grid, such as UABGrid (Gemmill & Bangalore, 2005). Portions of the following steps assume the availability of grid middleware such as the Globus Toolkit (GT) (Foster & Kesselman, 1999). This sample scenario assumes that a user authentication system is in place and that the user is able to access available resources using middleware tools. The following represents the scenario:

1. Obtain a valid user authentication proxy.
2. Manually log into each of the available resources and download BLAST.
3. Install BLAST individually on each resource.
4. Create a wrapper application that is responsible for necessary file transfers, iterative job submissions, job monitoring and notification using a grid middleware API or a commodity toolkit such as the Java CoG Kit (Laszewski et al., 2001). The wrapper developer must be aware of the method of interapplication communication and design the wrapper to accommodate for the appropriate task parallelization and communication methods with respect to the grid middleware. Also, additional levels of parallelization can be introduced at this level as is the case with a locally developed wrapper (Afgan et al., 2006). If a scheduler is not available for task scheduling among grid resources, this feature should also be included as part of the wrapper functionality.
5. Create a user interface where the application can be invoked. This can range from a set of command-line tools to a Web-based portal.

As can be seen from the given scenario, the process of enabling an application to execute on the grid requires a significant effort and high-level of expertise from the application deployer. Depending on the complexity of the wrapper, it may turn into a completely new application developed to maximize the potential of the original application in this new environment. One of the drawbacks is the requirement of the deployer to have a reasonably deep understanding of the application to ensure that the wrapper is correctly implemented.

APPLICATION SPECIFICATION LANGUAGE

Motivation and Background

A goal of an application developer is to see their solution adopted quickly by a large group of end-users. However, the typical path to adoption requires frequent interaction with end-users to address numerous questions, such as: how to install the application, how to invoke it, the purpose and function of the available options and arguments, as well as how to improve performance for a specific platform configuration. Through the lifetime of the application, additional documentation is created to address such common questions. However, with a new version of the software, much of this work needs to be discarded and redone. Avoiding this pipeline of events is challenging, but through gradual and systematic adoption of the technique proposed in this chapter, it is possible to provide more automated support for the end-user.

Computing has evolved from a user-centric context (where every detail of code execution needed human attentiveness and interaction) to a global service-driven view (such as Web Services, where complex, data dependent, goal seeking interactions and computations of independently developed components can be achieved with absolutely no human intervention). From this progression, we observe the common approach where automatic communication between programs is enabled. This is generally accomplished through definition of standard languages that specify protocols observed between applications. The following languages represent efforts within the grid community to automate specific deployment tasks:

- **Job submission description language (JSDL) (Anjomshoaa et al., 2005):** A recently completed standard from the Job Submission Working Group (JSWG, 2007) within the Open Grid Forum (OGF) (OGF, 2007). The JSDL is a specification of an abstract and independent language used for describing requirements of computational jobs in grid environments. It contains a vocabulary and normative XML schema that builds on the idea of standardizing a language to accommodate a variety of job management systems, which alleviates the problem of heterogeneity in the grid. By having a standard language available, a job submission description can enable diverse job management systems to communicate and complement job description in a more simplified manner.
- **Resource specification language (RSL) (Czajkowski et al., 1998):** Provides a common interchange language to describe resources and jobs to run on them. It is a language developed by the Globus Toolkit Project (Foster & Kesselman, 1999) and is represented by various <*name, value*> pairs that are used by the Grid Resource Allocation Manager (GRAM) (Czajkowski et al., 1998) to perform complex resource descriptions in cooperation with other components in the system. RSL preceded JSDL, so RSL can be seen as an earlier scaled-down version of JSDL.
- **Resource description language (RDL) (Anjomshoaa et al., 2005):** A language that is part of the JSDL standard document for describing underlying grid resources in terms of CPU speed, number of CPUs, and main memory. Even though not yet realized, the concept of this language has been propagated through other tools such as Condor and ClassAds (Solomon, 2004) and the necessary information (e.g., current resource status) can be obtained from MDS (Czajkowski et al., 2001) and Ganglia (Massie, Chun, & Culler, 2004).

The above mentioned languages have been developed to enable standardized exchange of

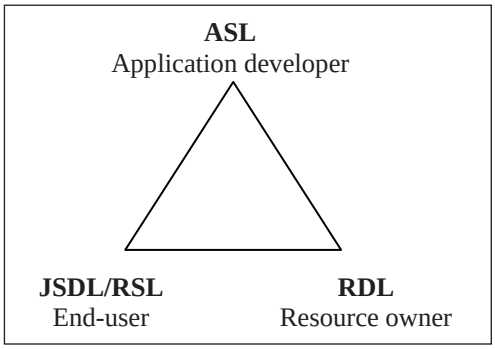
information between grid resources and provide support for direct and concrete communication between these resources. By defining documents specified by these language constraints, one can rely on automated negotiation during the job submission process without regard for the heterogeneity of the underlying hardware and software. When mapping the above mentioned languages to individual grid user categories, the JSDL and RSL map most favorably to the end-user category (where the user is mainly interested in adopting the grid as a pool of resources). The JSDL and RSL enable users to limit and perform resource selection. At the same time, RDL can be classified as a language for allowing resource owners to describe the capabilities of their resources and advertise that information for wider use through mechanisms such as MDS and Ganglia. What is evidently missing is the need to support the application developer. Once an application is developed, there is no standardized way to publicize the name and capabilities of the application in a manner that can be accessed by other tools. Rather, the application developer and the end-user are forced to interact in an interrogative manner, perhaps using a wiki or similar tools to advertise the availability of an application.

By supporting a method for capturing the core purpose of the application, requirements, and options, the end-user is provided with specific information that describes the application. After successful installation, the second most important feature enabling application use is the interface that the application provides to its users. With respect to grid applications, the most appropriate way to interact with an application is through a Web-based interface that requires no local installation of the application. A Web-based interaction may also provide special tools and knowledge to access available resources (Gannon et al., 2003; Hernandez, Bangalore, Gray, Guan, & Reilly, 2006). By providing a default standardized interface to the given application automatically, a resource owner may reuse the interface rather than implement their own. The benefit for the end-user is that the interface stays constant across different providers. An additional benefit is a reduction of possible errors in interface generation originating from the resource provider due to possible misunderstanding or lack of application knowledge. The Pasteur Institute Software Environment (PISE) (Letondal, 2000) is an example of previous work where this idea has been adopted in practice. PISE is a transformation tool that receives as input a PISE-DTD compliant XML document and interprets the document to create any of the suite of interfaces ranging from HTML to CGI and IPSH (Letondal, 2000). A scalable core is also provided by PISE that can be extended to add additional interface interpreters as needed. PISE currently contains a database of over 200 XML documents corresponding to interfaces for various applications that are primarily focused on bioinformatics. By leveraging ideas and technologies such as those provided by PISE, many of the accidental complexities associated with grid application deployment can be removed.

Description of the Language

To address the challenges raised in the previous sections, we present a new language called the Application Specification Language (ASL) (Afgan & Bangalore, 2007). As a new approach toward application specification that focuses on the needs of grid users and grid applications, the ASL is able to capture essential application information. Through standardized protocols, tools can be passed the information about an application that is specified in an ASL description. ASL is a language for describing any application's requirements, attributes, and options. The ASL directly supports the ability to capture application-specific information that is not necessarily found in the general pool of available description tags. Through the use of ASL, factors such as software and hardware requirements, data constraints, and algorithm complexity can be provided to a user. As

Figure 3. ASL-RDL-JSDL/RSL triangle showing direct communication paths between corresponding user categories



can be seen in Figure 3, the ASL may be composed with other groups of established grid languages (e.g., JSDL/RSL, RDL). The interactions implied in the triangle connect all perspectives and user categories of a grid environment, which enable communication to take place over well-designed paths to facilitate further communication, refinement, contract creation and the possibility of higher QoS for all participants.

ASL is applicable before and during installation, during job scheduling, during job execution, and even after the job has completed. It can be complemented and modified as knowledge about an application increases. The ASL can be used with legacy applications (requiring adaptation), or with newly developed applications designed specifically for the grid (often called "Smart Applications"). By providing a standardized way to describe application requirements, the ASL enables an automated capability to compare applications. Without ASL, such comparisons are very hard to perform manually because of their subjectivity. Such comparisons can be useful in numerous cases, such as application scheduling and software cost estimation (Afgan et al., 2006).

In essence, ASL is an extended application version of RSL. It provides a set of specialized tags used to capture application-specific details and thus provide a description of an application. The goal of this language is to use *application-specific descriptions* that enable deployment, maintenance, and execution of an application in a standardized and simplified way. The structure of the language is intended to describe entire, operational applications rather than individual components of an application or other software that may subsequently be composed into an application. The intent of ASL and individual ASL documents is to enable needed communication between heterogeneous resources in the grid through standard interfaces. Just like ASL's sister languages JSDL and RSL, ASL is not a grammar-based language, but rather a specification language establishing and defining a standard interface needed for heterogeneous grid resources to communicate with each other. A grammar-based language refers to a language that is defined and constrained by a context-free grammar. A schema-based language is rooted in an XML schema and is constrained by the tags defined in the associated schema.

By providing the appropriate set of tags, ASL enables application comparison and interface generation. Because every grid application is custom built to meet a certain need, the implementation details may be difficult to describe. Many of the options available during application specification often require significant human intervention as well as use of human language descriptions that cannot be modeled and captured by a general-purpose computer language (e.g., Java or C++). Providing a standardized set of tags to capture information about an application in a concise and precise manner is difficult. The requirements imposed when selecting a given set of tags must focus on capturing the core set of characteristics describing any application and then provide an extended set of tags that allow unique application components to be specified. Our first attempt at defining this set of tags considered existing languages such as JSDL and RSL, which capture job

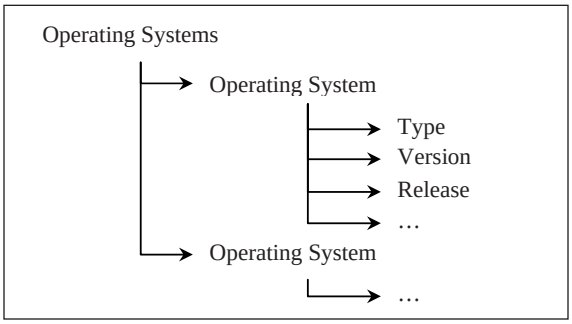
submission requirements that map onto resource and application requirements. Examples of such tags include numerical values (e.g., CPU speed and amount of main memory required), as well as a predefined set of values (e.g., operating system and CPU architecture type).

Additional tags were created by a systematic analysis of characteristics that describe an application but were not required for application invocation (e.g., max number of CPUs an application scales to). Many of these tags are simple in nature, allowing the definition of a range of valid values that can be used to validate data entered by the end-user. The more difficult set of requirements deals with values that are dependent on each other, but can be viewed individually as containers of simple values. Thus, these tags were organized in groups where subelements define individual pieces of the larger component. An example is the operating system requirement. An application may be suitable for many operating systems as well as different versions of an individual operating system. Thus, creating higher level elements that contain equivalent subelements allows different version dependencies to be specified (please see Figure 4).

One concern with adopting ASL, especially when viewed from the perspective of a developer creating the ASL document, is the requirement of the document syntax to be specified correctly (i.e., equivalent tags may have a different meaning when placed in different element groups). The most difficult part of describing an application concisely occurs with tags that cannot be constrained to a set of predefined values (e.g., tags that represent a human readable text string, such as copyright policy). The obvious impediment with such tags is the lack of precision needed for formal interpretation. However, the additional information provided within these tags can benefit end-user understandability of the application. The use of the tags for all types of descriptors (e.g., simple, complex groups and natural language) helps to partition the entire document and provide guided help for natural language descriptors. A further benefit of such tags is the possibility of developing additional translators to generate application Web documentation automatically.

The completed ASL document consists of several parts (discussed in subsequent sections of this chapter) each focusing on a particular portion of an application deployment lifecycle. With the blend of the formal tags (i.e., computer readable) and informal tags (i.e., end-user understandable), the ASL assists in application description from different perspectives and provides user support in multiple formats. Examples of interpretation include an application description Web page with installation and invocation instructions, script generation for automated application installation, as well as optimal system requirements for job submission.

Figure 4. Grouped set of elements and subelements with appropriate tags



Structure of the Language

An ASL document consists of four distinct yet related sections that are described in XML. By dividing the document into these distinct sections, an ASL document is more modular and allows for easier initial generation and later modification. With the use of appropriate tools, each section of the document can have its own permissions, which allows the document to be modified in-

dependently and securely. As the application receives a wider user base, additional information may become available from its users. As a result of multiple executions of an application, additional information can be gathered, such as profiles of application performance, unexpected behavior, or suggestions for future enhancements. Beyond the collection of application information, the segregation of the document into appropriate sections allows for shorter search times among users allowing them to focus on sections of the document of most interest. To provide segregation of information collection and retrieval, an ASL document consists of application name and description, installation requirements, job invocation requirements, and hints.

The sections are not directly connected to each other, but the data is stored only once per ASL document. Because of this, inadvertent references to information provided in other parts of the document may exist. These sections are correspondingly mapped to XML with appropriate tags. Each of the sections is described below and the schema is provided for the given section.

Application Name and Description

The application name and description section contains the most basic information about an application and acts as the application identification component. It specifies the name and version of the application as can be found in an application repository. This section also contains subelements such as the application description describing the application in a human readable format. The description identifies the problem the application solves and maps the application to an application category. The application category element is limited to a predefined set of values as described below. It is intended to offer better understanding of the application deployment process on the grid and it is essential in classifying different types of applications deployed in a grid environment. The majority of the applications typically deployed in a grid environment can be classified into the following categories:

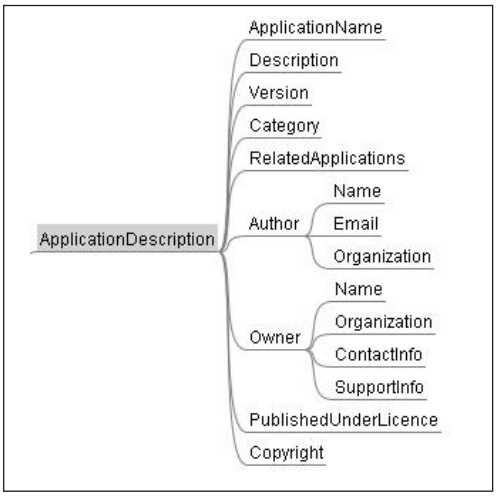
1. **Sequential applications:** Traditional applications developed to execute on a single node machine. For the applications that require larger resources (e.g., more memory, disk, or faster CPUs), the grid will also provide redundancy, fail-safe capability, and excess capacity.
2. **Parametric sweeps:** Multiple copies of sequential jobs using different input datasets or parameters. These applications are often submitted independently by a single user in an effort to reduce overall task execution time. Benefits of using the grid are the same as sequential applications with the addition of multiple instance coordination performed by grid tools and middleware.
3. **Master-Worker applications:** Master-worker or bag-of-tasks model (Kumar et al., 1994), where a master process distributes work (either statically or dynamically) to a set of worker processes and aggregates the results at the end. Many financial and bioinformatics applications fall into this category, each in constant need of surplus compute resources. The main differences between parametric sweeps and master-worker applications is that the individual tasks do not have to be executing the same code, but a workflow system can be in place with the master-worker model possibly delivering a more complex application functionality. The coordination between the worker nodes and task assignment must be handled by the master process.
4. **All-Worker applications:** Similar to the master-worker model, except that each process involved, including the master, share the workload equally and data is exchanged between individual processes in some pattern (point-to-point or group communication).

5. **Loosely coupled parallel applications:** Parallel applications (e.g., coupled fluid flow and wave models) that exchange data occasionally through files during execution (e.g., beginning and ending of an outer iteration).
6. **Tightly coupled parallel applications:** A single Message Passing Interface (MPI) (Forum, 1998) application distributed across multiple systems sharing data during the execution, possibly at frequent intervals, through passed messages. It requires interoperability between different MPI libraries or an MPI library such as MPICH-G2 (Karonis, Toonen, & Foster, 2003).
7. **Workflow applications:** A model connecting many individual applications executing at different geographically distributed locations, which are chained together to perform a complex simulation. For an application to be classified as a workflow application, additional information is needed to identify dependencies with other applications in terms of input and output data streams.

When selecting the application category for an ASL specification, the following considerations should be examined by a user composing the document. Applications that belong to categories (1) and (2) can be distributed and scheduled across any available computational resource on the grid because there is no synchronization or coordination required between individual tasks. However, applications in categories (3) and (4) must be scheduled on a single computational resource and cannot be distributed across multiple resources. This does not imply that the application may not use additional, distributed resources. If an application has been developed specifically for the grid, it can utilize middleware components to enable cross-resource task execution. In that case, task scheduling is the responsibility of the application itself because it would be deployed on a dedicated resource. Applications in categories (5) and (6) expect that the individual applications are distributed and assume that the individual tasks are scheduled to execute at the same time (through advanced reservation or mutual agreement with the resource providers). Workflow applications (category 7) assume that the scheduler can trigger the execution of one or more applications as described by the workflow. Because most of the existing schedulers (Berman et al., 2001; Casanova, Obertelli, Bernan, & Wolski, 2000; Huedo et al., 2004; Venugopal, Buyya, & Winton, 2005) do not handle advance reservation, the use of workflow applications is limited and intended mostly for future generations of applications and grid schedulers.

The remaining elements in this section are illustrated in Figure 5. *Category* element set has a predefined set of values from which a user must choose. The remaining elements found in this schema section do not have their values predefined, but can be defined by the person creating the document enabling desired application description.

Figure 5. Application description section schema

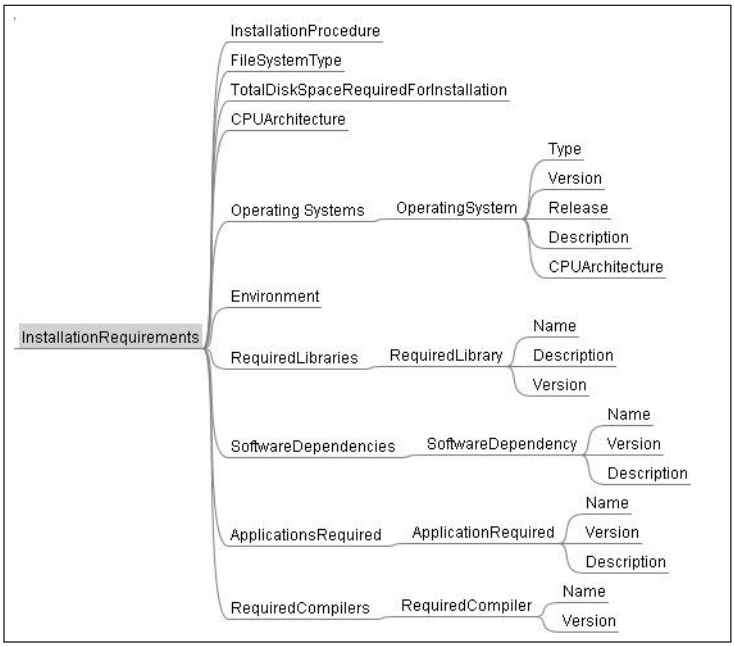


Installation Requirements

The installation requirements section of an ASL document contains a set of required elements that describe the pre-installation requirements as well as the installation procedure. Some of the examples of this type of element set include minimum processor speed, processor architecture, minimum amount of memory needed for installation of the application, libraries, applications required for the installation procedure (e.g., compilers, (un)packaging tools), licenses needed for application installation, network requirements, and required amount of disk space for the installation. The tags used are simple declarations that specify the value of a predefined type (e.g., string, integer). Even though this model may result in unnecessary inconsistencies between application descriptions, we believe at this stage of ASL development and definition this variability is necessary to allow for the correct words to be selected from a constrained set of choices. The full schema of the installation requirements section is given in Figure 6.

Among the elements defined in the installation category are *SoftwareDependencies* and *Applications Required* tags. The information these tags contain is intended strictly to be used during the installation procedure. The *SoftwareDependencies* tag refers to any other software that will be needed for the application execution. This can be viewed as a prerequisite for the installation; that is, in case software packages declared within this tag are not installed, the application cannot be expected to execute. Examples of such software dependencies would include Perl (Wall, Christiansen, & Orwant, 2000) with certain libraries and Postgres database (Momjian, 2000). With respect to installation, the *ApplicationsRequired* tag refers to other complete applications required to perform the installation. These applications may be invoked during the installation procedure, such as unpackaging and installation tools (e.g., Ant, make).

Figure 6. Application installation section schema



Job Invocation Requirements

The job invocation requirements section focuses on providing a user with the information needed to execute the application. Starting with the executable name, it also provides the available switches and minimum hardware requirements, as well as allows the developer to specify the number of input and output files with examples of their respective formats. This section does not represent a duplication of effort found in JSDL/RSL, but it is alternatively used to specify requirements for the entire application. Such specification is needed not only when executing a single job, but to describe the available options and how to use them. Rather than specifying exact input files and other job-specific parameters, the category defines application requirements, such as: the required input files, required format of those files, any output files and corresponding format of the output files, libraries required to invoke the application, and licenses needed to run the application. This capability can be viewed as a more detailed version of *man* pages in UNIX. This category allows the developer to be shielded through a contract-like document; that is, if any of the requirements are not met, the application cannot be expected to execute correctly.

The majority of the application description is provided in this section of the ASL document, so it is natural for a set of tools to be based on this category. An example tool is a translator that formats the appropriate information into a Web page allowing the information to be read through a browser, or a correct and application-specific job submission interface. Another example tool serves as a data verification tool that ensures input files are in the correct format. The complete schema is shown in Figure 8 on the next page. Similar to the installation section, the application invocation section has elements *SoftwareDependencies* and *RequiresApplications.* In this context, software dependencies refer to any software packages that are necessary and will be used as part of the ap-

Figure 7. Hints section schema

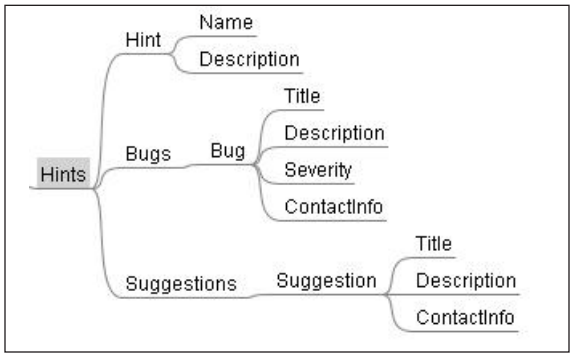
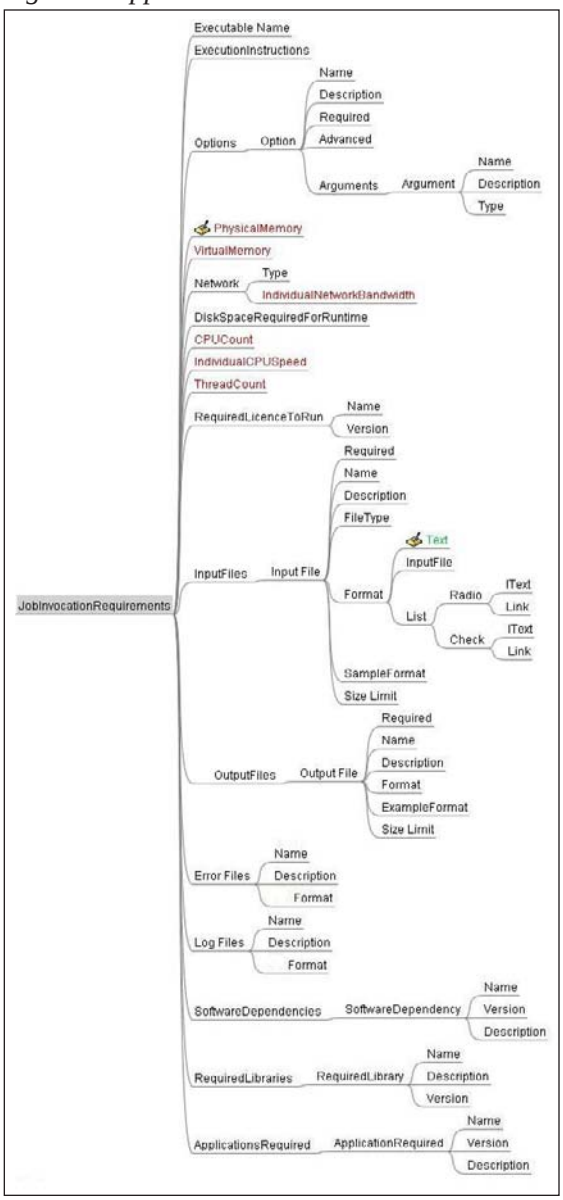


Figure 8. Application invocation section schema



plication during its execution. An example would include a call to a Perl module. The description of the application requirements tag is similar to the description from the application installation section of ASL, where it specifies any other applications that may be invoked during this application's execution. This tag can be used to specify the requirements for a workflow, even though any further enforcement and coordination during execution must be done within the given application.

Hints

Due to the inherent variability of applications, information describing an application may not be adequately captured in the preceding sections of ASL due to noncompliance and uniqueness of the application. Also, the succinctness of available options in ASL tags or already existing data may prevent additional and possibly more complete application information to be stored. In order to accommodate for these possible shortcomings, there is an additional section found in every ASL document, which is entitled *Hints.* This section contains instructions and comments, mostly in natural language, which provide additional information about the application. The purpose of this information is to allow detailed descriptions for areas of high application complexity, either for the users of the applications or other developers who may use this application as a base layer for development. Another important goal behind this section and its element set is that it can be accessed and edited by a wide user group. Performance information may be stored in this section to specify the optimal parameters on a particular hardware architecture.

Depending on application type (e.g., sequential, embarrassingly parallel, MPI-based application), certain input parameters (e.g., size of input file, input file format, number of processors) may alter application performance and thus information found here could be useful for the resource owner, end-user and even the scheduler developer. By giving permissions to a wide range of users, known bugs as well as suggestions for future advancements can be documented. A large portion of its use can be found in troubleshooting an application where expected errors can be explained. Figure 7 shows the current Hints section schema.

Example ASL Documents

This section demonstrates the use of ASL to describe an initial set of applications that show the ability of the language to specify and distinguish applications. In these examples, ASL was manually generated by providing the values associated with appropriate tags as defined in the schema. This generation is quite straightforward, which provides the user variability in selecting the tags to be defined depending on the application. The rest of this section provides snippets of ASL documents with their respective applications. Differences between applications are outlined and the ability of ASL to capture these differences is discussed.

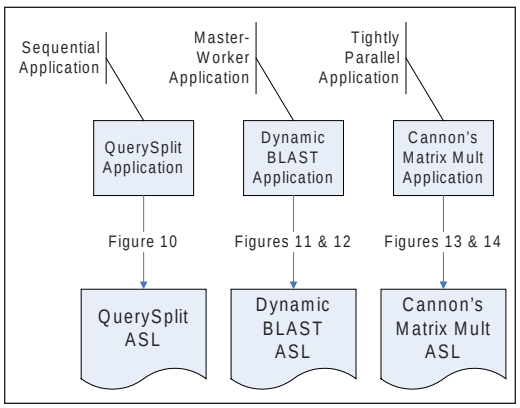
Application Descriptions

To show the ability of ASL to capture descriptions of applications belonging to different categories, three applications are specified, each from a different application category. The first two applications correspond to the sample scenario described in the *Grid Application Deployment Scenario* section. The first application is a sequential application implemented in Perl called *QuerySplit* that performs segmentation of the BLAST input query file (please see Figure 10). The *QuerySplit* application takes a text file as a parameter, which contains query sequences of variable length. It then proceeds to analyze the file and create several smaller files, each containing a number of queries so that the overall size of all the files is as close to each other as possible. This application

is used by the second application called Dynamic BLAST, a master-worker type application (Afgan et al., 2006) (please see Figure 11 and Figure 12). Dynamic BLAST is a custom-built application intended to maximize the use of small, distributed, readily available resources found in the grid to minimize the time needed to perform BLAST searches (Altschul et al., 1990). It uses established grid protocols for communication and task coordination while the custom scheduler handles task allocation and data transfers. The final application for this chapter is a parallel implementation of matrix-matrix multiplication using Cannon's algorithm (Cannon, 1969) (please see Figure 13 and Figure 14). Figure 9 illustrates relationships between application categories, applications, and corresponding ASL documents.

ASL is a schema and tag driven language enabling a new type of communication between heterogeneous grid resources. As indicated earlier by Figure 3, ASL enables capturing of application developer specific information and subsequent communication between other existing languages. In this context, the information that needs to be communicated is the information that can be compared to information already existing in other, currently available languages. In the grid environment, the goal of such information is to mitigate inherent heterogeneity of underlying resources, and thus the information needs to be capable of representing individual components of communicating systems that are the cause of this heterogeneity.

Figure 9. Relationship between application categories, applications and shown ASL documents



ASL Document Snippets

This section provides selected snippets with essential parts of the application descriptions outlining the capabilities of ASL to capture application information and how dependencies can be drawn between applications, software packages and required libraries.

Figure 10 is an example of a complete ASL *JobInvocationSection* for the *QuerySplit* application described in the previous section. This description starts off by capturing all the basic application run-time information (lines 2 through 9), such as the application invocation method, the minimum amount of memory required for the application to run, as well as the minimum speed of a host CPU. The most significant part of the provided example is to show how to describe an input file for an application. Lines 10 through 39 point out that the given application requires one input file to be provided in plain text (i.e., ASCII) and should be formatted according to the comments available in lines 17 through 33. In particular, lines 18 and 19 indicate that the properties input file must contain two plain text fields with user name and job name, respectively, followed by a pointer to another input file. The format of the contained input file is described as being in FASTA format (FASTA, 2006) whose sample is also provided in the *format* tag. The tags used for capturing necessary information (e.g., text) map favorably to individual components of a job submission interface (e.g., HTML). By analyzing the available tags and provided information, enough details are available to fully automate creation of a job submission interface by a higher level tool with items such as text boxes, radio buttons and drop-down menus with entries predefined in the ASL

Figure 10. ASL document snippet showing Job Invocation Section for the QuerySplit application

```
1.  <asl:JobInvocationRequirements>
2.  <asl:ExecutableName>QuerySplit</asl:ExecutableName>
3.  <asl:ExecutionInstructions>Usage:  perl QuerySplit propertieFile
    </asl:ExecutionInstructions>
4.  <asl:PhysicalMemory>
5.  <asl:LowerBoundedRange>5.0</asl:LowerBoundedRange>
6.  </asl:PhysicalMemory>
7.  <asl:IndividualCPUSpeed>
8.  <asl:LowerBoundedRange>200.0</asl:LowerBoundedRange>
9.  </asl:IndividualCPUSpeed>
10. <asl:InputFiles>
11.   <asl:NumberOfInputFiles>1</asl:NumberOfInputFiles>
12.   <asl:InputFile>
13.     <asl:Name>PropertiesFile</asl:Name>
14.     <asl:Required>true</asl:Required>
15.     <asl:Description>Job properties file whose format is described below.
        </asl:Description>
16.     <asl:FileType>ASCII</asl:FileType>
17.     <asl:Format>
18.       <asl:Text required="1">userName</asl:Text>
19.       <asl:Text required="1">jobName</asl:Text>
20.       <asl:InputFile>
21.           <asl:Name>queryInputFileName</asl:Name>
22.           <asl:Required>true</asl:Required>
23.           <asl:Description>This input file is submitted as part of the properties
24.            input file. A query input file in FASTA format containing at least one
25.            query.</asl:Description>
26.           <asl:FileType>FASTA</asl:FileType>
27.           <asl:Format>
28.            >gi|5524211|gb|AAD44166.1| cytochrome b (Elephas maximus maximus)
29.            LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQ
30.           MSFWGATVITNLFSAIPYIGTNLVEWIWGGFSVDKATLNRFFAFHFIL
31.           PFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLGLL
32.           </asl:Format>
33.       </asl:InputFile>
34.       <asl:Text required="0">totalNumFragments</asl:Text>
35.   </asl:Format>
36.   <asl:SizeLimit><asl:UpperBound>0.1</asl:UpperBound></asl:SizeLimit>
37. </asl:InputFile>
38. </asl:InputFiles>
39. </asl:JobInvocationRequirements>
```

Figure 11. Job Invocation Section of ASL for Dynamic BLAST application showing input file options

```
1.  …
2.  <asl:InputFiles>
3.     <asl:NumberOfInputFiles>1</asl:NumberOfInputFiles>
4.     <asl:InputFile>
5.        <asl:Name>PropertiesFile</asl:Name>
6.        …
7.        <asl:List>
8.           <asl:Radio>
9.              <asl:lText>blastp</asl:lText>
10.             <asl:lText>blastn</asl:lText>
11.             <asl:lText>blastx</asl:lText>
12.             <asl:lText>tblastn</asl:lText>
13.             <asl:lText>tblastx</asl:lText>
14.             <asl:lText>psiblastpn</asl:lText>
15.          </asl:Radio>
16.       </asl:List>
17.       …
18.    </asl:InputFile>
19.    ...
20. </asl:InputFiles>
```

Figure 12. Job Invocation Section of ASL for Dynamic BLAST application showing application, library, and software dependencies

```
<asl:ApplicationsRequired>
   <asl:ApplicationRequired>
      <asl:Name>GridWay</asl:Name>
      <asl:Version>5.0+</asl:Version>
   </asl:ApplicationRequired>
   <asl:ApplicationRequired>
      <asl:Name>QuerySplit</asl:Name>
      <asl:Version>1.0</asl:Version>
   </asl:ApplicationRequired>
</asl:ApplicationsRequired>
<asl:RequiredLibraries>
   <asl:RequiredLibrary>
      <asl:Name>org.ggf.drmaa.*</asl:Name>
   </asl:RequiredLibrary>
</asl:RequiredLibraries>
<asl:SoftwareDependencies>
   <asl:SoftwareDependency>
      <asl:Name>GlobusToolkit</asl:Name>
      <asl:Version>4.0.2+</asl:Version>
   </asl:SoftwareDependency>
   <asl:SoftwareDependency>
      <asl:Name>java</asl:Name>
      <asl:Version>1.5+</asl:Version>
   </asl:SoftwareDependency>
</asl:SoftwareDependencies>
```

Figure 13. Job Invocation Section of ASL for parallel matrix-matrix multiplication application showing and describing application invocation options and arguments

```
<asl:ExecutableName>pmatmul</asl:ExecutableName>
<asl:ExecutionInstructions>Usage: mpirun -np <numCPUs> pmatmul    <N> <P>
<Q> <flag> </numCPUs>
</asl:ExecutionInstructions>
<asl:Options>
   <asl:Option>
      <asl:Arguments>
         <asl:Argument>N</asl:Argument>
         <asl:Description>Matrix size. Works for square matrices
           only.</asl:Description>
      </asl:Arguments>
      <asl:Required>true</asl:Required>
   </asl:Option>
   <asl:Option>
   …
   </asl:Option>
```

Figure 14. Job Invocation Section of ASL for parallel matrix-matrix multiplication application showing ability of ASL to capture memory and network requirements

```
<asl:PhysicalMemory>
   <asl:Formula>5*N^2/sqrt(P)</asl:Formula>
</asl:PhysicalMemory>
<asl:Network>
   <asl:Type>single</asl:Type>
</asl:Network>
<asl:CPUCount>
   <asl:LowerBoundedRange>2.0</asl:LowerBoundedRange>
</asl:CPUCount>
```

document. Through this approach, an application-specific job submission interface can be generated directly from the application description, thus properly directed by an application developer rather than an application deployer.

Figure 11 provides an example of how an application developer can provide all the available values for a selected application invocation option, even within an input file. The XML snippet describes an application's single input file argument options. These are listed in lines 8 through 15, which limit the user's choice to any single value when invoking the application. This is an additional example of how ASL information can be applied in dual context for automated interface generation denoting that this information belongs to a single list and can be organized into a radio button group.

Figure 12 is a continuation of the ASL document for a Dynamic BLAST application. This snippet highlights the application, library, and software dependencies that Dynamic BLAST depends on or requires. As can be seen from the XML, a Dynamic BLAST application depends on being able to correctly invoke two other applications, namely GridWay (Huedo et al., 2004) and *QuerySplit*, whose description was provided in Figure 10. This idea, although simple to comprehend, has a significant potential in terms of application dependency visualization and installation tools. Through the use of this formalized method of declaring direct dependencies, much automation can be achieved. The remainder of the application description, shown in lines 11 through 15, indicates that Dynamic BLAST requires a specified library. Finally, lines 16 through 25 denote other software packages required to run Dynamic BLAST.

Figure 13 also shows the job invocation section of a parallel matrix-matrix multiplication. Line 3 specifies the format of the invocation command with several options. Lines 5 through 11 point out the necessary details about the first argument (e.g., description, whether it is required or optional). The remainder of the argument descriptions is omitted for brevity, but the information provided shows the ability of ASL to structurally and formally capture such information while allowing the user enough freedom to describe each of the arguments at the desired level of detail.

Finally, Figure 14 points out two more interesting points supported by ASL. Lines 4 through 6 deal with network requirements. Because this application is an example of a tightly coupled parallel application, the communication patterns are frequent throughout the algorithm iterations and thus the message passing requires the existence of a fast-speed, local network. Although this requirement can be built into an ASL definition and made a default requirement for all applications of this type, advances in message-passing technologies are enabling the communication to take place across administrative domains (e.g., MPICH-G2 (Karonis et al., 2003)), which would make this a possible hindrance to future applications. To avoid this limitation, the network type tag accepts "single" as its value denoting this application can be executed only on a local network, limited to a single resource. There are other possibilities here, but these options would all require relationships to be made between parts of an ASL document. The final interesting feature found in this part of the sample ASL document is the use of formulas as part of the application description (line 2). This information can be used by a scheduler when the input data is already known to perform not only application-specific but also data-specific scheduling.

Application of ASL Documents

The previous section provided an overview of ASL, its schema, and its sample use for various application categories. The examples demonstrated the capability of ASL to capture needed information as well as provide concrete samples for its creation. The use, possibilities, extensions, and limitation of the language will be explored

over time as its use becomes more widespread and tools emerge. Some examples of possible functionality and tools that can make use of ASL, either by complementing existing technologies or enabling new ones, are outlined in the *Future Trends* section of this chapter. In addition, this section introduces several projects with different goals and how adoption of ASL would improve them.

GridBench (Tsouloupas & Dikaiakos) is a project that focuses on providing a core set of benchmarks that characterize grid resources. Use of such benchmarks allows prediction of performance and scalability of compositions of applications on desired systems. The proposed framework supports collecting, archiving, and publishing collected data. The goal of GridBench is to provide a collection of kernels representative of applications and application categories that can be used to benchmark and characterize components that affect performance of applications and resources, allowing comparisons to be made. Use of ASL in this context to describe applications and subsequently include performance information into an ASL document enables automated sharing of performance associated with individual application and resources. Additional projects similar to GridBench that could use ASL are STAPL (Thomas et al., 2005), Prophesy (Taylor et al., 2000), and application performance predictors such as Downey (1997), Gibbons (1997) and Smith, Foster and Taylor (1998).

GridWay (Montero, Huedo, & Llorente, 2006) aims at supporting the "submit and forget" ideology where the user is abstracted from the grid middleware details by supporting easier and more efficient job execution on the grid. GridWay works on top of the Globus Toolkit and provides a job manager-like interface for its user where a job can be submitted, status checked, and results retrieved. Use of ASL in this context is broad and could include enablement of application-specific information to be presented to the user when selecting among several applications to execute, matching of performance of selected application to available resources, as well as fully automating application invocation parameters and options. Additional examples of grid resource brokers that could benefit from ASL in similar ways as GridWay include Nimrod/G and its parametric job description language (Steen, 2004), Condor-G (Frey, Tannenbaum, Foster, Livny, & Tuecke, 2001) and incorporation of ASL into ClassAds.

The dynamic nature of workflow systems demands the need for Grid services to be automatically generated according to the execution environment and resources available for execution. GridDeploy (Guan, Velusamy, & Bangalore, 2005) is a toolkit for automatically deploying applications as Grid services, and providing the necessary infrastructure for other applications to invoke them. Integration of ASL into GridDeploy would enable more streamlined generation of needed services because ASL provides needed descriptions of applications and corresponding performance on various resources. Use of standardized protocols to combine information available in ASL along with information from JSDL and MDS can enable user and job-oriented service composition by tools such as GridDeploy.

MODELING WIZARDS TO GENERATE ASL

There are several accidental complexities that emerge when using XML as a specification language. For example, an XML file is embodied as linear text, which makes it difficult to capture hierarchical structures that are shared across a document. To represent hierarchy, it is often the case that an XML document contains multiple links to other parts of the document. This reduces the comprehensibility of the XML document because it requires the user to search through the whole file for the desired section link. This becomes unfeasible as the size of the XML file grows. Furthermore, the extensive tags that de-

lineate the sections of an XML document make it very verbose, which introduces another degree for error in terms of the syntactical correctness of the document. In essence, XML serves as an excellent machine readable format due to its formal structure, but has many limitations as a human-centered language. This section introduces the approach we adopted that allows an end-user to specify properties about their grid application from higher level abstractions using a graphical modeling language. These models serve as input to a translator that assists in generating the ASL specification. An overview of domain-specific modeling is presented in the next section, followed by a description of a modeling language that generates wizards for extracting information required in an ASL specification.

Domain-Specific Modeling: Reducing the Effects of Platform Dependency

Model-driven Engineering (MDE) (Schmidt, 2006) represents a design approach that enables description of the essential characteristics of an application in a manner that is decoupled from the details of a specific technology (e.g., dependence on specific middleware or programming language). Domain-Specific Modeling (DSM) (Gray, Tolvanen, Kelly, Gokhale, Neema, & Sprinkle, 2007) is an MDE methodology that generates customized modeling languages and environments from metamodels that define a narrow domain of interest. In this context, a model corresponds to an abstraction whose concrete syntax is rendered in a graphical iconic notation that assists domain experts in constructing a problem description using concepts familiar to them. A model compiler can be associated with a specific modeling language to generate different artifacts (e.g., source code or deployment files) from a model. DSM has been shown to offer specific technical advantages in supporting rapid evolution of computer-based systems when the hardware and software configuration is tightly coupled and must frequently evolve to a new configuration schema (e.g., retooling an automotive factory (Long et al., 1998), or reconfiguring an avionics product-line (Gray, Lin, & Zhang, 2006)).

A benefit of DSM is the ability to describe properties of a system at a high-level of abstraction and in a platform-independent notation, which protects key intellectual assets from technology obsolescence. Initial research in DSM offers a contribution to end-user programming by supporting experts who may not have software development skills (e.g., physicists and avionics engineers) with a visual notation for expressing domain concepts in a specific *problem space*; thus, hiding the accidental complexities of using a general-purpose programming language. In DSM, models and model transformations are first-class entities in the development process and represent the initial point for generation of low-level details in the *solution space*.

The goal of providing new language abstractions appropriate to the specific programming task and user has been a common objective of language researchers for several decades (Floyd, 1979) and shares a common vision with another new approach called intentional programming (Simonyi, Christerson, & Clifford, 2006). Furthermore, the DSM philosophy of narrowly defined modeling languages can be contrasted with larger standardized modeling languages, such as the UML, which are fixed and whose size and complexity (France, Ghosh, & Dinh-Trong, 2006) provide abstractions that may not be needed in every domain, adding to the confusion of domain experts. The concept of "little languages" (Bentley, 1986), or similarly named Domain-specific Languages (DSLs) (Deursen, Klint, & Visser, 2000; Wile, 2004), share the same intent of DSM, but with different concrete syntax (i.e., DSLs are typically textual languages rather than graphical).

The Generic Modeling Environment (GME) (Balasubramanian, Gokhale, Karsai, Sztipanovits, & Neema, 2006) is the metamodeling tool

that we used in our approach to DSM. The next sections present our use of the GME to provide a modeling language that generates wizards for constructing an ASL specification.

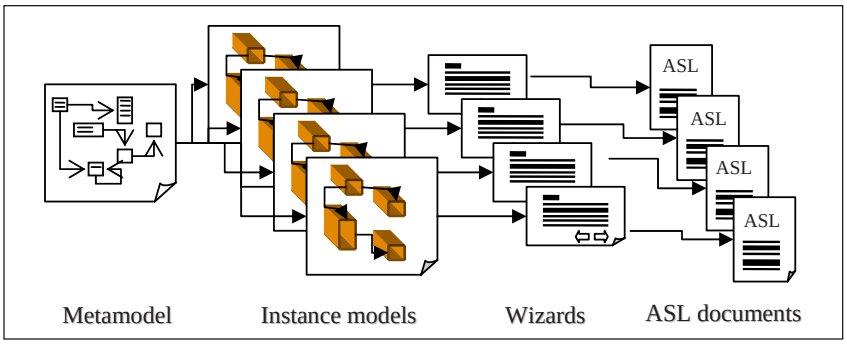
ASL Wizard Metamodel

The use of a higher level graphical modeling language to create ASL specifications consists of constructing the necessary modeling elements describing an application and filling in the parameters that describe the corresponding application feature. This method would help to remove many of the errors in formatting an ASL document to ensure that the document was created correctly (i.e., the application developer would be alleviated of a lot of typing and checking the correctness of XML tags). However, ASL is a language that has a one-to-one mapping between a value and tag pair (i.e., XML tag and the associated value). Because of this, the benefits of adopting a modeling language specifically for ASL are minimized. We realized that a much more significant improvement could be offered by guiding a user in creating an ASL document through wizards. The subsequent challenge became the specific technique to generate the implementation of the wizard.

Depending on the application type, entire element sets available in ASL may not make sense for a particular application. Non-savvy users may find themselves overwhelmed with the number of available options when creating such a document, resulting in slower acceptance or rejection of the ASL. There is a need to ease this transition and provide tools enabling quick and guided creation of such documents. Unfortunately, due to the need to have many possible variations of individual wizards to handle different application categories and variations within these categories, a single wizard would fail to accommodate all the needs. Generating many wizards is prohibitive due to the possibly large number of wizard variations, as well as their temporary lifespan. To accommodate for these variable needs, we have realized a method to generate custom wizards using high-level, domain-specific models. The approach considers the desired components found in ASL and connects them into a meaningful event-driven workflow. The result is a customized wizard that assists in creating a complete ASL document. Figure 15 shows the generation pipeline: a metamodel defines the wizard modeling language, from which specific instance configurations are created; from the instance models, the specific wizard is created (either in HTML, or some other technology); the generated wizards then create the corresponding ASL based on the user response to the wizard interaction.

There are two levels of indirection that are illustrated in Figure 15. The first level is specified in the instance models, which describe the essential elements of each wizard page and the control flow

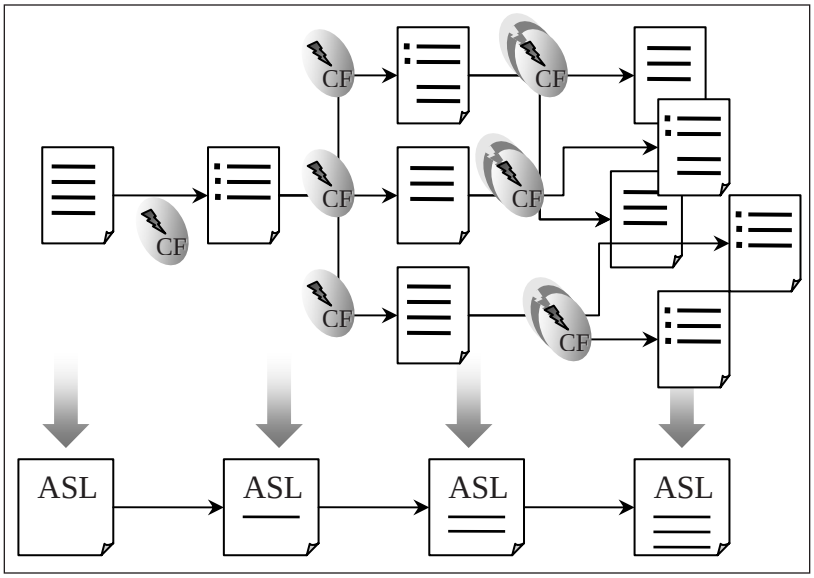
Figure 15. Three-level modeling stack used to generate ASL



(CF) between each page based on user interaction. The instance models must handle creation of objects composing the wizard being generated and must also process all of the possible choices available to adjust the control flow of the wizard. The second level of indirection is built into the generated wizard pages. The wizards are responsible for the actual generation of the ASL document. The control flow captured by the wizards must format the output of an ASL document and store the necessary data in the ASL document. The wizards must also pass any needed information among the wizard pages. To summarize, there are two levels of generation: the generation of wizard pages from the instance models, and the generation of ASL from the wizards; in essence, the process represents a generator of a generator, a unique concept within the area of DSM.

Figure 16 is a diagram showing the possible branching of the wizard pages depending on the user input. The first key observation is the possibility of any single page to be used at multiple levels of the wizard by several parent pages. The second observation is the number of required locations where automatic handling of the control flow and ASL document generation must take place. The model must allow the user accessibility and connectivity to individual pages between different stages of the wizard. This is handled through multiple levels of wizard generation within the model where the same level pages are all viewed on the same aspect within GME. Connections are allowed to be made between compatible objects at appropriate levels through the use of constraints. Connections between individual pages are then made accordingly by the user. The difficulty raised by this approach is the size of the instance model as well as the multiplicity of possible connections resulting in an instance model that is difficult to comprehend. The major obstacle in change adoption and correct execution of the composed wizard comes from the fact that the generator needs to be capable of handling multiple instances of control flow. Each connection that is made at the instance model level must be handled automatically by the generator to provide correct

Figure 16. Wizard flow showing multiplicity of generated wizard pages and control flow (CF) at each junction as the ASL document is being generated



execution flow. Beyond making static connections between wizard pages, additional code must be generated that is model-specific and corresponds to the choices the user is presented at each stage of the wizard.

Generating a Wizard

To highlight the challenges involved in constructing the ASL-driven wizards, this section provides a short scenario to illustrate the multiple levels of control the models must represent. From the perspective of the model user, the elements provided by the modeling language (i.e., the constructs defined in the metamodel) correspond to portions of an ASL document. These are classified as individual components ranging from high-level components (e.g., an entire job invocation section) down to low-level components (e.g., tags such as the application executable name). The iconic representation of these modeling elements, as viewed from a user perspective, are shown in Figure 17.

The end-user is considered to be the domain expert who helps create a model representing the wizard control flow. This is the first difficult task to be handled by the metamodel and subsequently the code generator within the model compiler. The challenge arises from the need for the connections to be made between wizard pages when the initial wizard code is generated, as well as the additional code that needs to be incorporated to handle user choices. This implies that group components that exist in the metamodel (that correspond to equivalent structures in the ASL), such as a drop down menu with many options, must be decomposed into individual pieces and conditional connections made to subsequent pages of the wizard from each of these options. This presents an inherent burden on the model user due to the large number of possible combinations as well as coordination of model layering where complying pages can be reused. The solution is found in the composition of ASL wizard pages. Instead of coupling all of the ASL sections onto a single page, the wizard and corresponding pages can be split into a form. As such, the scope of the ASL is reduced, thus limiting the number of elements, connections and choices the user must consider in the model. The downside to this solution is that equivalent pages within the wizard may belong to a different scope and cannot be reused. Because the division is starting from the most general concepts and progressing into

Figure 17. Available model components describing ASL wizard pages

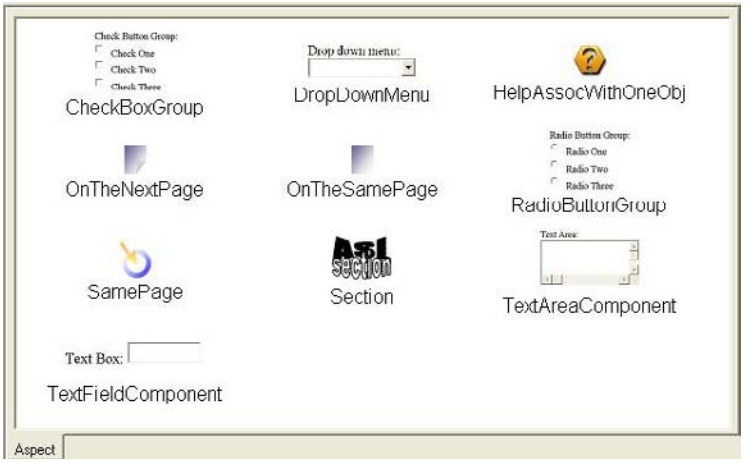


Figure 18. Model view of an ASL document skeleton

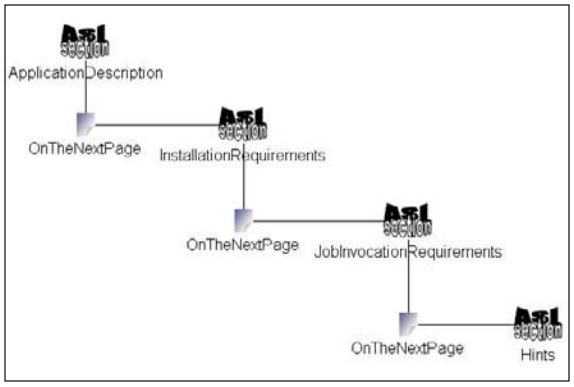
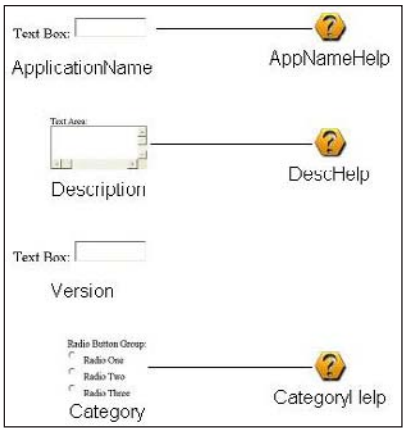


Figure 19. The details of the Application Description element set



more detailed elements of an ASL document, this behavior should be minimized.

Figure 18 shows the skeleton of a typical ASL document with the indication of the necessary connections at this level for the generating wizard. The top-down scoping of the wizard can also be seen in this figure. Because the goal of the project is to simplify ASL generation, individual sections of the model (and thus wizard) directly map to the resulting ASL document. After a given layer of the wizard is fully generated (as shown in Figure 18), the user can provide further details on each individual section. This is accomplished by double-clicking the desired section (i.e., GME model) and opening a new set of model options. This step can be seen in Figure 19 where further details of the *ApplicationDescription* section are provided. The components used at this level map directly to the interface of the generated wizard (except the *Category* section, which is a model requiring individual elements to be provided by accessing the lower layer). This is primarily an example of a two-layer composition where wizard interface components were provided immediately, although additional layers could also be created requiring more pages and connections to be made, as they map to the appropriate page.

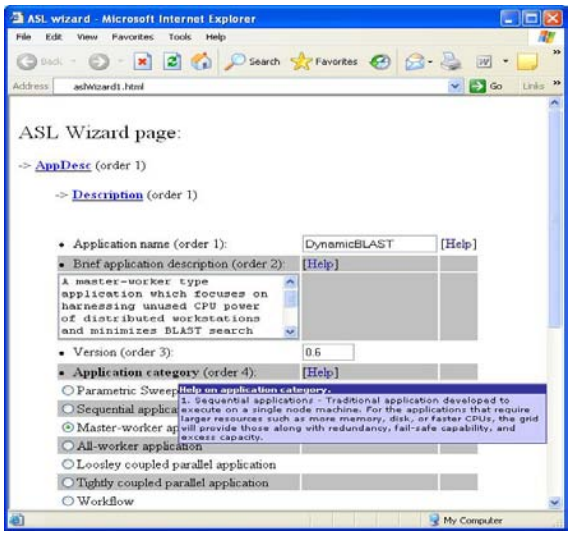
The model compiler that performs the transformation from the model to the generated wizards is presented with a challenge to deliver the requested code among the possible variations. This considerably complicates the code generator within the model compiler. The most difficult part of developing the metamodel and model compiler is the control flow component of the wizard where the connecting pages of the wizard must execute additional connection point code to handle storage and passing of the necessary information between wizard pages. The method of message passing is not defined by the user but rather implied by the generated model. This logic must be customized to the particular wizard and be completely transparent to the end-user. Beyond making the necessary connections at the page level, the generator must be capable of composing necessary code, including page scoping.

To illustrate this idea, a sample workflow can be followed: Upon generation of the wizard, a user instantiates it and provides some data to the wizard. The wizard advances to a subsequent page where the user is asked for more data, this page may branch off into several possible pages,

and so forth. Selection of the subsequent page is determined from a user's input at run-time. As each page may branch into multiple subsequent pages, and even though the paths to individual pages may differ, some of the pages may be equivalent in context. At each step of the wizard, control flow code logic must be provided by the wizard generator to accommodate for this variability and workflow coordination. In addition, the code for data storage that incrementally constructs the resulting ASL document must be generated by the wizard.

We have written a model compiler that generates HTML to represent the wizard pages. This model compiler creates and formats the necessary HTML forms for each wizard page. The compiler also generates the control flow and how the data is passed between each wizard page (i.e., using CGI POST, GET). As the ASL document is being created from multiple levels within the wizard, an additional concern is the correctness of the created document with respect to the generation of proper tags for each ASL element. As discussed in the previous section, ASL contains tags that are lexicographically equivalent. If the tags are placed outside of the correct context, an application interpreting the ASL specification would not be able to disambiguate the ASL. All of these difficulties must be handled in a generic manner within the generated code to accommodate the numerous permutations that may arise. A sample HTML-based wizard can be seen in Figure 20. The generated HTML page is part of the wizard and has been automatically created by the model compiler. The HTML is generated by the *ApplicationDescription* section of the model presented in Figure 18 and Figure 19. Each of the individual components has properties associated with them; thus, the ordering and layering of the wizard is based on the ordering property provided by the user when creating an instance model. Each of the components can also have help information, which appears as a text balloon in the generated wizard.

Figure 20. Sample page of generated wizard to collect initial application information



We have found that describing an application from a higher level of abstraction reduces the learning curve for a new technology such as ASL, but it has drawbacks in that it requires a new tool to be added in the repertoire of an application developer. The only related tool that accomplishes a similar idea of composing wizards from a higher level specification targets the .Net platform and generates MS Windows forms wizards (Actipro, 2006). However, integration of interoperable tools that share a single interface can advance the productivity of a software developer. An example is the Grid Automation and Generative Environment (GAUGE) (Hernandez et al., 2006), which also reduces accidental complexities of grid development through adoption of DSM. Because the development process within GAUGE takes place within GME, its extension to include concepts introduced by ASL represents a possible integration opportunity. By incorporating and extending an intelligent agent used for the application development process, data can also be extracted automatically to populate sections of the corresponding ASL document, thus reducing the

work required by the application developer. The integration with GAUGE represents future work that can ease the path of ASL adoption.

FUTURE TRENDS

As the general adoption of the grid moves from the research labs and campus infrastructures into mainstream industry (with the goal of offering high-levels of Quality of Service), many components (in terms of pervasive availability, secure use of systems, and seamless access) must still be provided. Many of the individual components and additional functionalities are being addressed by many researchers. The following is a list of research issues and further advancements that remain open in the context of this chapter as it fits into the larger picture of the grid:

- The initial schema and corresponding ASL documents have been constructed to ensure the language is capable of representing applications. The ASL schema should also be able to define individual components that comprise an application. The differences between two applications should also be discernable from the respective ASL specifications. Many more applications need to be described with the goal of adjusting the ASL schema to address new application categories.
- Publicizing the benefits of the ASL is necessary to promote adoption by the wider grid community. The current focus considers the standardization of the ASL within the Open Grid Forum. This is following a standardization path similar to JSDL.
- Additional tools are required to promote generation of ASL, such as the modeling approach described in this chapter.
- Tools that make use of ASL are of significant importance. Many tools can be adapted to make use of ASL documents but standardization is required before these actions can take place. Examples of such tools include the following:
  - *Application information services* (AIS), which represent a suite of services where registration, discovery, and monitoring of an application could take place at the level of a VO. Through a set of interfaces, the users would be able to discover services and applications, and obtain help documentation.
  - *Automated HTML application help generator*, which would be an application to interpret ASL documents and generate application-related information. This can be as simple as a static Web page being generated as the default help documentation or a more advanced on-demand search-and-match custom page generation based on user queries of problem identification to include the latest available information.
  - *GridRPM*, would be an application to take advantage of installation and dependency information available through the mesh of ASL documents (as available in AIS) and automatically construct descriptor files before proceeding with automatic application installation and deployment.
  - *Grid scheduler applications* could utilize the ASL information to perform application-specific scheduling without the need to keep historical application execution data.
  - *Accounting and software license accountability* is an area of the grid currently in its initial stages but will advance quickly in response to the high demand of grid services from industry. This area provides a vast number of options where relevant information

can be stored into ASL documents and later used through automated billing and license verification.

CONCLUSION

This chapter provides an overview of developing intensive software applications in a grid computing environment along with the taxonomy of grid users. For each user category the specific responsibilities, roles, and requirements were described. A sample application scenario was provided to illustrate the interaction required among the different user categories and the grid middleware. The functionality provided by the grid middleware to accomplish the successful deployment of the sample scenario was presented and difficulties with the current solutions were also outlined. To further illustrate the complexity involved in deploying grid applications, the grid applications were categorized into seven major categories. For each deployment category the various steps involved in the deployment process were discussed. It was noted that there is no support currently available for the application developer category to describe the requirements and capabilities of an application. In order to accommodate this need, a new language was presented–the Application Specification Language (ASL)–that supports the description of applications in a grid environment. Examples outlining the capabilities of ASL were provided using the sample application scenario. Finally, the chapter described a supporting modeling language for generating ASL documents from a higher level specification, which minimizes the effort of an application developer and deployer.

ASL enables the application developer to describe the application requirements and capabilities. Using domain-specific modeling for the creation of ASL documents reduces some of the accidental complexities encountered in the grid application development and deployment process. Metamodeling techniques could be further extended to provide additional tools for applying information available in ASL to other areas of grid computing (e.g., automated user interface generation for job registration, submission, and monitoring). The work presented here takes a bottom-up approach to solving some of the problems in information availability and standardization between grid entities (e.g., applications, users, and resources) in the heterogeneous environment. Through the adoption of technology presented here, individual pieces within a complex grid system can communicate through a common language and at the appropriate level of granularity permitting context-controlled information delivery in a range or formats. Overall adoption of the domain-specific modeling ideas into the area of grid computing as an effective tool to leverage some of the grid-inherent difficulties may result in shorter application development and deployment times by reducing complexity and expertise required.

ACKNOWLEDGMENT

This was supported in part by an NSF CAREER award (CCF-0643725).

REFERENCES

Aalst, W. V. D., & Hee, K. V. (2002). *Workflow management: Models, methods, and systems.* Cambridge, MA: The MIT Press.

Actipro. (2006). *Actipro wizard - Wizard Generation.NET framework windows forms control.* Retrieved March 23, 2008, from http://www.actiprosoftware.com/Products/DotNet/WindowsForms/Wizard/Default.aspx

Afgan, E. (2004). Role of the resource broker in the grid. In Proceedings of the *42nd Annual ACM Southeast Conference*, (pp. 299-300). Huntsville, AL: ACM Press.

Afgan, E., & Bangalore, P. (2007). Application specification language (ASL)–a language for describing applications in grid computing. In *Proceedings of the 4th International Conference on Grid Services Engineering and Management - GSEM 2007,* Leipzig, Germany.

Afgan, E., Sathyanarayana, P., & Bangalore, P. (2006). Dynamic task distribution in the grid for BLAST. In *Proceedings of Granular Computing 2006,* (pp. 554-557). Atlanta, GA: IEEE.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., et al. (2001). Data management and transfer in high-performance computational grid environments. *Parallel Computing, 28*(5), 749-771.

Allen, G., Davis, K., Goodale, T., Hutanu, A., Kaiser, H., Kielmann, T., et al. (2005). The grid application toolkit: Toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE, 93*(3), 534-550.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Mol Biol, 215*(3), 403-410.

Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., et al. (2005). *Job submission description language (JSDL) specification, version 1.0* (Tech. Rep. No. GFD-R.056). Global Grid Forum (GGF).

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., et al. (1999). Toward a common component architecture for high-performance scientific computing. In *Proceedings of the High-Performance Distributed Computing Conference (HPDC),* (pp. 115-124). Redondo Beach, CA: IEEE Computer Society.

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., & Neema, S. (2006). Developing applications using model-driven design environments. *IEEE Computer, 39*(2), 33-40.

Baru, C., Moore, R., Rajasekar, A., & Wan, M. (1998). The SDSC storage resource broker. In *Proceedings of CASCON '98,* (pp.). Toronto, Canada: IBM Press.

Bentley, J. (1986). Little languages. *Communications of the ACM, 29*(8), 711-721.

Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., et al. (2001). The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications, 15*(4), 327-344.

Berman, F., Fox, G., & Hey, T. (2003a). The grid: Past, present, future. In F. Berman, G. Fox, & T. Hey (Eds.), *Grid computing--making the global infrastructure a reality* (pp. 9-51). Hoboken, NJ: John Wiley & Sons.

Berman, F., Hey, A., & Fox, G. (Eds.). (2003b). *Grid computing: Making the global infrastructure a reality.* New York: John Wiley & Sons.

Buyya, R., Abramson, D., & Giddy, J. (2000). Nimrod-G: An architecture for a resource management and scheduling in a global computational grid. In *Proceedings of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000),* (pp. 283-289). Beijing, China: IEEE CS Press.

Cannon, L. (1969). *A cellular computer to implement the Kalman Filter Algorithm*. Bozeman, MT: Montana State University.

Casanova, H., Obertelli, G., Berman, F., & Wolski, R. (2000). The AppLeS parameter sweep template: User-level middleware for the grid. *Supercomputing 2000.* Dallas, TX: IEEE Computer Society.

Chase, N. (2005). *Grid-enable an existing Java technology application*: IBM. Retrieved March 23, 2008, from http://www.ibm.com/developerworks/edu/gr-dw-gr-javagrid-i.html

Czajkowski, K., Fitzgerald, S., Foster, I., & Kesselman, C. (2001). Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC),* (pp. 181-195). Los Alamitos, CA: IEEE Computer Society.

Czajkowski, K., Foster, I., Kesselman, C., Martin, S., Smith, W., & Tuecke, S. (1998). A resource management architecture for metacomputing systems. In *Proceedings of the IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing,* (pp. 62-82). Springer-Verlag.

Deursen, A. V., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices, 35*(6), 26-36.

Downey, A. (1997). Predicting queue times on space-sharing parallel computers. In *Proceedings of the International Parallel Processing Symposium (IPPS '97),* (pp. 209-218). Geneva, Switzerland.

FASTA. (2006, December 4). *FASTA format*. Retrieved March 23, 2008, from http://en.wikipedia.org/wiki/Fasta

Floyd, R. (1979). The paradigms of programming. *Communications of the ACM, 22*(8), 455-460.

Forum, M. P. I. (1998). *MPI message-passing interface standard, version 2.0*. Retrieved March 23, 2008, from http://www.mpi-forum.org/docs/docs.html

Foster, I., & Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications, 11*(2), 115-128.

Foster, I., & Kesselman, C. (1999). The Globus toolkit. In I. Foster & C. Kesselman (Eds.), *The grid: Blueprint for a new computing infrastructure* (pp. 259-278). San Francisco: Morgan Kaufmann.

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002a). *The physiology of the grid: An open grid services architecture for distributed systems integration*. Global Grid Forum, Open Grid Service Infrastructure Working Group.

Foster, I., Kesselman, C., Tsudik, G., & Tuecke, S. (1998). A security architecture for computational grids. In *Proceedings of the 5th ACM Conference on Computer and Communication Security Conference,* (pp. 83-92). San Francisco: ACM Press.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid. *Lecture Notes in Computer Science, 2150,* 1-28.

Foster, I., Voeckler, J., Wilde, M., & Zhao, Y. (2002b). Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM'02),* (pp. 37-46). Edinburgh, Scotland.

France, R., Ghosh, S., & Dinh-Trong, T. (2006). Model-driven development using UML 2: Promises and pitfalls. *IEEE Computer (Special Issue on Model-Driven Engineering, 39*(2), 41-48.

Frey, J., Tannenbaum, T., Foster, I., Livny, M., & Tuecke, S. (2001). Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the IEEE Symposium on High Performance Distributed Computing (HPDC10),* (pp. 9). San Francisco, CA.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Gannon, D., Fox, G., Pierce, M., Plale, B., Laszewski, G. V., Severance, C., et al. (2003). *Grid portals: A scientist's access point for grid services.* Global Grid Forum (GGF).

Gemmill, J., & Bangalore, P. (2005). *UABGrid--a campus-wide distributed computational infrastructure*. Birmingham, AL: UAB. Retrieved March 23, 2008, from http://uabgrid.uab.edu/

Gibbons, R. (1997). A historical application profiler for use by parallel schedulers. *Lecture Notes in Computer Science, 1291*, 58-77.

Gottschalk, K., Graham, S., Kreger, H., & Snell, J. (2002). Introduction to Web services architecture. *IBM Systems Journal, 41*(2), 170-178.

Gray, J., Lin, Y., & Zhang, J. (2006). Automating change evolution in model-driven engineering. *IEEE Computer, 39*(2), 51-58.

Gray, J., Tolvanen, J. P., Kelly, S., Gokhale, A., Neema, S., & Sprinkle, J. (2007). Domain-specific modeling. In P. Fishwick (Ed.), *Handbook on dynamic system modeling* (1st ed.). CRC Press.

Grid, O. S. (2007, Januray 19). *Open science grid*. Retrieved March 23, 2008, from http://www.opensciencegrid.org/

GriPhyN. (2006, July 30). *GriPhyN--grid physics network*. Retrieved March 23, 2008, from http://www.griphyn.org/

Guan, Z., Velusamy, V., & Bangalore, P. (2005). GridDeploy: A toolkit for deploying applications as grid services.In *Proceedings of the International Conference on Information Technology Coding and Computing*, Las Vegas, NV.

Hernandez, F. A., Bangalore, P., Gray, J., Guan, Z., & Reilly, K. (2006). GAUGE: Grid automation and generative environment using domain engineering and domain modeling for drafting applications for the grid. *Concurrency and Computation: Practice & Experience, 18*(10), 1293-1316.

Huedo, E., Montero, R. S., & Llorente, I. M. (2004). A framework for adaptive execution on grids. *Journal of Software: Practice and Experience, 34*(7), 631-651.

IVDGL. (2006, July 30). *International virtual data grid laboratory*. Retrieved March 23, 2008, from http://www.ivdgl.org/

JSWG. (2007). *Job submission description language WG (JSDL-WG)*. Retrieved March 23, 2008, from https://forge.gridforum.org/projects/jsdl-wg/

Karonis, N., Toonen, B., & Foster, I. (2003). MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC), 63*(5), 551-563.

Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing: Design and analysis of algorithms*. Redwood City, CA: Benjamin/Cummings.

Laszewski, G. V., Foster, I., Gawor, J., & Lane, P. (2001). A Java commodity grid kit. *Concurrency and Computation: Practice and Experience, 13*(8-9), 643-662.

Letondal, C. (2000). A Web interface generator for molecular biology programs in Unix. *Bioinformatics, 17*(1), 73-82.

Litzkow, M., Livny, M., & Mutka, M. (1988). Condor--a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, (pp. 104-111).

Massie, M. L., Chun, B. N., & Culler, D. E. (2004). The Ganglia distributed monitoring system: Design, Implementation, and Experience. *Parallel Computing, 30*(7), 817-840.

McConnell, S. (1996). *Rapid development* (1st ed.). Redmond, WA: Microsoft Press.

Momjian, B. (2000). *PostgreSQL: Introduction and concepts* (1st ed.): Pearson Education.

Montero, R. S., Huedo, E., & Llorente, I. M. (2006). Grid scheduling infrastructures based on the GridWay meta-scheduler. *IEEE Techni-*

cal Committee on Scalable Computing (TCSC) Newsletter, 8(2).

NEES. (2006, November 6). *Network for earthquake engineering simulation*. Retrieved March 23, 2008, from http://it.nees.org/

Novotny, J., Russell, M., & Wehrens, O. (2004). GridSphere: A portal framework for building collaborations. *Concurrency and Computation: Practice & Experience, 16*(5), 503-513.

OGF. (2007). *Open grid forum*. Retrieved March 23, 2008, from http://www.ogf.org

PPDG. (2006, July). *Particle physics data grid*. Retrieved March 23, 2008, from http://www.ppdg.net/

Schmidt, D. (2006). Model-driven engineering. *IEEE Computer, 39*(2), 25-32.

Simonyi, C., Christerson, M., & Clifford, S. (2006). Intentional software. *Object oriented programming systems languages and applications (OOPSLA),* (pp. 451-464). Portland, OR: ACM Press.

Smith, W., Foster, I., & Taylor, V. (1998). Predicting application run times using historical information. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing,* (pp. 122-142). Springer-Verlag.

Solomon, M. (2004, May). *The ClassAd language reference manual.* Retrieved March 23, 2008, from http://www.cs.wisc.edu/condor/classad/refman/

Sotomayor, B., & Childers, L. (2005). *Globus toolkit: Programming Java services* (1st ed.). Morgan Kaufmann.

Steen, M. V. (2004). *Nimrod-G resource broker for service-oriented grid computing.* Retrieved March 23, 2008, from http://dsonline.computer.org/0107/departments/res0107_print.htm

Tanenbaum, A. S., & Steen, M. V. (2002). *Distributed systems: Principles and paradigms.* Prentice Hall.

Taylor, V., Wu, X., Geisler, J., Li, X., Lan, Z., Stevens, R., et al. (2000). Prophesy: An infrastructure for analyzing and modeling the performance of parallel and distributed applications. In *Proceedings of the High Performance Distributed Computing (HPDC) 2000,* Pittsburgh, PA, (pp. 302-303).

Thomas, M., Mock, S., Dahan, M., Mueller, K., Sutton, D., & Boisseau, J. (2001). The GridPort toolkit: A system for building grid portals. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC),* (pp. 216-227). San Francisco: IEEE.

Thomas, N., Tanase, G., Tkachyshyn, O., Perdue, J., Amato, N., & Rauchwerger, L. (2005). A framework for adaptive algorithm selection in STAPL. In *Proceedings of the ACM SIGPLAN 2005 Symposium on Principles and Practices of Parallel Programming (PPoPP),* Chicago, IL.

Tsouloupas, G., & Dikaiakos, M. (2003). GridBench: A tool for benchmarking grids. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003),* Phoenix, AZ, (pp. 60-67).

Ulrich, W. M. (2002). *Legacy systems: Transformation strategies* (1st ed.). Prentice Hall PTR.

Venugopal, S., Buyya, R., & Winton, L. (2005). A grid service broker for scheduling e-science applications on global data grids. *Journal of Concurrency and Computation: Practice and Experience, 18*(6), 685-699.

Wall, L., Christiansen, T., & Orwant, J. (2000). *Programming Perl* (3rd ed., Vol. 2000). O'Reilly Media.

Wile, D. (2004). Lessons learned from real DSL experiments. *Science of Computer Programming, 51*(3), 265-290.

ADDITIONAL READING

Afgan, E., & Purushotham, B. (2007). Computation cost in grid computing environments. In *Proceedings of the First International Workshop on the Economics of Software and Computation in Conjunction with International Conference on Software Engineering (ICSE) 2007,* Minneapolis, MN, (pp. 9-13).

Batory, D. (2006). Multiple models in model-driven engineering, product lines, and metaprogramming. *IBM Systems Journal, 45*(3), 451-461.

Brooks, F. P. (1987). No silver bullet-essence and accidents of software. *IEEE Computer, 20*(4), 10-19.

Buyya, R., & Murshed, M. (2002). GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14*(13-15), 1175-1220.

Cao, F., Bryant, B. R., Raje, R. R., Auguston, M., Olson, A. M., & Burt, C. C. (2005). A component assembly approach based on aspect-oriented generative domain modeling. *Electronic Notes in Theoretical Computer Science, 114,* 119-136.

Cunha, J. C., & Rana, O. F. (2005). *Grid computing: Software environments and tools* (1st ed.). Springer-Verlag.

Foster, I., & Kesselman, C. (1998). *The grid: Blueprint for a new computing infrastructure* (1st ed.). Morgan Kaufmann.

Hernandez, F., Bangalore, P., Gray, J., & Reilly, K. (2004). A graphical modeling environment for the generation of workflows for the Globus toolkit. In *Proceedings of the Workshop on Component Models and Systems for Grid Applications, 18th Annual ACM International Conference on Supercomputing,* Saint-Malo, France.

Hey, A. J. G., Papay, J., & Surridge, M. (2005). The role of performance engineering techniques in the context of the Grid. *Concurrency and Computation: Practice & Experience, 17*(2-4), 297-316.

Horst, J., Messina, E., Kramer, T., & Huang, H.-M. (1997). Precise definition of software component specifications. In *Proceedings of the 7th Symposium on Computer-Aided Control System Design (CACSD '97),* (pp. 145-150). Gent, Belgium.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., et al. (1997). Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP),* (pp. 220-242). Jyvoskyla, Finland: Springer-Verlag.

Kra, D. (2004). *Six strategies for grid application enablement*: *IBM*. Retrieved March 23, 2008, from http://www.ibm.com/developerworks/grid/library/gr-enable/

Kurtev, I., Bézivin, J., Jouault, F., & Valduriez, P. (2006). Model-based DSL frameworks. In *Proceedings of the Companion of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA),* Portland, OR, (pp. 602-616).

Nabrzyski, J., Schopf, J. M., & Weglarz, J. (2003). *Grid resource management: State of the art and future trends. International series in operations research & management science* (1st ed.). Springer-Verlag.

Silva, V. (2005). *Grid computing for developers (Programming Series)* (1st ed.). Charles River Media.

Sodhi, S., & Subhlok, J. (2005). Automatic construction and evaluation of performance skeletons. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS '05),* Denver, CO, (p. 10).

Song, H., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., et al. (2000). The MicroGrid: A scientific tool for modeling computational grids. In *Proceedings of the IEEE Supercomputing (SC 2000),* Dallas, TX.

This work was previously published in Designing Software-Intensive Systems: Methods and Principles, edited by P. Tiako, pp. 402-438, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.25
Performance Analysis of a Web Server

Jijun Lu
University of Connecticut, USA

Swapna S. Gokhale
University of Connecticut, USA

ABSTRACT

With the rapid development and widespread use of the Internet, Web servers have become a dominant source of information and services. The use of Web servers in business and critical application domains imposes stringent performance requirements on them. These performance requirements cast a direct influence on the choice of the configuration options of the hardware and the software infrastructure on which a Web server is deployed. In addition to the selection of configuration options, for a given level of load and a particular hardware and software configuration, it is necessary to estimate the performance of a Web server prior to deployment.

INTRODUCTION AND MOTIVATION

The World Wide Web (WWW) has experienced an exponential growth in the last 10 years and today Web servers are important sources of information and services. Web servers, which are typically based on the HTTP protocol running over TCP/IP, are expected to serve millions of transaction requests per day with acceptable performance, which may be defined in terms of transaction throughput and latency experienced by the users (Van der Mei, Hariharan, & Reeser, 2001). The stringent performance requirements imposed on Web servers have a direct influence on the configuration options of the hardware and software infrastructure used for deployment. Hardware configuration options may include the capacity and the number of processors and caching strategies. Software configuration options may include the number of server threads/processes to serve client requests, the buffer size, and the scheduling discipline. Prior to deployment, for a given level of load, it is necessary to determine the hardware and software configuration options that would provide acceptable server performance.

One of the ways of estimating the performance of a Web server is by conducting actual measurements. While the measurement-based approach may be viable to estimate the performance for a given set of configuration options, it is cumbersome and expensive for "predictive" or "what-if" analysis and for an exploration of a set of alternative configurations. Model-based analysis, which consists of capturing the relevant aspects of a Web server into an appropriate model, validating the model and then using the validated model to predict the performance for different settings is an attractive alternative to the measurement-based approach.

Web servers receive and process a continuous stream of requests. As a result, a vast majority of their time is spent waiting for I/O operations to complete, making them particularly apt to fall under the category of I/O intensive applications (Ling, Mullen, & Lin, 2000). The performance of such I/O intensive applications (responsiveness, scalability, and throughput) can be improved dramatically if they are provided with the capability to process multiple requests concurrently. Thus, modern Web servers invariably process multiple requests concurrently to enhance their performance and to fulfill their workload demands. Considering the concurrent processing capability, we propose the use of a multiserver *M/G/m* queue to model a Web server with an I/O intensive workload. The performance metric of interest is the response time of a client request. Since there is no known analytically or computationally tractable method to derive an exact solution for the response time of the *M/G/m* queue, we use an approximation proposed by Sakasegawa (1977). We validate the model for deterministic and heavy-tailed workloads using experimentation. Our results indicate that the *M/G/m* queue provides a reasonable estimate of the response time for moderately high traffic intensity. The conceptual simplicity of the model combined with the fact that it needs the estimation of very few parameters makes it easy to apply.

The balance of the article is organized as follows: First, we present the performance model of a Web server. We then discuss the workload characteristics used for the experimental validation of the model, followed by a description of the experimental infrastructure used for validation. Subsequently, we present and discuss the experimental results. Research related to the present work is summarized next. Finally, we offer concluding remarks and directions for future research.

PERFORMANCE MODEL

We describe the performance model of a Web server in this section. Towards this end, we first provide an overview of the software architecture of a Web server. Subsequently, we discuss the rationale for modeling a Web server using an *M/G/m* queue and present an analytical expression to compute the approximate response time.

Web Server Software Architecture

Modern Web servers implement concurrent processing capability using a thread-based, a process-based, or a hybrid approach (Menasce, 2003). An example of a thread-based server is the Microsoft IIS server (Microsoft Corporation, n.d.), a process-based server is the Apache HTTP server 1.3, and a hybrid server is the Apache HTTP server 2.0 (Apache Software Foundation, n.d.).

In both the thread-based and process-based architectures, to avoid the overheads of forking a process/thread for every request, the Web server can fork a pool of processes/threads at start-up. If all these threads/processes are busy, either additional threads/processes can be forked or the request waits in a queue. In the former case, the size of the thread/process pool changes dynamically, whereas in the latter case the size of the thread/process pool is fixed and a new request

waits in the queue if all the threads/processes are busy.

Queuing Model

We assume that the Web server consists of a static thread/process pool, with the number of threads/processes in the pool or the pool size, denoted *m*. Requests arrive at the server according to a Poisson distribution and the request service time is exponentially distributed. In most Web servers, the capacity of the queue to hold requests when all the threads/processes are busy is typically very large to ensure low probability of denying a request. Thus, for the purpose of modeling we assume the queue size to be infinite. The queuing model that coincides with these characteristics of a Web server, capable of processing requests

Figure 1. M/G/m queuing model

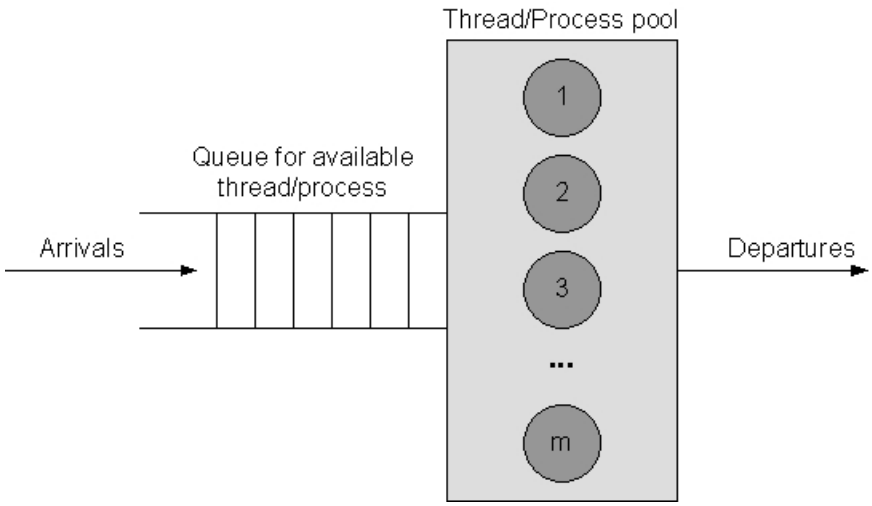


Table 1. Parameters of the M/G/m queuing model

Symbol	Meaning
R	Response time
m	Number of servers
λ	Arrival rate
μ	Service rate
c_s	Coefficient of variance of service time (ratio of standard deviation to the mean)
ρ	Traffic intensity $\rho = \lambda/(m\mu)$

concurrently is an *M/G/m* queue. We note that since the model is based on the existence of concurrent processing capabilities, and not on the specific implementation of concurrency, it is general and equally applicable to a thread-based, a process-based or hybrid architecture.

Figure 1 shows a pictorial depiction of the *M/G/m* queue. The arrival process is Poisson with rate and there are *m* parallel servers serving the requests. The service times of the requests are independent and identically distributed random variables, with a general distribution that has a finite mean μ^{-1} and a finite coefficient of variance c_s (the ratio of standard deviation to the mean). Specifically, for deterministic service times, c_s = 0; for exponential service times, c_s = 1; and for heavy-tailed service times, $c_s \to \infty$ (Lipsky, 1992). Let $\rho = \lambda/(m\mu)$ denote the traffic intensity, also known as utilization factor (Lipsky, 1992). For a stable system $\rho < 1$ and the queuing discipline is first-come-first-served (FCFS). Table 1 summarizes the parameters of the *M/G/m* model.

The performance metric of interest is the expected or the average response time of a client request denoted *R*. Except for certain special cases, there is no known analytically or computationally tractable method to derive an exact solution for the response time of the *M/G/m* queue. Several approximations have been proposed (Hokstad, 1978; Kimura, 1983; Sakasegawa, 1977; Yao, 1985) and among these we consider the one by Sakasegawa (Sakasegawa, 1977) since it requires very few parameters and involves straightforward computations. Using this approximation, the mean number of requests in the queue for the *M/G/m* model is given by:

$$L_q \approx \frac{(1+c_s^{\,2})}{2}\frac{\rho^{\sqrt{2(m+1)}}}{1-\rho} \qquad (1)$$

Equation (1) indicates that the queue length increases as ρ increases. Further, for a service time distribution with high variability (where c_s is high), such as in the heavy-tailed distribution (Crovella, Taqqu, & Bestavros, 1998), the queue length will increase very rapidly.

Using Equation (1), the mean response time for the *M/G/m* queue as per Little's law (Kleinrock, 1976) is:

$$R_{M/G/m} \approx 1/\mu + \frac{L_q}{\lambda} = 1/\mu + \frac{(1+c_s^{\,2})}{2\lambda}\frac{\rho^{\sqrt{2(m+1)}}}{1-\rho} \qquad (2)$$

Since a single server queue is commonly used to model a Web server (Cao, Anderson, Nyberg, & Kihl, 2003; Nossenson & Attiya, 2004; Squillante, Yao, & Zhang, 1999), we compare the mean response time of a multiserver *M/G/m* queue to the response time of a single-server *M/G/1* queue, which is given by:

$$R_{M/G/1} = 1/\mu + \frac{(1+c_s^{\,2})}{2\lambda}\frac{\rho'^{2}}{1-\rho'} \qquad (3)$$

where $\rho' = 1/\mu = m\rho$. Notice that $R_{M/G/1} > R_{M/G/M}$ holds for $\forall m \geq 2$.

WORKLOAD CHARACTERISTICS

We consider service scenarios where each request is for a single static file and the server responds to the client with the file. Since the file size is demonstrated to be heavy-tailed (Crovella et al., 1998), we consider two types of workloads, namely, deterministic and heavy-tailed, which are described next.

Deterministic Workload

For a deterministic workload, all the requests are for the same file size in a single experiment. We consider four deterministic workloads with file

sizes of 64 KB, 128 KB, 256 KB, and 512 KB. The smallest size considered is 64 KB due to the measurement granularity, which is 10 *msec* for our test bed (described in the next section). We note that even for a deterministic workload, in which the file size in every request is the same, the service time on an experimental test bed/real system will fluctuate (Lipsky, 1992), implying that $c_s \neq 0$ in Equation (1). Since the magnitude of these fluctuations is expected to be small compared to the actual service time, we use the term *deterministic workload* to indicate a relatively stable workload compared to the heavy-tailed one described next.

Heavy-Tailed Workload

In this section we discuss how the heavy-tailed workload was generated based on the Pareto distribution, which has been widely used to model such characteristics in a variety of phenomena including inter-arrival times (Paxson & Floyd, 1995), burst sizes (Charsinski, 2000), topological properties (Faloutsos, Faloutsos, & Faloutsos, 1999), workload properties (Crovella et al., 1998), and error rates (Goseva-Postojanova, Mazimdar, & Singh, 2004).

The Pareto distribution is characterized by a shape parameter α and a scale parameter k. The probability density function (pdf) of the Pareto distribution is given by:

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1} \quad (4)$$

where α, $k > 0$ and $x > k$. The cumulative distribution function (CDF) of the Pareto distribution is given by:

$$F(x) = 1 - k^{\alpha} x^{-\alpha} \quad (5)$$

When $\alpha > 1$, the Pareto distribution has a finite mean which is given by:

$$E[X] = \frac{\alpha k}{\alpha - 1} \quad (6)$$

A series of numbers following the Pareto distribution can be generated by computing the inverse of the CDF presented in Equation (5) as below (Jin & Bestavros, 2005):

$$x = F^{-1}(u) = k(1-u)^{-1/\alpha} \quad (7)$$

In Equation (7), if α and k are known, then for random variates u_i's generated from the uniform distribution $U(0,1)$, $\{x_i\}$'s following the Pareto distribution can be generated. Note here $x_i > k$ for every i. It is expected that most $\{x_i\}$'s will have values close to k, with a small number being extraordinary high. When the shape parameter α of the Pareto distribution is fixed, the scale parameter k for a given mean $E[X]$ can be obtained as follows:

$$k = E[X]\,(1 - 1/\alpha) \quad (8)$$

We set the value of the shape parameter α to 1.36, based on prior research (Crovella, Taqqu & Bestavros, 1998). Four different mean file sizes, namely, 120 KB, 150 KB, 180 KB, and 200 KB were considered, and for each mean file size, using the preselected shape parameter α, the scale parameter k was obtained using Equation (8). Using the (α, k) pair, the CDF F_{S_i} of file size S_i is obtained. The probability of each file size is then computed as:

$$P_{S_i} = F_{S_i} - F_{S_{i-1}} \quad (9)$$

The individual file sizes considered are 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, and 2 MB and the distributions of these file sizes in the four workloads are reported in Table 2.

EXPERIMENTAL INFRASTRUCTURE

In this section, we describe the experimental infrastructure used for model validation. We also

Table 2. File size distribution (P_{S_i}) in heavy-tailed workload

File size (S_i)	Mean file size $\bar{s}_i$			
	120KB	150KB	180KB	200KB
64 KB	0.6099	0.4776	0.3305	0.2274
128 KB	0.2381	0.3189	0.4086	0.4716
256 KB	0.0928	0.1242	0.1592	0.1837
512 KB	0.0361	0.0484	0.0620	0.0716
1 MB	0.0141	0.0189	0.0242	0.0279
2 MB	0.0055	0.0073	0.0094	0.0109

Table 3. System configuration parameters

Machine	Hardware	Software
Server	Dell OptiPlex GX260 (Intel Pentium 4 processor at 2.4GHz, 1GB of RAM, 40GB hard driver and Intel PRO 1000 MT network adapter)	Microsoft Windows XP Professional SP2, Microsoft IIS 5.1
Client	IBM ThinkPad T40 (Intel Pentium-M processor at 1.5GHz, 1GB of RAM, 40GB hard driver and Intel PRO 100 VE network adaptor)	Microsoft Windows XP Professional SP2

describe the process used to estimate the mean service time for each file size.

Test Bed Description

The experimental infrastructure comprises of a single server and a single client, with configurations summarized in Table 3. Both computers are connected via a 100M Ethernet across a LAN.

In a multithreaded server, as the size of the thread pool increases, each thread may experience some performance degradation. However, when the thread pool size is below a threshold, this degradation is negligible (Xu & Bode, 2004). As a rule of thumb, to ensure negligible degradation, the size of the thread pool should be twice the number of CPUs on the host machine (Ling et al., 2000; Richter, 2000). Thus, for the single processor server in our test bed, the size of the thread pool is set to 2. The configurations of the Web server used in the experiments are summarized in Table 4. Since the maximum queue length used is very large as compared to the number of threads, the assumption of infinite queue length needed to apply the *M/G/m* model is reasonable.

Service Time Estimation

Table 4. Web server configurations

Parameter	Value
CPU	1
Thread	2
Web caching	No
Network Bandwidth	100 Mbps
Maximum queue length	3000

Table 5. Mean service time of each file size (msec)

File size (S_i)	Service time ($\bar{T}_{S_i}$)
64 KB	10.00
128 KB	16.25
256 KB	32.85
512 KB	69.13
1 MB	133.03
2 MB	286.38

To compute the mean response time of a workload based on the *M/G/m* model using Equation (2), the mean service time for the workload needs to be obtained. The mean service time of a workload will depend on the distribution of the file sizes in the workload and the mean service time for each of those file sizes. Thus, to obtain the mean service times of the workloads, the mean service times of the file sizes in the workload need to be obtained.

To obtain the mean service time for a single file size, client requests at a very low arrival rate such that the traffic intensity is low (less than 0.4%) were presented to the server. For each file size, the average response time was computed using response time measurements for 200 requests. The low traffic intensity causes negligible queuing delay and the response time mostly consists of the service time. Thus, the mean service time was approximated by the response time under very low load, which for each file size is summarized in Table 5.

RESULTS AND DISCUSSION

In this section we present and discuss the results of the experimental validation of the *M/G/m* model. Several experimental scenarios for both deterministic and heavy-tailed workloads were considered, with each scenario comprised of a single workload (deterministic or heavy-tailed). For the deterministic workload, all the client requests were for the same file size. For the heavy-tailed workload, requests were generated according to the file size distribution for a given mean file size. In each scenario, request arrivals were generated according to a Poisson distribution with varying rates, chosen to maintain the condition $\rho < 1$ necessary for a stable system. Due to the high variability of the arrival rates of the Poisson process and for the file size distributions, the steady state mean queue length given by Equation (1) increases with ρ (Lipsky, 1992). The traffic intensity is considered to be low for $\rho < 0.3$, medium for $0.3 \leq \rho \leq 0.5$, and high for $\rho \geq 0.5$ (Lipsky, 1992; Nossenson & Attiya, 2004).

For each arrival rate, using the measured response times of 200 requests, the average response time was obtained. The mean response times were computed for the *M/G/m* model from Equation (2) and *M/G/1* model from Equation (3) using MATLAB. For each workload, the traffic intensity ρ for each arrival rate is also reported.

Deterministic Workload

The measured and the computed response times along with the traffic intensities for deterministic workloads of 64 KB, 128 KB, 256 KB, and 512 KB are reported in Tables 6, 7, 8, and 9, respectively. The results indicate that the response time predicted by the *M/G/m* model is very close to the measured response time for low to moderately high loads, with value of ρ around 0.5. The predicted response time is lower than the measured, and the deviations are within 7–8% for low and medium traffic intensities. As the traffic intensity increases, greater deviations occur. However, the deviation is within 10% even for high traffic intensities, with ρ close to 0.6. For the arrival rates and file size distributions, the system is already heavily loaded when $\rho \geq 0.5$ (Nossenson & Attiya, 2004) which cause these deviations.

The results also indicate that the response time predicted by the *M/G/1* model is always higher

Table 6. Deterministic workload (S_i = 64KB) (msec)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0024	10.07	10.00	10.03
1000	0.0049	10.17	10.01	10.05
500	0.0098	10.36	10.06	10.10
200	0.0243	10.55	10.14	10.26
100	0.0463	10.80	10.30	10.56
50	0.1067	11.05	10.79	11.25
40	0.1283	11.72	11.32	11.67
30	0.1743	12.74	12.03	12.50
20	0.2953	13.46	12.64	15.00
15	0.5405	16.88	15.41	--

Table 7. Deterministic workload (S_i = 128KB) (msec)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0045	17.07	16.26	16.32
1000	0.0083	17.20	16.33	16.38
500	0.0182	17.80	16.71	16.82
200	0.0446	18.11	17.08	17.17
100	0.0861	18.63	17.37	17.83
50	0.1764	20.38	17.13	20.16
40	0.2614	23.35	21.97	21.81
30	0.3485	24.47	22.72	25.85
20	0.5650	27.26	24.84	--

Table 8. Deterministic workload (S_i = 256KB) (msec)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0082	33.15	32.86	33.12
1000	0.0164	33.75	32.83	33.41
500	0.0329	34.07	32.97	34.01
200	0.0936	36.51	34.76	36.08
100	0.1947	39.25	38.09	40.89
50	0.3912	45.25	43.86	64.31
40	0.4993	50.38	47.45	108.31
35	0.5889	62.11	56.13	--

Table 9. Deterministic workload (S_i = 512KB) (msec)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0178	69.25	69.13	70.37
1000	0.0334	69.67	69.34	71.70
500	0.0688	70.53	69.74	74.68
200	0.1738	71.22	71.13	87.39
100	0.3944	90.13	87.45	146.53
80	0.5073	105.65	96.91	--
70	0.5683	110.69	100.26	--

than the response time predicted by the *M/G/m* model. As the arrival rate increases, the rate at which the predicted response time increases is much higher for the *M/G/1* model as compared to the *M/G/m* model. Further, as the arrival rate λ approaches service rate μ while maintaining λ < μ, the response time predicted by the *M/G/1* model deviates significantly from the measured response time. When μ < λ < 2μ, the response time of the single-server *M/G/1* model is not meaningful since it leads to ρ′ > 1, resulting in an unstable system. The *M/G/m* model, however, still represents a stable system for 0.5 < ρ < 1 and hence provides meaningful predictions of response time. Thus, a single-server queue is not adequate to represent a Web server equipped with concurrent processing capabilities.

Heavy-Tailed Workload

To compute the average response time for a heavy-tailed workload using Equation (2), the mean service time for the workload with a given mean file size can be computed using:

$$\overline{T}_{\overline{S}_i} = \sum_i \overline{T}_{S_i} P_{S_i} \qquad (10)$$

where S_i denotes the file size, $\overline{T}_{S_i}$ the mean service

time of file size S_i, P_{S_i} the probability of requesting a file of size S_i, and $\overline{T}_{\overline{S}_i}$ the mean service time of a workload with mean file size $\overline{S}_i$. As discussed earlier, the mean file sizes considered are 120KB, 150KB, 180KB, and 200KB. The mean service times of these workloads computed using the service times and probabilities reported in Table 2 and Table 5 respectively, are summarized in Table 10.

The measured and the computed response times for the heavy-tailed workload for different mean file sizes are summarized in Tables 11, 12, 13, and 14. The trend in the deviation between the predicted and measured response times in this case is similar as in the case of the deterministic workload. The predicted and the mean response times are within 10%, with the predicted one being lower than the measured for low and medium traffic intensities ($\rho < 0.5$). As indicated earlier, for the arrival rate and file size distributions, the system is already heavily loaded when $\rho < 0.5$ (Nossenson & Attiya, 2004) which cause these deviations. We note, however, that the difference between the computed and measured response time is around 12% even for moderately high traffic intensity with ρ close to 0.6. The response times predicted by the M/G/1 model follow similar trends as in the case of deterministic workload.

Table 10. Mean service times of heavy-tailed workload (msec)

Mean file size ($\overline{S}_i$)	Mean service time ($\overline{T}_{S_i}$)
120 KB	18.95
150 KB	21.99
180 KB	25.37
200 KB	27.74

RELATED RESEARCH

Several efforts have focused on Web server and workload performance modeling and analysis. Slothouber (1996) proposes to model a Web server as an open queuing network. Heidemann, Obraczka, and Touch (1997) present analytical models for the interaction of HTTP with several transport layers. Van der Mei et al. (2001) present an end-to-end queuing model for the performance of Web servers, encompassing the impact of client workload characteristics, server hardware/software

Table 11. Heavy-tailed workload ($\overline{S}_i$ = 120KB)

$1/\lambda$	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0047	19.65	18.95	19.04
1000	0.0097	20.30	19.34	19.43
500	0.0198	20.60	19.34	19.52
200	0.0534	21.25	19.98	20.04
100	0.1066	23.25	21.93	22.17
50	0.2414	24.25	22.56	24.73
40	0.2875	27.12	24.97	27.48
30	0.3731	31.27	28.92	35.20
20	0.5547	40.39	35.94	--

Table 12. Heavy-tailed workload ($\overline{S}_i$ = 150KB)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0053	22.32	22.00	22.11
1000	0.0099	23.05	22.52	22.74
500	0.0255	24.31	23.64	23.80
200	0.0582	26.44	24.50	24.75
100	0.1215	27.31	25.30	25.89
50	0.2253	27.64	25.59	30.62
40	0.2974	29.22	26.97	35.41
30	0.4455	32.78	29.94	52.17
25	0.5631	38.25	34.03	--

Table 13. Heavy-tailed workload ($\overline{S}_i$ = 180KB)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0056	25.63	25.37	25.53
1000	0.0117	25.86	25.41	25.70
500	0.0243	27.48	26.35	26.65
200	0.0673	28.69	27.23	27.61
100	0.1291	30.67	29.63	30.68
50	0.2843	33.40	31.90	38.44
40	0.3636	35.02	32.33	47.37
30	0.5303	43.12	39.15	--

Table 14. Heavy-tailed workload ($\overline{S}_i$ = 200KB)

1/λ	ρ	$R_{measure}$	$R_{M/G/m}$	$R_{M/G/1}$
2000	0.0066	28.43	27.74	27.94
1000	0.0136	29.28	28.21	28.64
500	0.0266	30.55	29.01	29.55
200	0.0714	31.17	29.87	30.97
100	0.1553	33.65	31.88	33.06
50	0.3105	37.75	35.08	45.02
40	0.4006	39.66	36.89	59.12
30	0.5637	49.50	44.15	--

configurations, communication protocols, and interconnect topologies. Kamra, Misra, and Nahum (2004) present a control-theoretic approach that achieves dual objectives, namely, preventing overload and enforcing absolute response times. Cao et al. (2003) use an *M/G/1/K*PS* queuing model to model the performance of a Web server. Nossenson and Attiya (2004) introduce a new *N-Burst/G/1* queuing model with heavy-tailed service time distribution for Web server performance modeling. Squillante et al. (1999) employ a *G/G/1* queue to model high-volume Web sites. Liu, Heo, and Sha (2005) provide a model of a three-tiered Web services architecture, where each tier is modeled by a multistation queuing center. Kant and Sundaram (2000) present a queuing network model for a multiprocessor system running a static Web workload. Hu, Nanda, and Yang (1999) measure and analyze the behavior of a Web server driven by benchmarks and propose techniques for performance improvement. Hardwick, Papaefstathiou, and Guimbellot (2001) use a performance modeling framework, to create a performance analysis tool for database-backed Web sites. Kohavi and Parekh (2003) offer several useful, practical recommendations for supplementary analyses. Iyengar, Challenger, Dias, and Dantzig (2000) present techniques to be used at popular sites to improve performance and availability based on two case studies.

Previous works mostly use a *single* server queue to model a Web server. However, modern Web servers typically have multiple processes/threads that work independently and simultaneously to service requests. Thus, it is appropriate to consider this system a multiserver system, as in the *M/G/m* model in this article. While being an accurate representation of the characteristics of a modern Web server, the *M/G/m* model is simple to comprehend, needs the estimation of very few parameters, and involves straightforward computations, which makes it easy to apply. Since the model is not tied to a specific implementation of concurrency, it is equally applicable to a thread-based, a process-based, or hybrid architecture.

CONCLUSION AND FUTURE RESEARCH

In this article we propose the use of an *M/G/m* queue to model the performance of a Web server capable of processing multiple requests concurrently. The performance metric of interest is the response time of a client request. We validate the model experimentally for deterministic and heavy-tailed workloads. Our results indicate that the *M/G/m* queue provides a reasonable estimate of the service response time for low to moderately high traffic intensities. It is a more accurate representation of modern server characteristics, is conceptually simple, requires the estimation of very few parameters, involves straightforward computations, and is hence easy to apply.

Our future research includes extending the methodology to: (i) consider general arrival processes in performance analysis (Nossenson & Attiya, 2004), (ii) apply the methodology to model the performance of an application server in a three-tier Web services architecture, and (iii) consider QoS provisioning and overload control.

REFERENCES

Apache Software Foundation (n.d.). *Apache HTTP server project.* Retrieved June 9, 2008, from http://httpd.apache.org/

Cao, J., Andersson, M., Nyberg, C., & Kihl, M. (2003). Web server performance modeling using an *M/G/1/K*PS* queue. In *Proceedings of the 10th International Conference on Telecommunications* (pp. 1501–1506).

Crovella, M., Taqqu, M. S., & Bestavros, A. (1998). *Heavy-tailed probability distributions in the World Wide Web. A practical guide to heavy tails: Statistical techniques and application.*

Boston: Birkhauser.

Faloutsos, M., Faloutsos, P., & Faloutsos, C. (1999). On the power-law relationships of the Internet topology. In *Proceedings of ACM SIGCOMM* (pp. 251–262).

Goseva-Postojanova, K., Mazimdar, S., & Singh, A. (2004). Empirical study of session-based workload and reliability of Web servers. In *Proceedings of the 15th International Symposium on Software Reliability Engineering* (pp. 403–414).

Hardwick, J. C., Papaefstathiou, E., & Guimbellot, D. (2001). Modeling the performance of e-commerce sites. In *Proceedings of the 27th International Conference of the Computer Measurement Group* (pp. 3–12).

Heidemann, J., Obraczka, K., & Touch, J. (1997). Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking, 5*(5), 616–630.

Hokstad, P.(1978). Approximation for the *M/G/m* queue. *Operations Research, 26*(3), 510–523.

Hu, Y., Nanda, A., & Yang, Q. (1999). Measurement, analysis and performance improvement of the Apache Web server. In *Proceedings of the IEEE International Performance, Computing and Communications Conference* (pp. 261–267).

Iyengar, A., Challenger, J., Dias, D., & Dantzig, P. (2000). High-performance Web site design techniques. *IEEE Internet Computing, 4*(2), 17–26.

Jin, S., & Bestavros, A. (2005). *Generating Internet streaming media objects and workloads* (chap. 1, Recent advances on Web content delivery). Kluwer Academic Publishers.

Kamra, A., Misra, V., & Nahum, E. (2004). Controlling the performance of 3-tiered Web sites: Modeling, design and implementation. In *Proceedings of SIGMETRICS 2004/PERFORMANCE 2004* (pp. 414–415).

Kant, K., & Sundaram, C. R. M. (2000). A server performance model for static Web workloads. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software* (pp. 201–206).

Kimura, T. (1983). Diffusion approximation for an *M/G/m* queue. *Operations Research, 31*(2), 304–321.

Kleinrock, L. (1976). *Queueing systems, Volume 1: Theory.* New York: John Wiley & Sons.

Kohavi, R., & Parekh, R. (2003). Ten supplementary analyses to improve e-commerce Web sites. In *Proceedings of the 5th WEBKDD Workshop* (pp. 29–36).

Ling, Y., Mullen, T., & Lin, X. (2000). Analysis of optimal thread pool size. *ACM SIGOPS Operating System Review, 34*(2), 42–55.

Lipsky, L. (1992). *Queueing theory: A linear algebraic approach.* New York: McMillan and Company.

Liu, X., Heo, J., & Sha, L. (2005). Modeling 3-tiered Web applications. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems* (pp. 307–310).

Menasce, D. (2003). Web server software architecture. *IEEE Internet Computing, 7*(6), 78–81.

Microsoft Corporation (n.d). *Internet information services (IIS).* Retrieved June 9, 2008, from http://www.microsoft.com/WindowsServer2003/iis/default.mspx

Nossenson, R., & Attiya, H. (2004). The N-burst/G/1 model with heavy-tailed service-times distribution. In *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*

(pp. 131–138).

Paxson, V., & Floyd, S. (1995). Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking, 3*(3), 226–244.

Richter, J. (2000). *Programming server-side applications for Microsoft Windows 2000*. Microsoft Press.

Sakasegawa, H. (1977). An approximation formula $L_q = ar^b/(1-r)$. *Annals of the Institute of Statistical Mathematics, 29*(1), 67–75.

Slothouber, L. (1996). A model of Web server performance. In *Proceedings of the 5th International World Wide Web Conference.*

Squillante, M. S., Yao, D. D., & Zhang, L. (1999). Web traffic modeling and Web server performance analysis. In *Proceedings of the 38th Conference on Decision and Control* (pp. 4432–4439).

Van der Mei, R. D., Hariharan, R., & Reeser, P. (2001). Web server performance modeling. *Telecommunication Systems, 16*(3–4), 361–378.

Xu, D., & Bode, B. (2004). Performance study and dynamic optimization design for thread pool systems. In *Proceedings of the International Conference on Computing, Communications and Control Technologies.*

Yao, D. (1985). Refining the diffusion approximation for the *M/G/m* queue. *Operations Research, 33*(6), 1266–1277.

This work was previously published in the International Journal of Information Technology and Web Engineering, edited by G. Alkhatib, Volume 3, Issue 3, pp. 50-65, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 1.26
Software Modernization of Legacy Systems for Web Services Interoperability

Chia-Chu Chiang
University of Arkansas at Little Rock, USA

INTRODUCTION

Software maintenance is an inevitable process due to program evolution (Lehman & Belady, 1985). Adaptive maintenance (Schenidewind, 1987) is an activity used to adapt software to new environments or new requirements due to the evolving needs of new platforms, new operating systems, new software, and evolving business requirements. For example, companies have been adapting their legacy systems to Web-enabling environments of doing business that could not have been imagined even a decade ago (Khosrow-Pour & Herman, 2001; Werthner & Ricci, 2004).

To understand software modernization of legacy systems for Web services, it is necessary to address how legacy integration has evolved from centralized computing to distributed, component-based computing due to the advent and widespread use of object-oriented and client-server technologies. Legacy systems were typically developed on a centralized, terminal-to-host architecture. Users usually accessed their legacy systems through terminals that included character-based menus and data entry screens. Consequently, legacy systems built on the central mainframe are inaccessible remotely without adaptations.

Component-based middleware technologies, such as Java RMI, common object request broker architecture (CORBA), and component object model/distributed component object model (COM/DCOM), provide solutions to support the interoperability of legacy systems in a heterogeneous and distributed environment (Chiang, 2001). Unfortunately, the technologies have proved to be insufficient in application integration solutions for several reasons (Stal, 2002). Although the technologies share common communication architectural foundations, the implementation of each technology differs in several aspects, including the object models provided, the communication protocols, and data marshaling/demarshaling. Due to the proprietary implementations of the technologies, they do not interoperate well with

each other. Obviously, existing component-based middleware only partially solves the interoperability problems of legacy systems. More effort is still required to make the legacy systems totally interoperable in a heterogeneous and distributed environment.

BACKGROUND

Web services have been widely considered as a better solution to legacy integration for software interoperability using open standards that include extensible markup language (XML), the simple object access protocol (SOAP), the Web services description language (WSDL), and the universal description, discovery, and integration (UDDI) (Chung, Lin, & Mathieu, 2003; Stal, 2002; Zhang & Yang, 2004). Service requesters and providers follow the Web service standards for message exchanges. When a service provider has a service for public exposure, it must write a description of the service in WSDL and register the service description with UDDI to a global repository. A service requester can then query the repository using UDDI to retrieve the service description. The service requester uses the service description in WSDL to send requests, and the service provider replies to the requests under SOAP.

LEGACY MODERNIZATION FOR WEB SERVICES AND CHALLENGES

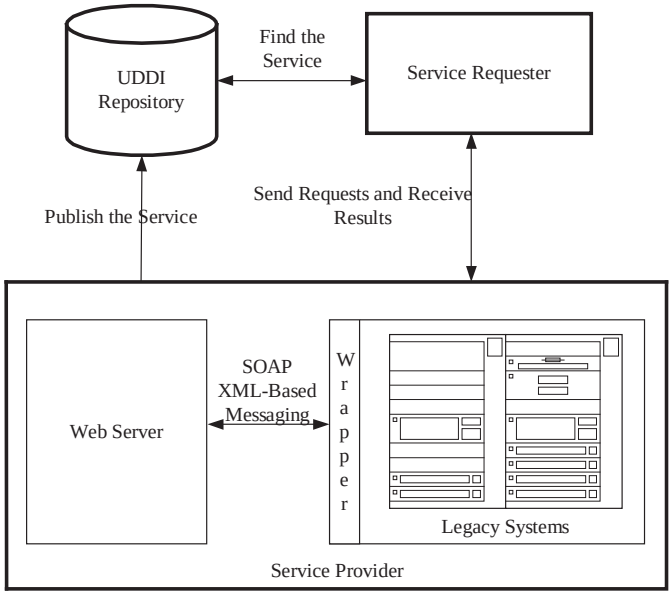
There are three main reasons for modernizing legacy systems: to reduce the system evolution risk, to recoup the investment on the systems, and to make the system distributed and scalable for business-to-consumer and business-to-business, as well as making it highly available to Web users.

Companies usually have two approaches to turn their legacy systems into Web services: wrapping and reengineering. Wrapping provides a cost-effective way to integrate legacy systems with Web services into a heterogeneously distributed computing environment. Unfortunately, the wrapping approach requires the whole legacy system to be exposed to the public as a Web service, which fails to properly abstract the system (Vinoski, 2002; Vogels, 2003). Furthermore, the wrapping approach increases the difficulty of maintaining the legacy system in the long run. Thus, the wrapping approach is generally a temporary solution, rather than a strategic one. The reengineering approach applies reverse engineering techniques to legacy systems to recover business rules, and develop Web services from the extracted business rules. This approach streamlines legacy systems but is highly dependent on the success of recovery on the business rules from legacy systems.

Wrapping legacy systems for Web services can be performed through wrappers or adapters. A wrapper is built to encapsulate a legacy system and provide access to the legacy system through the encapsulation layer. This layer exposes only the methods with parameter attributes to remote service requesters. In addition, the wrapper must resolve the incompatible communication issues between the legacy systems and the Web server using SOAP/XML messaging. Therefore, programmers are required to write a wrapper to reconcile the issues, as well as a WSDL for public exposure. Unfortunately, a wrapper is difficult to maintain, inefficient, and error-prone (Engelen, Gupta, & Pant, 2003). A sample Web service architecture via a wrapper is shown in Figure 1.

Turning legacy systems in middleware-based components into Web services is slightly different from the technique described above. Because the legacy system has already been wrapped in middleware, companies may be unwilling to unwrap their systems in order to turn the system into a Web service. Fortunately, there are Web services toolkits available to turn middleware-based componentized legacy systems into Web services (Engelen, Gupta, & Pant, 2003). First, the toolkits translate the interface definition of a

Figure 1. A Web service architecture to access legacy systems via wrappers

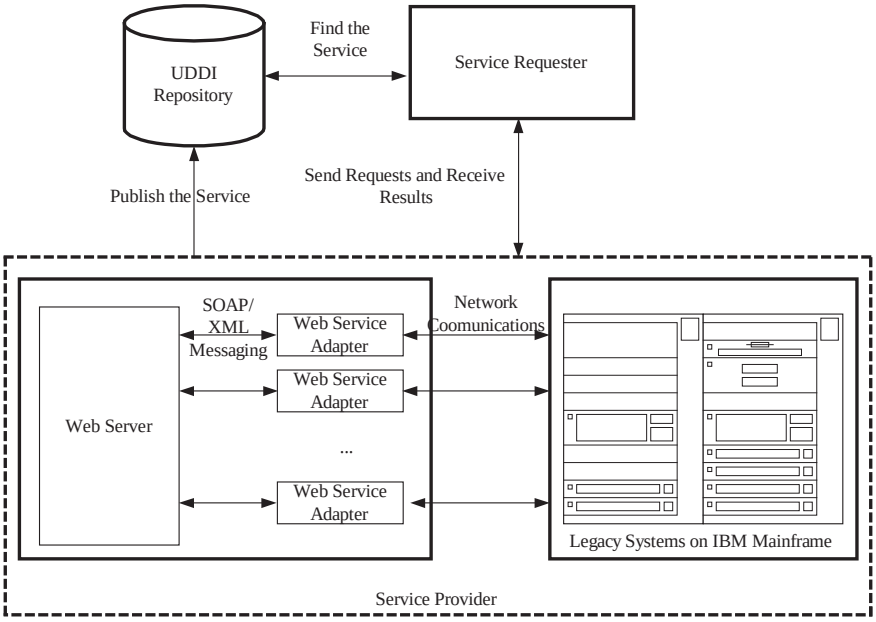


component in the interface definition language (IDL) into a service description in WSDL for public exposure. These toolkits then provide a wrapper component that enables distributed access to the component as a Web service through SOAP. A service requester can then find the WSDL in the registry and interact with the component as a Web service. Using the Web services toolkits, the creation of a wrapper component can be simplified. However, the technique could quickly reach design limitations as legacy systems continue to evolve. One limitation is that the operating system and programming language support in middleware is totally dependent on the development and deployment platforms that are offered by middleware vendors. In many situations, the platforms supporting the legacy systems may not be available for the development of middleware. An alternative to solve this problem is to create an adapter on an available platform and use some other strategy.

An adapter can resolve the language and platform dependency through inter-language binding, inter-process communications, and network communications. Compared to a wrapper, the adapter provides a better and more flexible solution to encapsulate legacy systems by reducing some degree of platform, operating system, and programming language dependency. However, an adapter may create an additional point of failure introduced by a service provider. A sample architecture of using an adapter is shown in Figure 2, and an application of this architecture is the IBM Web Services Technologies (Barcia, Hines, Alcott, & Botzum, 2004; Kreger, 2001).

In Figure 2, a Web service adapter is developed for each service invocation. The connection between the legacy systems and the adapter can be

Figure 2. A Web service architecture to access legacy systems via adapters



any communication protocol that includes TCP/IP and CICS supported by the backend server hosting the legacy system. The Web service adapter may call one single backend system per SOAP service request. It is also possible for the adapter to transform one SOAP request to multiple requests to one or more backend systems. The combined results of the backend requests are composed into one SOAP response, which is then passed back to the server requester.

Unlike the wrapping approach, the reengineering approach develops business rules from legacy systems into Web services. Legacy systems are first thoroughly analyzed for understanding. Code corresponding to the business rules is identified and extracted into a library for the creation of Web services as shown in Figure 3.

Next, the interface of the extracted code as a Web service is constructed for creating the Web service in WSDL. The WSDL specification is then compiled into a skeleton of a service provider as shown in Figure 4. Whenever a service requester sends a request message to the service provider, the skeleton automatically demarshals the request and invokes the service. When the service provider returns the results, the skeleton marshals the results into a response message and sends the response back to the service requester.

The WSDL specification of a service provider is also used to create a stub of a service requester for the invocation of the service remotely. The WSDL specification is retrieved from a UDDI repository, and a description of the remote procedures is created from the WSDL specification. The description of the remote procedures is then compiled into a stub of the service requester for marshaling and demarshaling request and response messages at runtime. The development

Figure 3. Code extraction from legacy systems

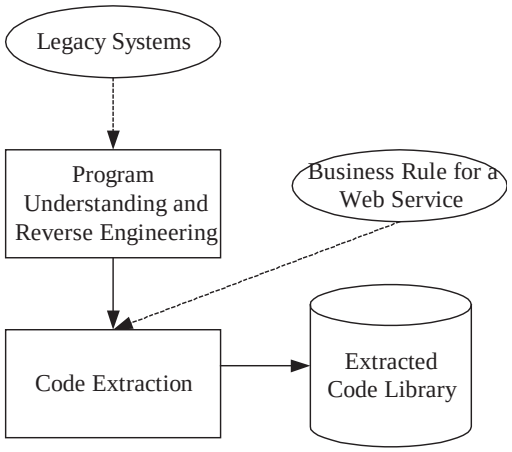


Figure 4. Creation of skeletons via a WSDL compiler

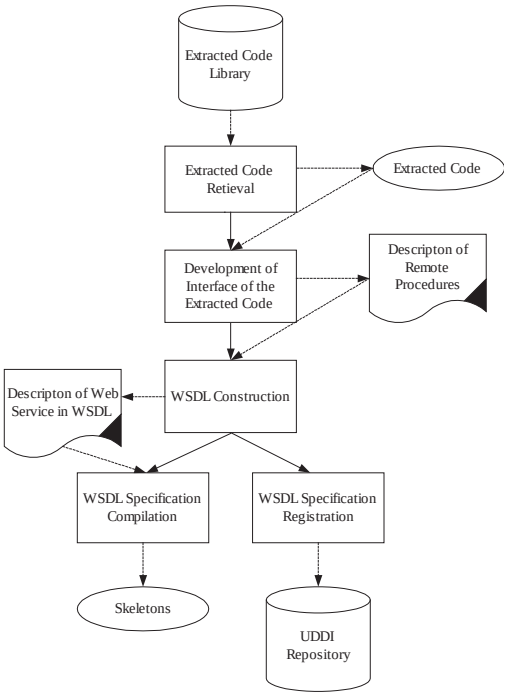
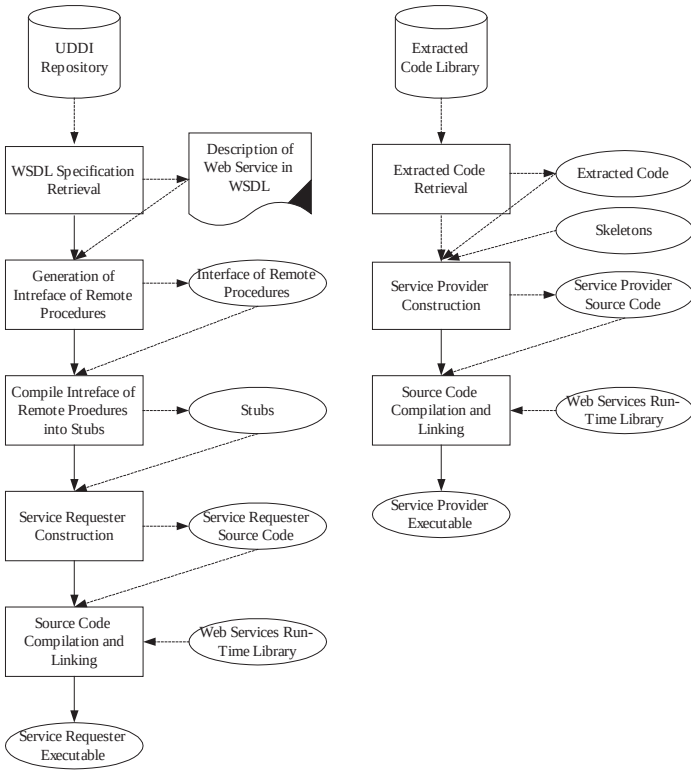


Figure 5. Construction of service requester and provider executables



strategy of constructing executable files of a service requester and provider for Web services is illustrated in Figure 5.

In this approach, a WSDL specification does not define any language bindings. This allows stubs and skeletons to be constructed in any programming languages, such as C, C++, Visual Basic, Perl, and Python. Because a service provider's skeleton leaves the implementation of the service unspecified, the extracted code from the legacy system can be inserted into the skeleton to make the concrete implementation of the service. This approach is more than just converting code from one language to another. It is the conversion of an entire system architecture into a Web services architecture, including the user interfaces and database structures. The approach to developing Web services takes the same development strategy as the middleware approach for the development of distributed applications. The proxies are automatically generated from a WSDL specification describing the interface of the service. The extracted code from the legacy system fills in the concrete implementation of the service. Samples of the reengineering approach can be found in references (Engelen, Gupta, & Pant, 2003; Graveley, 2001). Two primary Web services architectures supporting the reengineering approach are the Java 2 platform Enterprise Edition (J2EE) and the .NET platform (Erl, 2005). The J2EE platform is a development and runtime environment based on the Java programming language. The .NET platform (MacDonald, 2003) is a proprietary development and runtime environment developed for use with Windows operating systems. The .NET platform provides unified support for a set of programming languages including Visual Basic, C, C++, and C#.

Web services identification plays an important role in the reengineering approach. It starts with program understanding. Program understanding via reverse engineering provides different views of abstraction of legacy systems which are helpful for identifying services. The code corresponding to the services is extracted from the legacy systems in terms of the business rules defined in the systems. Several techniques used for supporting reverse engineering and design recovery activities have been cataloged in various collections and surveys (Arnold, 1994; Bellay & Gall, 1997; Zvegintzov, 1997). Program slicing techniques (Huang et al., 1996; Sneed & Erdos, 1996; Wang, Sun, Yang, He, & Maddineni, 2004) are used to extract source code corresponding to the business rules from legacy systems. The extracted code may be used as a candidate for Web services. Due to the inherent complexity of the system and the language in which it is written, existing program slicing techniques cannot be fully automated yet. Therefore, human interventions are required to filter the candidates and should be kept to a minimum. Clustering techniques (Wiggerts, 1997) can be used to group the relevant extracted code into classes in terms of the attributes that provide promising candidates for Web services.

Despite all its promises and glories (Arsanjani, Hailpern, Martin, & Tarr, 2003), software modernization of legacy systems for Web services is not free of associated risks and challenges (Tilley et al., 2002). From a business perspective, there are issues of cost, manpower, management, maintenance, personnel pressure, and return on investment that must be examined when considering the integration of legacy systems to a Web services computing environment. The immaturity of these technologies may also offer technical challenges. The current generation of Web services infrastructures and tools has problems of large space consumption and performance degradation. Tools of reverse engineering and program slicing are not mature enough to be fully automated. These issues need to be addressed as these technologies evolve. In order to make a rational decision on the potential of each technology, one must assess the costs, limitations, and risks (Seacord, Plakosh, & Lewis, 2003; Ulrich, 2002). The issues, problems,

and limitations of modernizing legacy systems for Web services interoperability are summarized as follows:

- **Economics of modernizing legacy systems:** Inability to assess the costs of each approach
- **Security:** Maintaining secure and safe systems and keeping unauthorized user access out
- **Performance:** Degrading the system's performance due to the migration from one environment to another or from one language to another
- **Quality:** Failing to attain the quality of the system due to a poorly designed quality plan or unforeseen results from the migration
- **Acceptance:** Rejection of the system by users due to lack of knowledge of the Internet and its usefulness
- **Exposure points:** Risks associated with accessing the firm's systems remotely
- **Privacy and confidentiality agreements:** Addressing an individual's right to privacy and the sharing of confidential information
- **Automated tools:** Effective tools that reduce user intervention during the migration

FUTURE TRENDS

Companies will continuously modernize their legacy systems from centralized to distributed and Web-enabled environments due to the needs of evolving business requirements. Companies may choose to outsource the legacy modernization process for Web services to another company that specializes in such tasks. Automated tools used for extracting business rules from legacy systems will be continuously required for construction of Web services. Web service support tools that migrate middleware-based components into Web services will continue to improve the technology for reliability and transactional guarantees.

CONCLUSION

The applications of the World Wide Web (WWW) are used not only for information gathering, but also as an exciting technological breakthrough providing companies with new opportunities for conducting their businesses. Business applications must respond to changing business requirements quickly to combat conflicts resulting from having heterogeneous computing environments. Modernizing legacy systems for Web services plays a key role for companies to achieve their system integration. The critical issues discussed in this paper provide many implications and challenges to the companies. These issues must be dealt with before new issues arise as Web services technologies continue to evolve.

REFERENCES

Arnold, R. S. (1994). *Software reengineering.* Los Altimos, CA: IEEE Computer Society Press.

Arsanjani, A., Hailpern, B., Martin, J., & Tarr, P. (2003). Web services: Promises and compromises. *ACM Queue, 1*(1). Retrieved March 27, 2006, from http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid =31

Barcia, R., Hines, B., Alcott, T., & Botzum, K. (2004). *IBM WebSphere: Development and advanced configuration*. New York: Prentice Hall PTR.

Bellay, B., & Gall, H. (1997). A comparison of four reverse engineering tools. *Proceedings of the 4th Working Conference on Reverse Engineering* (pp. 2-11).

Chiang, C-C. (2001). Wrapping legacy systems for use in heterogeneous computing environments. *Information and Software Technology, 43*(8), 497-507.

Chikofsky, E. J., & Cross II, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software, 7*(1), 13-17.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web services description language (WSDL) 1.1.* Retrieved March 27, 2006, from http://www.w3.org/TR/wsdl

Chung, J.-Y., Lin, K.-J., & Mathieu, R. G. (2003). Web services computing: Advancing software interoperability. *Computer, 36*(10), 35-37.

Engelen, R., Gupta, G., & Pant, S. (2003). Developing Web services for C and C++. *IEEE Internet Computing, 7*(2), 53-61.

Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design.* New York: Prentice Hall PTR.

Graveley, A. (2001). Making SOAP with soup. *Proceedings of Ottawa Linux Symposium.* Retrieved from http://lwn.net/2001/features/OLS/pdf/pdf/soup.pdf

Haas, H., & Brown, A. (2004). *Web services glossary.* Retrieved March 27, 2006, from http://www.w3.org/TR/ws-gloss/

Huang, H., Tsai, W. T., Bhattacharya, S., Chen, X. P., Wang, Y., & Sun, J. (1996). Business rule extraction from legacy code. *Proceedings of IEEE 20th Computer Software and Applications* (pp. 162-167).

Institute of Electrical and Electronics Engineering. (1993). *IEEE standard for software maintenance* (IEEE Publication No. IEEE STD 1219). Los Altimos, CA: IEEE Computer Society Press.

Khosrow-Pour, M., & Herman, N. (2001). Web-enabled technologies assessment and management: critical issues. In Khosrow-Pour & Herman (Eds.), *Managing Web-enabled technologies in organizations: A global perspective* (pp. 1-22). Hershey, PA: Idea Group Publishing.

Kreger, H. (2001). *Web services conceptual architecture (WSCA 1.0).* Retrieved March 27, 2006, from http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

Lehman, M. M., & Belady, L. (1985). *Program evolution: Processes of software change.* London: Academic Press.

MacDonald, M. (2003). *Microsoft .NET distributed applications: Integrating XML Web services and .NET remoting.* Redmond, NY: Microsoft Press.

Schneidewind, N. F. (1987). The state of software maintenance. *IEEE Transactions on Software Engineering, 13*(3), 303-310.

Seacord, R. C., Plakosh, D., & Lewis, G. (2003). *Modernizing legacy systems: Software technologies, engineering processes, and business practices.* Boston: Addison-Wesley.

Sneed, H. M., & Erdos, K. (1996). Extracting business rules from source code. *Proceedings of IEEE 4th International Workshop on Program Comprehension* (pp. 240-247).

Stal, M. (2002). Web services: Beyond component-based computing. *Communications of the ACM, 45*(10), 71-76.

Tilley, S., Gerdes, J., Hamilton, T., Huang, S., Müller, H., & Wong, K. (2002). Adoption challenges in migrating to Web services. *Proceedings of the Fourth International Workshop on Web Site Evolution.*

Ulrich, W. M. (2002). *Legacy systems: Transformation strategies.* New York: Prentice Hall PTR.

Vinoski, S. (2002). Web services interaction model. *IEEE Internet Computing, 6*(3), 89-91.

Vogels, W. (2003). Web services are not distributed objects. *IEEE Internet Computing, 7*(6), 59-66.

Wang, X., Sun, J., Yang, X., He, Z., & Maddineni, S. R. (2004). Business rules extraction from large legacy systems. *Proceedings of IEEE Eighth European Conference on Software Maintenance and Reengineering* (pp. 249-254).

Werthner, H., & Ricci, F. (2004). E-commerce and tourism. *Communication of the ACM, 47*(12), 101-105.

Wiggerts, T. A. (1997). Using clustering algorithms in legacy systems remodularization. *Proceedings of the Fourth Working Conference on Reverse Engineering* (pp. 33-43).

Zhang, Z., & Yang, H. (2004). Incubating services in legacy systems for architectural migration. *Proceedings of the 11th Asia-Pacific Software Engineering Conference* (pp. 196-203).

Zvegintzov, N. (1997). A resource guide to year 2000 tools. *Computer, 30*(3), 58-63.

KEY TERMS

Reverse Engineering: Reverse engineering is the process of discovering the functions and their interrelationships of a software system as well as creating representations of the system in another form or at a higher level of abstraction.

SOAP: The W3C definition of SOAP is "a set of protocols governing the format and processing rules of SOAP messages."

Software Maintenance: Software maintenance is the process of enhancing and adapting a software product after delivery as well as correcting faults.

Software Reengineering: Chikofsky and Cross define software reengineering as "the examination and alternation of a software system to reconstitute it in a new form and subsequent implementation of that form."

UDDI: UDDI is a Web services registry and discovery technology for strings and retrieving Web services interfaces.

Web Services: The W3C definition of a Web service is "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in WSDL. Other systems interact with the Web services using SOAP messaging defined in the WSDL specification."

WSDL: The W3C definition of WSDL is "an XML format for describing Web services interfaces, message types, operations, and protocol mappings."

This work was previously published in Encyclopedia of Internet Technologies and Applications, edited by M. Freire & M. Pereira, pp. 551-557, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 1.27
Approaches to Building High Performance Web Applications:
A Practical Look at Availability, Reliability, and Performance

Brian Goodman
IBM Corporation, USA

Maheshwar Inampudi
IBM Corporation, USA

James Doran
IBM Corporation, USA

ABSTRACT

In this chapter, we introduce five practices to help build scalable, resilient Web applications. In 2004, IBM launched its expertise location system, bringing together two legacy systems and transforming the employee's ability to find and connect with their extensive network. This chapter reviews five of the many issues that challenge enterprise Web applications: resource contention, managing transactions, application resiliency, geographic diversity, and exception perception management. Using the IBM expertise location system as context, we will present five key methods that mitigate these risks, achieving high availability and high performance goals.

INTRODUCTION

In this chapter, we introduce five practices for building scalable, resilient Web applications. First, we briefly review the context in which IBM launched its internal expertise location system in 2004. We then introduce the challenges we faced in implementing the business requirements and present five key methods that mitigated risks to achieving our high availability and performance goals.

Specifically, we will look at:

- caching strategies for high availability Web applications: beyond storing copies of HTML (Challenger, Dantzig, & Iyengar, 1998; Iyengar & Challenger, 1997);
- asynchronous task processing within Web applications: removing non-essential linear logic from high-volume transactions (Grand, 2002);
- building self-reliant autonomous behavior: encapsulating through services achieving tight integration with *true* loose coupling (Birman, van Renesse, & Vogels, 2004);
- client-side Model View Control (MVC): moving MVC to the browser supercharging the response times and getting a "wow" user experience (Murry, 2005; Sun Microsystems, 2002);
- graceful degradation: keeping users thinking and feeling "fast," "reliable," and "always on" (Florins & Vanderdonckt, 2004).

Caching Strategies

Caching strategies are a core part of high-performing Web experiences. In many cases (Amiri, Park, & Tewari, 2002; Candan, Li, Luo, Hsiung, & Agrawal, 2001; Liebmann & Dustdar, 2004; Rodriguez, Spanner, & Biersack, 2001), the assumption is that caching occurs at the edge of the network, the closest point to the consumer and the furthest from the data or application. Another somewhat overlooked approach is object caching.

Davison (2001) provides a wonderful primer on Web caching that illustrates the principles of caching and highlights some of the issues that may arise due to its use. Edge caching, or Web caching, is focused on storing and managing Web pages (static or dynamic) to help speed up transaction times. As the complexity of Web architecture evolves, applications have become more distributed. Edge solutions exemplify this, placing caches of content, sometimes fragments of executable code, in multiple geographical locations.

In recent years, object caching has enjoyed a revival and is now seen as a more desirable component of Web application architecture. Caching objects and sharing them across an infrastructure is a compelling capability. Often, the cost associated with building an object is considered quite high. The difference with an object cache mechanism is that it often resides at the Web server (Jadav & Gupta, 1997), the Web application, or as a middleware between the data and the Web application logic. Once an object is built, it can be cached, distributed, and managed for future reuse; it delivers performance at the application layer, whereas edge caching offers performance benefits to the delivery of data.

The strategies and issues found in edge caching are very similar to those encountered in object caching, and understanding their roles can offer a more complete view of a caching strategy. If it is possible to reuse an object (whether it is HTML or an object created from multiple data sources), an opportunity exists to increase performance. Caching takes advantage of predicting what resource might be required in the near future and storing a copy for later use. Later in the chapter, we explore approaches to managing custom object caches and situations where they can provide impressive benefits.

Asynchronous Processing

Asynchronous processing is often thought of as multithreaded computing or parallel computing, both of which are far more technical than intended. A common trap in transaction processing is the preconceived notion that all processing has to be linear. There are opportunities to move non-critical or batch-oriented logic out of the critical path of handling a transaction. However, this often introduces a level of complexity that surrounds the execution of tasks that are not time-critical.

For example, Reynal (2005) offers a palatable introduction to failure detectors, a common component of an asynchronous system. Failure detectors help the system decide when activity is not behaving as it should. For instance, an asynchronous task might connect to an XML feed to refresh a local data cache. Obviously, a simple rule such as *all transactions must take no longer than one minute* to be considered healthy might suffice. At the other end of the spectrum, a failure detector might consider many variables such as historical averages or current operating features like a process' memory usage. Asynchronous processing offers the opportunity to offload activities that are less time-sensitive and speed up a given transaction. While it introduces a different level of management, it can be done very effectively and contributes to the overall performance of a system.

Resilient and Autonomic Behavior

In recent years, resilient and autonomic behavior has been posited as one of the next great evolutions in computing, driven primarily by the need to reduce maintenance and administration of increasingly large and distributed systems (Kephart, 2005). The goal is to embed the capability for an entire system, including hardware, middleware and user facing systems, to take a proactive approach towards maintaining a level of performance and reliability with little or no human intervention. At the core of this motivation is the need to build resiliency and autonomic capability into the component portfolio itself. Service-oriented architecture (SOA) is seen as an opportunity to provide systems with more resilient and autonomic features (Gurguis & Zeid, 2005). The increased dependency on Web services as a critical part of an application's building blocks introduces a new requirement of availability. Birman et al. (2004) discuss making Web services highly available through extending the base standards with high assurance, enhanced communication, and fault detection, an insightful collection of capabilities that enable more consistent performance. While SOA offers loose coupling, problems in one part of the system can have ripple effects on others. Availability, resiliency, and self-management are core attributes to a high performance system.

Client-Side Model View Control

Within the last year, a great deal of focus has been placed on richer Web application front ends (Paulson, 2005; Weiss, 2005). Web 2.0 or Ajax (Asynchronous JavaScript and XML) offer compelling user experiences by delivering the model view control (MVC) design paradigm to the browser. The Web client can update its data without end user action or refreshing the user experience. The current page does not need to reload to pull new data. It uses JavaScript and the browser's XMLHttpRequest object to connect back to the server. The server might be running a Web service, but it is not required. The document format does not have to be XML, though it often is. Using the Ajax approach to build user interfaces for Web applications delivers a more responsive user experience. Later in the chapter, we examine the use of the Web 2.0 model to deliver a high performance Web experience to bandwidth-constrained geographies. Delivering a fast end-user experience can be the key to persuading users that they are working with a superior application.

Graceful Degradation

Andoni and Staddon (2005) offer a unique approach to notifying users that their status with application is about to change. They describe how a user might have access to a certain level of content, but are delinquent on the payment of a bill, thus jeopardizing access to the service. Andoni and Staddon offer an example of graceful degradation where the service not only delivers a less optimal experience, but links the warning

of their eminent status change to the content that they are viewing. The idea is that the user gets less and less value out of the content; for example, video content becomes choppier (Andoni et al, 2005). The key to graceful degradation is in altering the user experience to communicate that some features are not available while maintaining the overall performance and availability of the application. All too often, applications are built with hard dependencies on certain features that do (but should not) prevent users from benefiting from the other unaffected capabilities. Building high performance applications involves offering a system that can temporarily modify itself to maintain an acceptable level of performance while setting expectations, so that end users are subtly aware of any issues but are not helpless.

Using each of these approaches, we will identify the circumstances that lead the architectural and design changes, review the relevant design patterns, discuss the real-world implementation, and consider the before and after results. While IBM's expertise solution offers many opportunities for innovative thinking, the concepts discussed in this chapter are generally applicable when reviewing or designing Web applications for high load, high availability and fast performance.

SETTING THE CONTEXT

Web application design and architecture have matured greatly in recent years. Numerous books and articles discuss everything from design patterns and enterprise architecture to solution deployment strategy and high availability infrastructure (Alur, Crupi, & Malks, 2001; Eckstein, 2003; Fowler, 2000; Gamma, 2002; Grand, 2002; Schmid & Rossi, 2004). The current art grew out of the need to provide highly scalable and fast Web experiences to match the growing sophistication of end users (Burns, Rees, & Long, 2001; Goodman & Kebinger, 2004; Iyengar & Challenger, 1997; Malveau & Mowbray, 2001; Schmid & Rossi, 2004; Zhang & Buy, 2003). One area lacking in the current literature is a real-world strategy for applying appropriate design patterns and architectural approaches that improve the performance and reliability of today's enterprise Web experiences. In addition, intranets often impose additional constraints, budgets, legacy systems, network bandwidth limitations, and the general desire to do more with less (Appel, Dhadwal, & Pietraszek, 2003).

Intranets are a fertile ground for trying new approaches in the hope of solving real problems. The payoff for finding the right architecture or design is that it often leads to solutions that perform to expectations, are easily maintained and, most importantly, help transform the business. Popular examples of this are expertise location systems (AskMeCorp, 2005; Autonomy, 2005; Entopia, 2005; Tacit, 2005). These systems allow users to find people or answers to questions in a fast and efficient way. Helping employees in an enterprise to find each other or other knowledge repositories has a direct impact on the company's ability to draw from its network in real-time.

Example: *Amy is a sales representative with Acme, Inc., and is at a client's location pitching her products. The client has introduced a new requirement, integrating two technologies with Acme's solutions. Amy quickly does a search across the company for people with knowledge of the technologies in question. She gets twenty results, including three senior level technical architects and two that have actually worked with the client in the past. Amy holds her own, saying that there are experts in the company with those technologies, and we could certainly revisit and explore how it might come together. During the break, Amy chats with her colleagues to get more background, sends email to a few of them, and creates a group so she can refer to them at a later time. Amy is able to demonstrate responsiveness and strong ties to her company for the rest of the client visit.*

In 2004, IBM redesigned its internal corporate directory, "BluePages," to include extensive profiling information, updated search capabilities, real-time connection and common Web services. Some of the challenges included managing an increase in data (10-fold), a demanding world-wide target population, and a geographically-dispersed infrastructure. These are the somewhat typical issues architects face today, and the 2004 redesign proved several best-practice approaches to high performance Web application design.

CHALLENGES IN BUILDING HIGH PERFORMANCE WEB APPLICATIONS

Data-Driven Web Applications Drive Contention to the Data Source

Over the last few years, Web applications have moved from basic content presentation with a few dynamic elements to more advanced content personalization and collaboration capabilities. Regardless of where a given application lives on that spectrum, most aggregate data and every request to the Web application incurs at least one hit to a data source. This introduces some obvious opportunities to manage transactions, while ensuring that the contention to the data source is minimized. Among the many ways practitioners address this space are connection pooling, data source replication, and resetting expectations to redefine requirements.

Pooling

Connection pooling is the primary tool for ensuring fast transactions to data sources. Object pools are collections of already created and initialized objects (Grand, 2002). The pool often limits the number of possible objects to maintain resource predictability and force a level of resource management. Objects are created beforehand to eliminate the performance cost associated with creating and initializing the objects for use. In the case of a database connection pool, a series of database connection objects are created that maintain a connection to the database. This eliminates the need to create a connection or initialize any required parameters. Many application platforms provide database connection pooling as part of their design: however, object pools are fairly straightforward to implement and are required when there is no "out of the box" solution. In the end, connection pooling is all about making the transaction perform so well that any contention for a managed resource is minimized.

Replicate the Data Source

Enterprises often have guidance and standards for reuse and data mart creation and management. The goal is often to minimize redundant data in multiple places. The side affect of centralizing data is that many exploiting applications are requesting the same data source. If deployed properly, this is a suitable strategy: Each exploiter is given access to a specific number of connections, reducing one application's impact on another. An issue is still not resolved when databases are geographically separate from the exploiting application. Database connection pools and database connection tools usually connect to databases over a network protocol that carries all the risk of network latency and instability. One method of mitigating this variable is to replicate the data source to the same subnet as the application server (Pacitti & Simon, 2000). By collocating the data source with the Web application, the number of issues introduced by the network is greatly reduced. If there was an issue with contention or network performance, they are addressed. As with many of these solutions, contention can still exist (Abbadi & Toueg, 1989; Pacitti & Simon, 2000), but by addressing the performance of every link, contention is minimized.

Do Less, Simplify Views, Reduce Functionality

A requirements document is an architect's best friend. It clearly specifies both functional requirements (i.e., users need to be able to send pages from the Website) and non-functional requirements (i.e., needs to handle 200 concurrent users per second). Therefore, the client has a clear understanding of what he/she/it is asking and what the service provider is committing to deliver. There are times when requirements are accepted even with little knowledge of what will be required to fulfill the commitment. This is very common when the team either lacks the experience necessary to size appropriately or when it is breaking new ground. During these times, practitioners often return to their customers, presenting the issues of the moment and convincing everyone that the right thing to do is revisit the requirements document. If there is too much occurring in a given user scenario (impacting the ability to support a certain service level), the team must consider whether it should be doing less, simplifying view or reducing functionality. Sometimes this *really* is the only option; after all, ambitious customers ask for the world, and it is the service provider's job to help realize that vision as best he/she/it can.

Transaction-Based Systems Tend to Do Too Much

In the early days of the Web developer, applications were often created as Fast-CGIs hooked into Web servers, persistent in memory and performing functions on each request. The more promising applications often provided considerable value, but increased in complexity. Application design went through a period of linear programming and with the maturation of application models such as J2EE, developers were able to branch out into an object-oriented world where solutions were created by pulling together modular components. Unfortunately, the practitioners did not evolve at the same rate as the technology. All this is to say, developers of transaction-based systems often do too much in one area of the application, reducing flexibility and manageability.

The Servlet Controller

One of the common approaches to structure a Web application is to use a servlet controller (Alur et al., 2001), which handle all the requests into an application. They often perform some very basic request preparation and then forward the transaction to other servlets to perform specific tasks. Servlet controllers act as the gate keepers to the application, providing a single face to the end user while maintaining the flexibility and modularity of specialized servlets that handle only the function required at any one time.

STRUTS

One of the most popular implementations employing a servlet controller is STRUTS from the Apache Software Foundation (Apache Foundation, 2005). STRUTS is a very robust framework that allows Web applications to be built based on some of the best design patterns (Sun Microsystems, 2002). STRUTS includes several features that address the transition from a single bloated application to a highly modular system.

Deal with Slow Performance by Adding Servers /Over-Scale

There are times when a change to the requirements document is not an option. The next favorite solution is to scale the application horizontally. "We will just add more hardware" is what we often hear. This traditional approach usually means you have enough servers to handle peak loads, and during the lower traffic hours the application is over-provisioned. Adding hardware to solve an architectural or design problem is a tactical solution which only masks the problem. Adding

hardware often helps address load issues, but the increase in resource brings an increase in complexity. There is a balance to be found between budget, hardware capability, and application architecture and design.

Complex Applications Need Tender Loving Care

Even the simplest applications have a tendency to become complicated. Companies often have legacy systems, a variety of existing hardware, software and processes, and the integration task can therefore be daunting. Even the most skilled, well-funded projects have weak points. These Achilles heels often require constant monitoring.

Some systems integrate with in-shop or third-party monitoring systems, allowing systems administrators to be notified when issues arise. The basic checks include making a request to the application, and as long as there is a response, trouble goes unnoticed. Some systems are more proactive, looking for errors and response times. However, with the desire to mask errors or erratic behavior from end users, monitoring tools have a difficult time relying on the basic tests. The Web application may be performing fine, while the database is down.

Savvy developers build in special servlets that perform end-to-end tests of their Web application. These test servlets make a series of calls to ensure that all is right from the application's perspective. However, this level of development is often unfunded because the value it provides is seen only when there is a problem, and even then it simply quickens the time to resolution instead of eliminating it. Applications with a myriad of dependencies, network files stores, LDAP (Lightweight Directory Access Protocol) servers, databases, and Web services require monitoring as well. To an end user, an unresponsive or non-functioning application is as good as no application (Goodman et al., 2004).

The Platform and Plumbing Can Impact Perceived Performance

Managing end user perception is a key task in achieving high customer satisfaction. With the continual push for a service-oriented architecture and componentization, the dependency list keeps growing. Exposing services also means providing consistent, reliable service; impacting and exploiting applications in acceptable ways. One of the problems is that not everything is managed in the application architecture layer. Platform and network plumbing can have serious impacts to perceived performance.

Applications are often made up of many components and software stacks. Each component contributes to the overall application platform, and it is possible to combine modules that work better or worse. This fact often drives solution providers into single-silo approaches, embracing a single vendor or a specific recipe of vendors to deliver solutions. A more mature approach is to realize the need for standards-based solutions to ensure that components are easily mixed and matched without impacting the exact implementation of the application.

Network conditions have an obvious effect on applications, especially when a global audience is considered. A fast application in North America (.25 - .50 ms response time) can become slow in Asia Pacific (5-15 sec). Some of these challenges arise simply because of the amount of content being shuttled across networks. Some delay can be minimized through caching edge servers, better network backbones, and globally-distributed solutions.

Unexpected Errors Happen, but Who Needs to Know?

Application developers are told to handle errors with grace, but the definition of "grace" often varies in meaning and context (Nielsen, 2003; Preece, 1994; Raskin, 2000). If an application

supports a very small audience, it might be preferred not to spend a lot of time on exception handling. In most cases, though, applications are meant to benefit a larger population and, often, a demanding user.

The most basic approach to handling errors is to employ custom error pages, so that at least the error looks like it was intended to be read and responded to. This is the minimal amount of effort required to keep up appearances.

Some developers have resorted to irrational behavior to confuse the user. If an error occurs when submitting a form, some applications redirect the user to a different page, possibly unrelated, in the hopes of confusing or at least giving the appearance that the user might have actually been at fault. "When in doubt, redirect to the home page" is a motto no one should be fond of, but it happens.

More sophisticated developers include a clear and direct message acknowledging the error. Sometimes an alert is generated for the benefit of the systems team; other times, the Web applications present the message to stop any further complaint. These approaches often tailor errors with reasonably coherent messages. Instead of ERROR 500, it might read "We're sorry! The application is currently having difficulty handling your request."

APPROACHES TO BUILDING MORE RESILIENT WEB APPLICATIONS

Numerous challenges face application developers and architects developing scalable Web applications that also maintain performance and reliability. In the previous section, we covered five scenarios that often plague Web applications: data source contention; performing too many tasks; care and maintenance; platform and network capability; and exception handling. With each of these areas, we explored some of the popular approaches for addressing the problem space.

In this section we examine alternative approaches to these problems that have proven best practices in our own architecture and application design. We will cover:

- Caching strategies for high-availability Web applications: beyond storing copies of HTML (Challenger et al., 1998; Iyengar et al., 1997)
- Asynchronous task processing within Web applications: removing non-essential linear logic from high-volume transactions (Grand, 2002)
- Building self-reliant autonomous behavior: encapsulating through services achieving

Table 1. Sample non-functional requirements (NFR) for an enterprise application

	Target	Measured By
Scalability	Up to 250 requests per second	Application stability
Performance	- For 95% of the transactions < 7 seconds - For 5% transactions < 10 seconds	Probe tools measuring response times at different geographies
Availability	99.96% during schedules service hours	Tools measuring availability metrics

tight integration with *true* loose coupling (Birman et al., 2004)
- Client-side Model View Control (MVC): moving MVC to the browser supercharging the response times and getting a "wow" user experience (Murry, 2005; Sun Microsystems, 2002)
- Graceful degradation: keeping users thinking and feeling "fast," "reliable," and "always on" (Florins et al., 2004)

High-Availability Application Layer Caching

One of the challenges that busy Web applications and Web portals face is the need to perform data source queries to deliver features and functions to end users. As we reviewed earlier, this often drives contention to the data source. One overlooked approach is to design smart caching strategies to minimize calls to the data source. This aversion is well warranted, as the decision to cache often has implications in terms of high availability, data freshness, synchronization challenges, and real-time management. The following section reviews two strategies for caching: database-driven caching and more dynamic LDAP caching.

Database-Driven Caching

Typically, relational databases comprise a mix of static lookup-type tables and dynamic or updatable data sets. Each of these offer opportunities for application layer caching. To help provide context to these situations, we will use the IBM expertise location system as an example.

Static Table Data Cache

One of the popular aspects of an employee's profile is the contextual information which we display alongside the more central content. A drastic change from the previous version is the area of the profile that depicts organization structure. When this profile view was initially designed, the pages were getting response times around 850-1000 milliseconds and were driving, on average, six times more LDAP operations to our enterprise

Figure 1. Example of a BluePages profile view with reporting structure (right)

directory than the second most common function of viewing employee records.

While there are several factors contributing to the performance attributes, the primary one was the introducing of the reporting structure on every profile. To help address both of these undesirable characteristics, we designed a cache holding reporting structure information for all users, using a local database as the source. Obviously, there is huge performance gain by replacing three LDAP operations with three lookups in a local memory object. Reusing the application data source to build the data structure alleviated any additional work from the corporate LDAP directories.

Caching lookup tables and other relatively stable areas of the database is easy and reduces the workload on the database and the application. It is a simple way of maintaining the user experience while improving performance.

Dynamic Table Data Cache

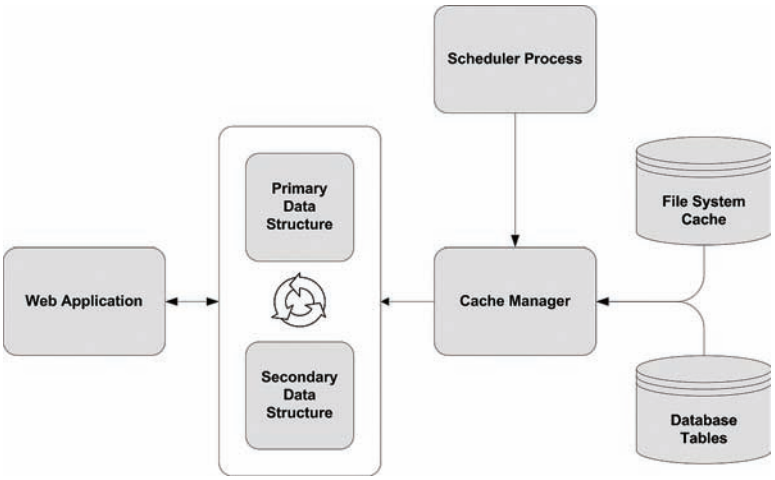
Static data is an easy and obvious caching opportunity, one that many architects and developers can easily apply to existing or new applications with very little investment. Obviously, things get challenging when caching occurs at the application layer, and the data has the possibility of being updated. Suddenly, maintaining appearances to an end user has to be well thought-out.

With the move to a modern Web application, IBM's expertise location exposed several common services and introduced an unexpected load. Caching at the service layer played a very important role in sustaining the growing popularity of service-oriented architecture and Web services.

In this type of cache, we need to deal with a table data which gets updated continuously by the Web application.

There are three key parts of this design (Figure 2): the scheduler, cache manager (Grand, 2002), and high-availability data structure. The scheduler initiates the cache manager and, in follow-on versions, also manages any message queues and events that trigger cache management features, such as changing the schedule for refresh. It performs most of the critical operations managing the cache data structures. Obviously, the cache manager is responsible for making calls to a Data Access Object (Alur et al., 2002) to load the

Figure 2. Highly-available data cache



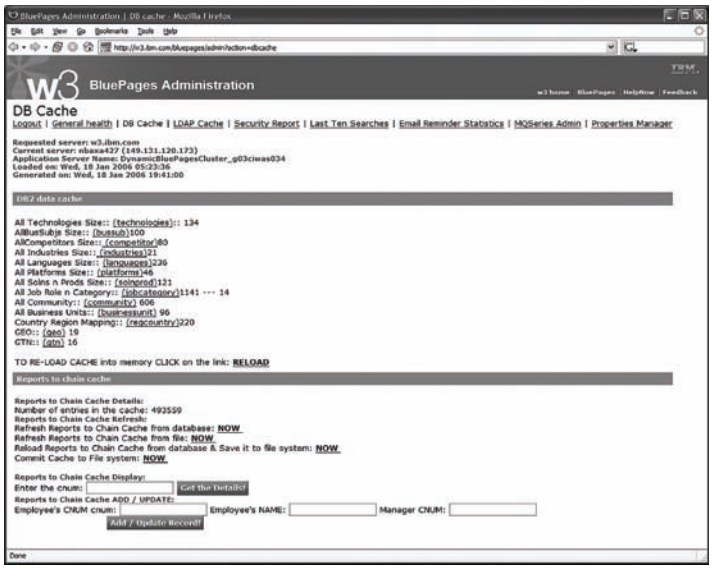
cache. This might be from the database or from the file system, depending on the configuration. The file system was used to eliminate start-up lag at risk of displaying stale data. For example, the reports-to chain cache (organizational) data object in BluePages is approximately 18MB size. It takes enough time that cycling the application produces undesired load and delay. By automatically committing the data structures in memory to disk, the cache manager can load instantly and update as needed.

The cache manager works with two data structures, the primary and secondary caches. Various data structures operate slightly differently and, often, some level of synchronization is required when performing manipulations. To eliminate contention of the data cache, a second one is created by the cache manager. It uses the second one to perform any management operations (add, update, delete) and sets a Boolean flag to immediately put it into play. The other cache can then be deleted or saved depending on the refresh strategy. For example, if the database is designed with time-stamps on changing data, deleting the cache is somewhat wasteful. There might be only five changes, and there is really no need to transfer all of the other data. By selecting only the items that changed and updating those in memory cache, updates can happen frequently and take effect more promptly with less resource churn.

One of the major undertakings for BluePages was the introduction of Web-based management for key subsystems (Figure 3). The cache management is one example of these Web-based management systems.

Web-based management of the cached object allows us to monitor the data that is in memory. For the reports-to chain cache, it shows data for an employee's reporting structure. It also allows an administrator to manage the data in memory (create, update, delete), commit those changes to the file system or database, and reload the cache, if necessary. Providing administrators with management capabilities to key subsystem

Figure 3. Data cache Web-based administration



components removes the specialists from having to perform more risky manual tasks in support of an application. It also allows the application to appear more responsive by supporting more real-time management instead of code changes or manual application restarts.

By using strategic caching, BluePages was able to see a reduction of 800 milliseconds in response time. Approximately 750 milliseconds were due to the reports-to chain cache and ~50 milliseconds were associated with database calls. Currently the same page delivers a 70ms server-side response time.

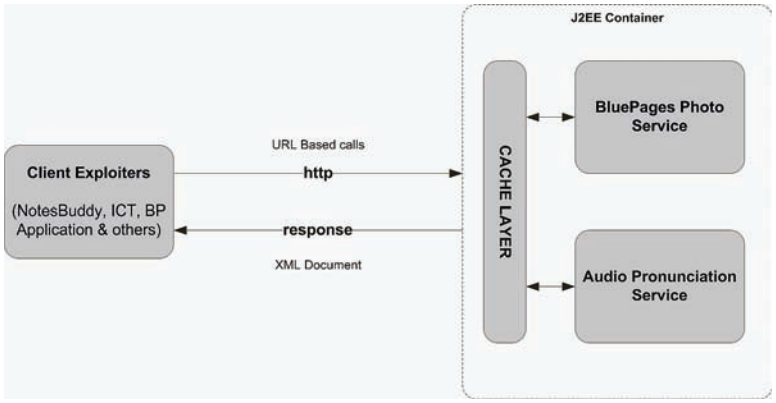
Another situation in which dynamic data caching plays an important role is the design of service implementations. Most services (Web services, XML-RCP, URL APIs, etc) take a considerable amount of overhead to process a request. For example, in Web services, a decent amount of XML is being manipulated; this in turn creates a considerable amount of garbage. Consider that overhead on top of the need to actually service the request by hitting data sources or performing calculations. BluePages receives approximately 3.2 million service calls per day and supports a response time of a few milliseconds for service exploiters. To achieve this level of efficiency, caching was employed to enable smarter transaction processing.

The key to smarter transaction processing is providing a proxy between the transaction request and the service logic, or process. The proxy determines if the request should pass into the process stage or if the request for whatever reason deserves a cached, static, or predefined response.

In the BluePages architecture, this "service cache layer" pre-fetches basic information related to the service and decides if it has enough information to respond. More specifically, BluePages offers a URL API to access a user's personal photo. The cache layer stores all the unique IDs and uses this information to short-circuit the request. If the request is for a user that does not have a photo, a redirect to a default image can be sent. The redirect is used instead of a full photo to keep the transaction light and to ensure that browsers get an opportunity to cache the default image for faster rendering.

In BluePages, this cache avoids over one million service calls from reaching the processing

Figure 4. Using caching to avoid unnecessary transactions



phase of the service. That translates into over one million database calls while improving overall response times.

Dynamic LDAP Caching

Sometimes caching all the data in RAM is just a waste of space. LDAP caching with some level of detail is probably not the best candidate to load into RAM. Presumably there is a high level of caching at the LDAP server, so it is best not to add another layer. Therefore, a cache is needed that builds based on usage, and some general rules about how memory can be reclaimed and which entries are always a "good idea" to have.

The typical reporting view includes a list of individuals related to the current profile. While one LDAP connection can be established with multiple queries executed, the transaction itself is still somewhat intensive. We think of the transaction time with LDAP as being very fast, and it is; but when you deal with so many transactions, general queuing theory and rendering speed tend to get the better of us. A read from RAM is simply much faster than a transaction to LDAP.

This view is supported with the lazy loading (Grand, 1999) dynamic LDAP cache. At least three levels of the chain already exist, and the subsequent entries are cached for future use.

The LDAP Cache Controller servlet acts as the central management point. It delivers the subsystem management views, and Web applications use a static method to access the Cache Manager (Grand, 2002). The Cache Manager is the object that populates the cache and fails over to LDAP as needed. It keeps track of metrics and performs any regular updates based on the Data Refresh Scheduler.

As with the other cache subsystem, the LDAP cache has a management interface. The main difference is the cache statistics view. With a database view, the hit rate is not relevant. In many cases, it is 100%. In the case of a dynamic cache where you are building it based on usage, hit rates are

Figure 5. BluePages reporting structure tab

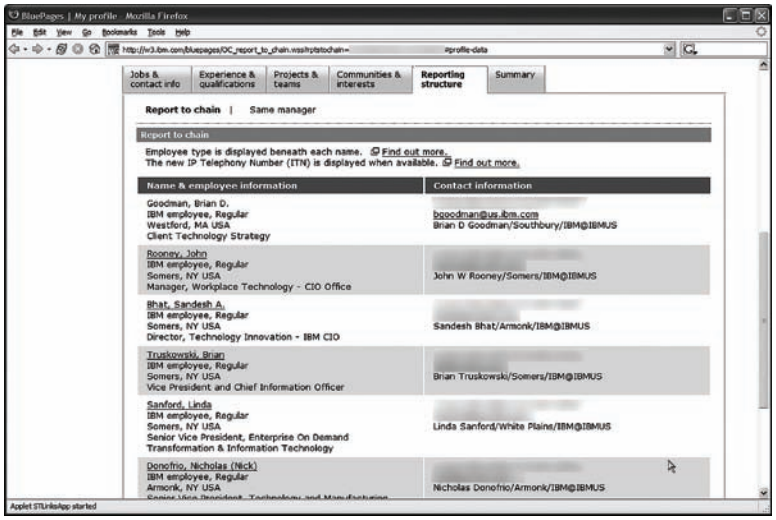


Figure 6. Dynamic LDAP cache

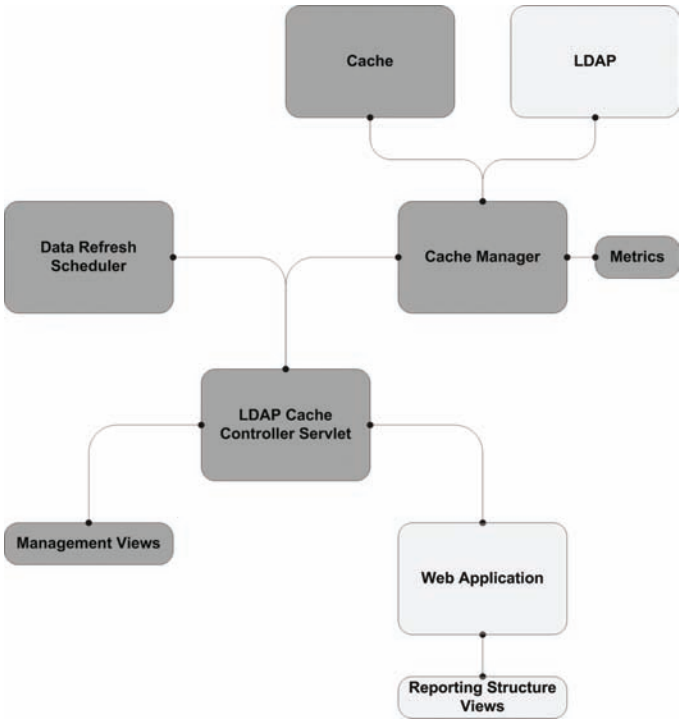
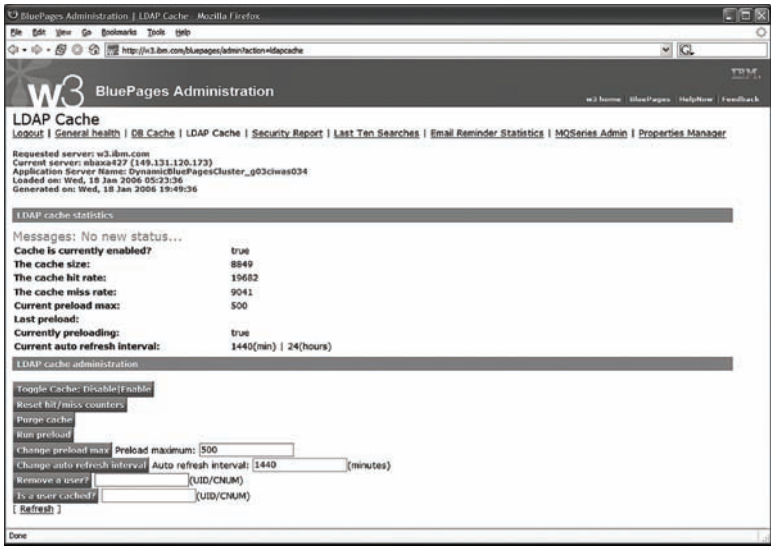


Figure 7. Dynamic LDAP cache Web administration



everything. When you consider that the only views driving this usage are organization structures (the third most accessed view), it is astounding that we experience an 80% hit rate.

In order to avoid the added traffic, caching was used. On some pages, we save up to 6 seconds—an eternity in Web time.

- Cache hit rate is 80% with a peak size of 80K entries stored
- Total LDAP call reduction by ~792K per day (240K queries vs. 48K queries)
- Avoiding up to ~6-10 LDAP operations and a response time of ~1000-1500 milliseconds on each organizational page

Asynchronous Task Processing

Making Transactions Non-Linear

One of the themes that this chapter raises is the notion that many operations can happen asynchronously (Grand, 2002). However, many architects and developers do not take the time to think about how to build their system to support it. Often implementing function wins out over responsible use of resources. A typical Web transaction is performed through a GET or a POST to a servlet or CGI. The servlet or CGI processes the request and returns a response. In many ways this is linear programming. If you use message

Figure 8. BluePages Web administration view of last ten searches

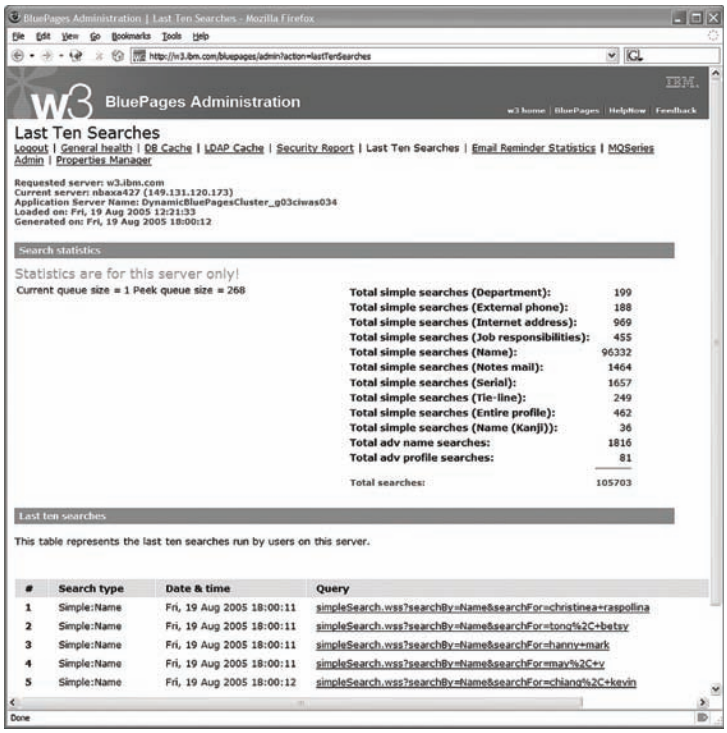
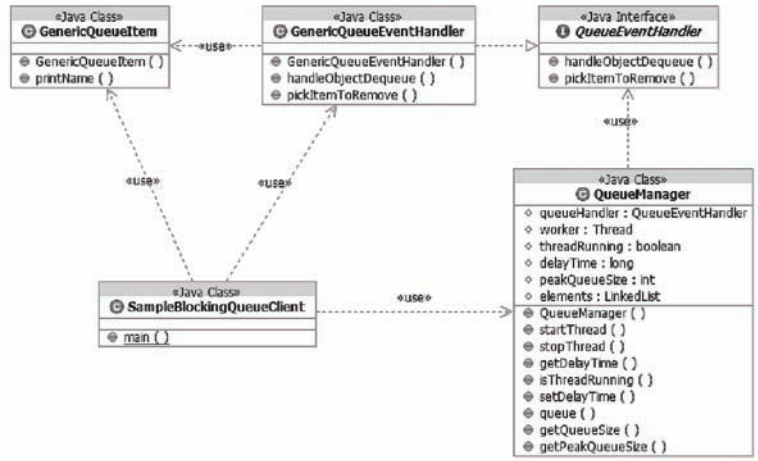


Figure 9. Blocking queue class diagram



queuing systems in your Web applications today, then you might already know the wonders of responding without necessarily completing the entire transaction. To help illustrate this design point, BluePages offers a couple of examples, all based on the same pattern.

One of the design criteria for the online administration page was to track operation statistics in real-time: Who were the last ten users to log-in (success/failure), what were the last ten searches, and so forth. One of the more costly aspects of collecting such performance statistics is that it takes cycles away from performing the user facing operations. The problem to solve was how to perform the tasks as part of a transaction without constantly having a locked state around a data structure, or processing requests while we should be sending back results.

The basic solution is based around a blocking queue design pattern (Grand, 2002; Kabutz, 2001) dequeueing only when there are objects present in the queue. This avoids spinning extra cycles with no operation. In the class diagram (Figure 9), SampleBlockingQueueClient is a client/component exploiter of QueueManager. QueueManager manages the thread and message brokering. QueueManager is initialized with a QueueEventHandler and uses that interface to interact with other implementations of handling the queue. For example, some queues need to be priority-sorted or dequeued based on some other criteria. The GenericQueueItem happens to be the object being enqueued and dequeued. It is a very simple and effective design pattern.

An example from BluePages illustrates the real-world use further. BluePages provides a keyword search to employee profiles. A user might search for other employees in the company who know about a certain client and a technology. The servlet processes the request and enqueues a value object to the queue with minimal amount of data (i.e., the query terms, the URL, the user ID, a time-stamp, etc.) and then respond with the search results. Asynchronously, a thread uses a QueueEventHandler object to decide which item in the queue should be dequeued and then passes

that object to get handled. The object performs any calculations (number of searches, list management, and clustering or trend-type calculation) and stores the data to be presented and used in other ways. An example of where this data can be used is in the popular, "what are other people searching for right now" capability. Even though the insert is synchronized (we used a Java ArrayList), they take more precedent because there are more enqueues than dequeues at any given time. You can see in the sample screen shot of the BluePages administration interface that the peak queue size for this specific server instance is 268, so we know the approach is working.

In some cases, a soft-queue is not sufficient, and a real message queue subsystem is required. The proposal of this section is simply to raise the issue that not all aspects of a transaction need to complete during the duration of handling a request. There are times when it is not encouraged, and specifically for J2EE, it is suggested to be incorrect. For example, when performing database transactions asynchronously, a transaction is preferred for all the security context concerns (Brown, 2005). Asynchronous beans helps alleviate some of this limitation, as discussed in the next section.

Asynchronous Beans in J2EE

Spinning new threads from a servlet for asynchronous task processing is a common design approach until recent times. Some of the drawbacks of this approach include: The transaction loses container security, transaction scope, and lack of failover during critical failures.

J2EE came up with a standard solution to solve this problem, in which the servlet request places the task information as a message in queue, and message-driven beans (MDB) read this message. A *message-driven bean* is an enterprise bean that allows J2EE applications to process messages asynchronously. It acts as a JMS message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component, an application client, another enterprise bean, or a Web component, or by a JMS application or system that does not use J2EE technology. MDBs process the task in an asynchronous way within the container, taking advantage of all the container's built-in features. As soon as the request message is processed, the MDB can place a response on the response queue. Servlet requests can keep polling the response queue for the status of the request submitted to let the end user know the results of the asynchronous task.

Business Grid Computing

Web application architectures often require batch processing of some of the maintenance tasks. It is not uncommon to have developers rewriting code with stand-alone applications or scripts to perform these operations outside of the Web container. This results in code redundancy and, obviously, such implementation loses the application server container advantages. Advanced J2EE containers such as IBM WebSphere Extended Deployment, allow asynchronous task processing by allowing batch, long-running or compute-intensive tasks to run within the container; the best application is chosen for this task based on resource utilization.

The business grid function in WebSphere Extended Deployment extends the WebSphere Application Server to accommodate applications that need to perform long-running work alongside transactional applications. Long-running work might take hours or even days to complete, and consumes large amounts of memory or processing power while it runs.

While IBM and other companies have advanced application platforms to help with the task of batch processing as an architect or application designer, it is important to ensure that your applications have a strategy to minimize the duplication of code and fragmentation.

Application Resiliency

As more applications are designed to participate in Service-Oriented Architectures, they increase their dependency on resources outside of their control. Applications make calls to external interfaces, such as databases, LDAP, messaging servers, SOAP-based Web service calls, or even HTTP-based services. Suddenly, the performance of an application depends on the external interfaces of other applications and services (Krishnamurthi & Bultan, 2005). The availability or poor performance of external services potentially impacts the availability of the dependent exploiting applications. This section will look at how applications can employ a more autonomic, resilient approach to these situations.

Human Required: The SOAP Fault Specification

For Web Services, W3C built the SOAP Fault (W3C, 2003) to represent the availability status of the service providers. The information provided by SOAP Fault is similar to the reason-phrase provided by HTTP (Internet Society, 1999) which underscores the nature of the problem (Table 2). One of the problems with these approaches is that system designers end up using generic issue codes, making it almost impossible to know what is really going wrong. Issues can be detected, but a human is still required. What if the dependant hosting environment has problems? This could result in Web application servers accepting requests, filling queues, and in turn affecting the entire infrastructure. This becomes even more unattractive if the hosting environment provides shared hosting or manages entry through proxy servers. A SOAP Fault indicates that an issue occurred with that transaction; however, it places the next step back onto the calling application, which may very well keep making service calls and impacting the overall application performance.

Communicating Availability

Transaction or session-specific errors often indicate that something is wrong, but a hung socket

Table 2. Example of SOAP fault

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:Body>
    <soap:Fault>
      <faultcode>-1000</faultcode>
   <faultstring>Database is unavailable.</faultstring>
   <detail/>
   </soap:Fault>
   </soap:Body>
   </soap:Envelope>
```

connection or a cascading error may never return a satisfactory message to allow an exploiting application to respond appropriately. In order to provide a higher level of resiliency, real-time status is required. UDDI servers can identify several end points, which may allow a Web service client the opportunity to use a different system to accomplish the same task. The problem is that any one of those services may be having problems. To help address this situation, system availability is included as part of the interaction between the service consumer, service discovery, and service provider.

This model (Figure 10) provides a level of service availability above and beyond the typical transaction-based error. Service clients can be advised of issues through simple calls to an independent third party, which allows appropriate logic to take place to maintain a higher level of service. This technique is similar to network dispatcher models, where a single service provides a feedback loop to ensure that servers in a cluster are available before being dispatched. For example, an asynchronous process may test the availability of an externally-dependant service through the central availability system. The response will provide information similar to typical faults; however, it is received outside of a transaction with the provider. This eliminates hung sockets or other variables that are only visible to the provider. Updates to availability status can be made from other enterprise monitoring tools, allowing for an integrated knowledge base of service availability.

In Figure 11, Enterprise Directory is a service provider. In this case, the service provider makes use of Tivoli Service Management to monitor its services. ED Monitor is one such service monitoring tool. BluePages monitors the health of the enterprise directory to failover or degrade service as necessary. Applications cannot perform in autonomic ways without being informed about

Figure 10. Service availability system for more predictable operations

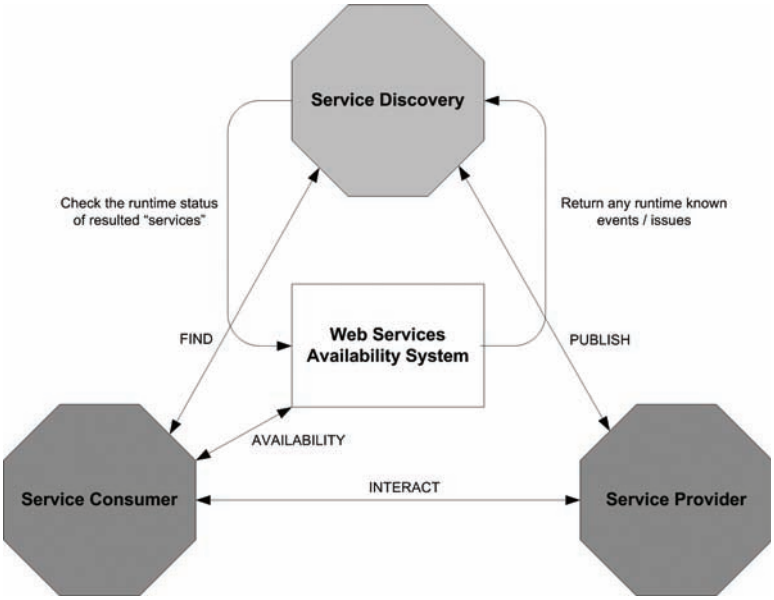
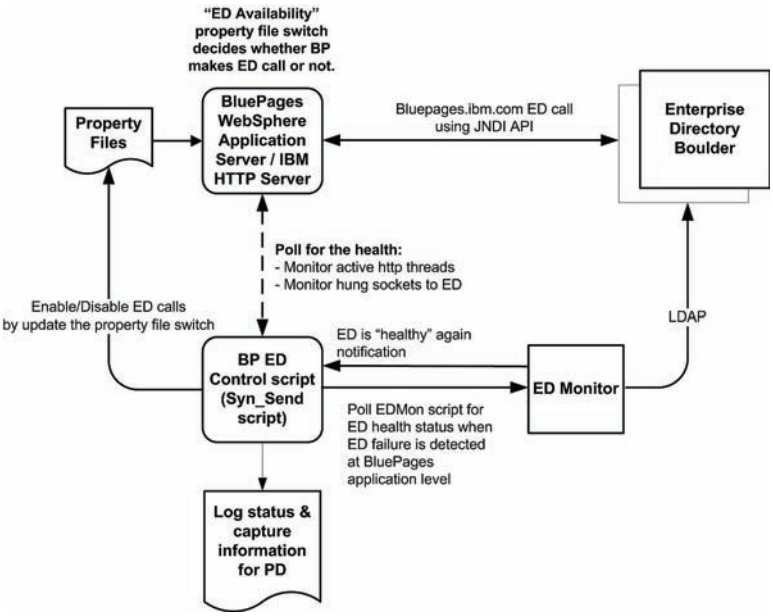


Figure 11. BluePages system checking availability of enterprise directory



their state, the state of the environment and, more importantly, the state of the systems and services on which it depends.

Client-Side Model View Control (MVC)

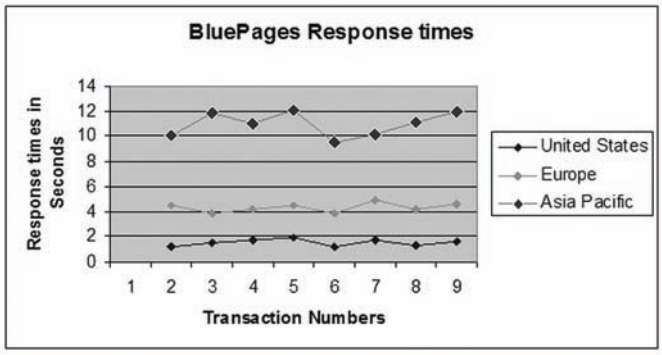
The Model View Control (Sun Microsystems, 2002) design pattern is a very popular approach to Web application design. Its one limitation is that the view is created server-side and must be transmitted to the client for interpretation. Moving MVC to the client truly separates the view from the middleware. The need for such activity is rooted in the desire for richer interactions and faster transactions. This section addresses client-side MVC and getting that "wow" user experience.

Common challenges for any enterprise Web application include internationalization and a variety of hosting issues that may impact responsiveness. While there are frameworks for addressing language translation among a diverse group of end users, the system deployment is often centrally located, resulting in poor performance for locations with limited bandwidth or congested links. BluePages is a real-world example of an enterprise application offering internationalization while being hosted from a single geography.

Figure 12 charts response times for the same page for Asia Pacific, Europe, and the United States. BluePages is deployed in the United States and, as the chart depicts, the performance of the application degrades as the end user's location moves further away.

The problem with the traditional (server-side) MVC design pattern is that it renders the entire view at the server, and makes the application and infrastructure components transmit the data to the Web clients. Web applications end up sending a

Figure 12. BluePages response time comparison



considerable amount of HTML back to the client Web browsers to achieve a rich user interface.

Consider users that rely on client-installed applications (VB-based, Lotus-based, or even Oracle forms), which query for information from a server. The amount of data transferred is the smallest possible, as it is no longer transmitting the view. The client-side applications display the data using the front-end logic held at the client. The client server model helps illustrate the benefits of a client-controlled view; however, it brings all the negative issues that drove the Web revolution. The ideal situation is a richer client side view that is managed via the Web.

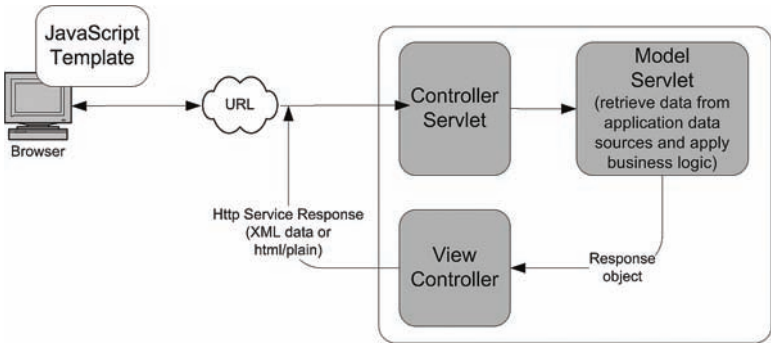
The decision regarding whether to use client-side MVC or server-side MVC may depends on several factors, including the kind of application, end user connectivity, or even how much data is being transferred to the client browser and what impact it has on the overall performance of the Web application. There are two approaches to client-side MVC: JavaScript Templates and the XMLHttpObject.

Client-Side MVC: Using JavaScript Templates

Using the JavaScript Template design pattern places all the rendering of the user interface into client-side cacheable assets. In this design pattern, the Controller Servlet invokes a Model Servlet. The Model code executes business logic, including actions such as retrieving information from database systems or other external services. The Model servlet passes values to the View Controller to render an XML document or HTML/plain text response. The View Controller decides which JavaScript template should be invoked for a particular service response. Once the response reaches the client browser, the JavaScript template specified by the View Controller renders the information retrieved by the service call. All the necessary HTML logic needed to display a rich GUI Web page is already built into the JavaScript templates.

This design pattern is dependent on how JavaScript files are cached by the Web browsers.

Figure 13. Client-side MVC, JavaScript templates



When a JavaScript file is requested through the browser, it is also accompanied by HTTP header directives that tell the browser how long the object can be considered fresh. Freshness translates to how long the resource can be retrieved directly from the browser cache instead of making calls to the originating end point. Since the browser represents the cache closest to the end user, it offers the maximum performance benefit whenever content can be stored there. Another consideration is that the JavaScript templates need to change infrequently to support a longer client-side timeout. Using this design pattern, BluePages shows a ten-fold improvement.

Client-Side MVC Using XMLHttpObject

Every modern Web browser has some form of XMLHttpObject that provides a socket connection to the originating server. This provides the opportunity to make subsequent asynchronous calls. Coupled with Dynamic HTML (DHTML), the user interface can be updated without a browser refresh. Avoiding the browser refresh provides a richer user experience because the context of the interaction is maintained. This is a popular approach and is often referred to as "Asynchronous JavaScript and XML" (AJAX) (Murry, 2005). The most popular examples are found from Google

Table 3. Sample search results response time comparison

Search Keyword	Number of Results	Server-Side MVC Design Response Times	Client-Side MVC Design Response Times
Smith, Chris	30	14.12	1.482
Linda	28	8.32	1.803
Kline	42	8.47	2.093
Kile	8	6.43	1.172
Inampudi	3	6.17	1.042

Figure 14. Client-Side MVC using XMLHttpObject

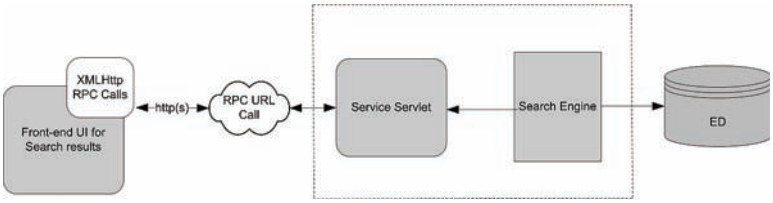
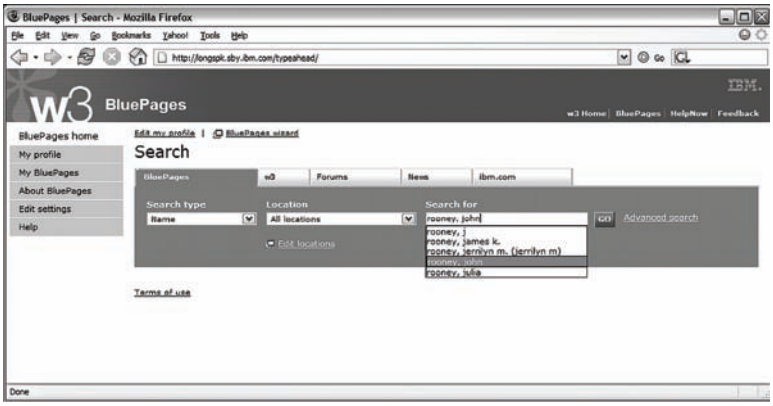


Figure 15. BluePages type ahead engine using XMLHttpObject



(GMail, Google Suggest, and Google Maps) (Google, 2005a, 2005b, 2005c).

In this design, pattern applications make asynchronous HTTP calls using XMLHttpObjects via JavaScript. A server-side service returns a response object. The response could be SOAP, XML, plain text, or JavaScript. The response text is then processed by the logic in JavaScript methods. Using DHTML, the content is displayed on the Web page.

Another use of this design pattern is depicted in Figure 15. As the user enters the employee's name for the search, the type-ahead engine makes a service call to retrieve matches from BluePages user dictionary. The list of potential names is displayed in the pull-down list based on the retrieved matches. As the user types more characters, the search is refined. In this scenario, it is important that the list retrieval is faster than the end user's typing speed.

Network Bandwidth Savings

Client-side MVC has another important advantage: network bandwidth savings. Since client-side MVC makes use of static templates to render the

Table 4. BluePages search performance using JavaScript template design pattern

ED Search	Current Implementation	BPv6 Implementation
Page size	77.9KB (Search on "kline" 25 results/page)	18.7KB (Search on "kline" 25 results/page)
Data transfer / day	77.9 GB	18.7 GB

user interface, the look and feel for an application can be cached in the client browser. The data that gets transferred to the client browser becomes much smaller.

While client-side MVC introduces some additional design complexity for Web applications, it can be worthwhile to achieve richer interfaces and more responsive Web experiences. There are sure to be many perspectives on the advantages and disadvantages of client-side MVC before it is universally accepted, but it has the potential to address some of the long-standing issues of Web application architecture.

Graceful Degradation

Most degrees in computer science and information technology do not require a class in human-computer interface design. It is not uncommon for an application to let the end user know that there was an error, for example, that it could not write to the memory and thus had to halt. The developer might have thought that the error would never occur, but even some of the common issue messages are unfriendly.

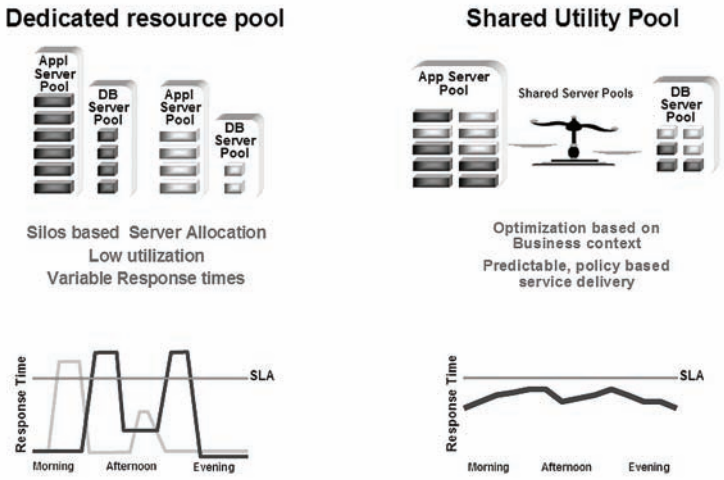
While there has been plenty of discussion over making exceptions appear informative and friendly (Nielson, 2000; Preece, 1994; Raskin, 2000), there is less focus around graceful degradation (Andoni et al., 2005; Florins et al., 2004; Herlihy & Wing, 1987). The driving principal is that an application can go beyond responding with friendly errors and actually disable features and functions until the environment stabilizes. Web applications in particular are vulnerable to an increasing number of dependencies: Web services, databases, LDAP directories, and file systems. Any single dependency can bring a Web application to a halt, and an unresponsive Web application might as well be a dead application. A simple method to address some of these issues is to place a timeout on the transaction, forcibly breaking the process to return an error to the end user. It is better to say something is broken than to be truly unavailable. This is really the first step. The second step is to prevent future access to the resource until a successful transaction indicates that the resource is again available. The challenge is to monitor your dependencies and code against the availability of those resources. If a database becomes unavailable, a backup might be used or menu items might be disabled, with ample indication to the end user that some of the features to which they may be accustomed have been disabled temporarily.

The BluePages Web application has many dependencies on external resources, and this provides an excellent real-world example. As part of the application, a service interface exposes read access to a back-end data store. Simple requests

to the service return a person object in the form of a DSML XML document. It happens that the full-text search engine requires XML documents to be written on a file system. This allows the file system to be the primary data source with the database as a backup. The added resources and processing required for reading a record from the database and returning an XML document are less preferable than serving up a static XML document. A simple servlet proxies requests to the appropriate data handlers by referencing a Boolean flag. This flag is used as an availability indicator of the primary data source. This indicator is set in two ways. First, any transaction which fails accessing the file system will set the flag so subsequent calls are routed to the secondary method. Second, an asynchronous process periodically checks the primary resource and proactively sets the flag in the hope that it will catch the issue before a failing request does. Additionally, this process sets the flag back when its periodic test no longer fails. From an exploiter's perspective, the service rarely fails. At most, there is a slight slowdown in performance, which is undesirable, but better than failing altogether. Adding a proxy layer between the requestor and the actual data access method lets the application short-circuit a potentially grave situation early, preserving the rest of the application.

Another example includes a typical dependency on a database or LDAP directory. BluePages provides both directory lookup and free text search. The truth is that the free text search can provide similar search results to that of the directory query, but the results are not exactly the same. Based on the previous example, it is easy to imagine the monitoring system provided in the application. The difference here is that if there is a problem with the LDAP resource, the current deployment has no effective way for failover. Instead of simply sending an error or notifying the user that the LDAP servers are causing a problem (not exactly something the average user understands), a subsequent query is automatically performed using the free text search. An unresponsive directory message might imply that the

Figure 16. Dedicated resources vs. shared utility



Web application itself can no longer perform any useful function. In our case this is not accurate, but end users should not have to figure this out. The search results are not perfect, but often they will suffice. Moreover, it is a single step to an end user; they cannot use the application improperly. Even if they do not see the general message that some features are disabled, the query is handled and results are provided as best as possible. The results are labeled appropriately, and the error messages are well communicated, while the user maintains access to the information. In this case we alter the behavior of the application slightly to support a more satisfying application experience. Failure is communicated, and next steps are clearly provided.

The principal of graceful degradation means that, as an application begins to fail, it does so in a way that is not jarring to the end users or other components or subsystems. If a single feature is broken, disable the components that rely on that feature, provide a method for resuming work, or failover to a backup method. There are times where budget, design limitations, or schedule restrict the ability to implement applications in this way, but the benefits are quite significant. Applications that gracefully degrade and re-enable as possible are easier to manage and maintain. They change the mode of application instability to monitoring, instead of scrambling to reboot and provide service. In some ways, we are talking about building smarter systems. At the heart of any autonomy is a core system that can indicate for it and others that a problem exists.

FUTURE TRENDS

Just as software design and architecture continue to evolve, underlying infrastructure and middleware continue to provide an ecosystem within which applications can execute. Key to maximizing future infrastructure advances is the ability for an application to tolerate and to exploit many of these advances. Recently, autonomic computing technologies have advanced to the point where grid-like technologies can be delivered, without the need for disruptive approaches to be embedded within application designs. By having a middleware environment that provides the needed abstraction layer around the underlying physical deployment topology, applications can be managed against a set of predetermined policies. These policies, while enforcing against a breach of their predefined goals, will provision and de-provision instances of an application, all without having the application or pre-mediation of the event.

Given today's large dynamically-configurable operating systems/hardware, applications should provide ample opportunities for the hypervisor to capitalize on parallelism. Key to this ability is having applications designed with clear units of work spawning threads which are then separately managed and driven to their logical conclusion. Just as loosely coupled applications need to understand their dependencies on downstream processing, highly threaded applications need to understand and take steps to ensure that they provide exception handling, for the non-serialized processing that is likely to occur.

In focusing on the maturing of the infrastructure and middleware, a key strategy within the industry is to deliver computing resources as a dependable, scalable, and transparent commodity which applications exploit through standard development and interface models. This strategy is squarely focused at delivery of the infrastructure as a utility.

Utility Computing

Utility computing is a provisioning paradigm in which a service provider makes computing resources available to a consumer as needed. Infrastructure management and the ability to charge for specific usage, rather than the traditional fixed rate model, are central to the value proposition.

The utility computing vision makes several promises revolving around reducing IT complexity, pricing based on usage and decreasing operating, and capital costs to the enterprise. Adapting an infrastructure to a utility computing model is not simply about making changes to IT infrastructure; people and business process changes are also critical for future success (Whatis.com, 2005)

A sub-strategy of utility computing is directly associated with consolidation. In a shared utility computing model, a service provider looks to centralize its computing resources to serve a large audience reducing overhead while maintaining business resiliency. The value proposition for utility computing can be difficult to express, but the following points to help articulate it:

- **Efficiency yields cost savings:** In accordance with industry observations, the average machine utilization of datacenter systems is between 15 and 25 percent (Coulter, 2005).
- **Abstraction and virtualization yields flexibility:** A dynamic infrastructure allows the environment to proactively respond to changing application loads (Chase, Anderson, Thakar, Vahdat, & Doyle, 2001). Through virtualization, transparent provisioning of system resources is performed against a constantly reevaluated set of policies. As load decreases and the likelihood of a policy breach is diminished, systems are de-provisioned and reallocated to other services. This increases the overall utilization of assets and saves money through less upfront infrastructure.
- **Standardizing the infrastructure stack speeds implementation:** By having a utility computing mindset, individual applications are freed from the overhead of having to procure, setup, and test dedicated infrastructure, which typically has long lead times and capital expenditure associated with it. This process is typically repeated numerous times within an enterprise portfolio, and little if any economies of scale are leveraged. By driving standardization into the infrastructure stack and a predictable set of services, applications can execute against a set of assumptions and deploy into a shared resource pool without unique and dedicated resources.

In order to support this utility computing services model, many fundamental infrastructure services are needed, such as the ability to meter, bill, provision, and de-provision the infrastructure. Niche industries will focus on these core services with much of the discussion around virtualization centers on increasing the utilization of assets. Standardization is key to a successful business in utility computing.

As the industry continues to move toward the utility delivery model, application architectures will need to evolve to a platform agnostic perspective where the MIPS and GIGs are of interest, rather than which level of operating system or service pack exists. Additional planning and specification will be required to externalize how an application is to be scaled, depending on the real-time operating characteristics.

Today's visualization tools allow the management of entire resource pools and proactively manage its utilization (IBM, 2005b). Virtualization is a journey in which applications and middleware are abstracted from progressively greater layers of the machine physical characteristics.

In a utility computing environment characteristic, workloads are highly distributed; applications instances must be allocated and de-allocated frequently and with minimal overhead and latency. Without this on-demand provisioning scheme, it will be difficult to maximize the efficiency of the core infrastructure. Currently, low utilization rates continue as a result of massive infrastructure build-outs. These build-outs are a direct result of static, fixed computing limitations and, quite

frankly, over-funded approaches to guarantee application availability; maintaining high utilization rates in all resources is critical if utility providers are to be profitable. Understanding how utility computing impacts their work is critical to architects, developers, and strategists catching the wave the first time around.

CONCLUSION

Every enterprise application has special considerations to achieve functional and non-functional requirements. As practitioners, we all look for a common platform to help alleviate some of our woes. It is not uncommon for people to say, "We will just add more servers," or "CPUs are getting faster and RAM is getting cheaper." It is true that application platform and server management can contribute substantially to success. The key to successful Web applications is the approach taken to application design and architecture while providing the necessary abstraction layers to allow exploitation by an ever-maturing systems management set of technologies. A well-designed application will thrive in future computing environments, but no amount of autonomics will efficiently deliver a poorly-designed application.

This chapter reviewed five common challenges in enterprise Web application design: resource contention, managing transactions, application resiliency, geographically-diverse and exception/perception management. Using our experiences with IBM's expertise location system, we presented five key methods:

- High-availability application layer caching
- Asynchronous task processing
- Self-reliant, autonomous behaviors
- Client-side MVC
- Graceful degradation

Finally, looking at trends that will continue to change and impact enterprise application design and architecture offers some insight to upcoming challenges. While the approaches in this chapter are somewhat generic, they are by no means the only way to address these issues. However, over the last 18 months, we have witnessed their immense contribution to the resilience, scalability, and overall performance from transaction time to network throughput and bandwidth usage.

REFERENCES

Abbadi, A. E., & Toueg, S. (1989, June). Maintaining availability in partitioned replicated databases. *ACM Trans. Database Syst., 14*(2), 264-290.

Alur, D., Crupi, J., & Malks, D. (2001). *Core J2EE patterns: Best practices and design strategies, pp. 172-407.* Palo Alto, CA: Prentice Hall.

Amiri, K., Park, S., & Tewari, R. (2002, November 4-9). A self-managing data cache for edge-of-network Web applications. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management, CIKM 2002,* McLean, VA (pp. 177-185). New York: ACM Press..

Andoni, A., & Staddon, J. (2005, June 5-8). Graceful service degradation (or, how to know your payment is late). In *Proceedings of the 6th ACM Conference on Electronic Commerce, EC 2005,* Vancouver, BC, Canada (pp. 9-18). New York: ACM Press.

Apache Foundation (2005). *The Apache Struts Web Application Framework.* Retrieved September 10, 2005, from http://struts.apache.org/

Appel, A., Dhadwal, A., & Pietraszek, W. E. (2003). More bang for the IT buck. *The McKinsey Quarterly, 2,* 130-141.

AskMeCorp Inc. (2005). *Knowledge Management Software and Services | AskMe.* Retrieved June 7, 2005, from http://www.askmecorp.com

Autonomy Inc. (2005). *Autonomy.* Retrieved June 7, 2005, from http://www.autonomy.com

Bates, M., Davis, K., & Haynes, D. (2003). Re-inventing IT services. *The McKinsey Quarterly, 2*, 144-153.

Birman, K., van Renesse, R., & Vogels, W. (2004, May 23-28). Adding high availability and autonomic behavior to Web services. In *Proceedings of the 26th International Conference on Software Engineering* (pp. 17-26). Washington, DC: IEEE Computer Society.

Brown, K. (2005). Asynchronous queries in J2EE. *JavaRanch*. Retrieved from http://javaranch.com/newsletter/200403/AsynchronousProcessingFromServlets.html

Burns, R. C., Rees, R. M., & Long, D. D .E. (2001). Efficient data distribution in a Web server farm. *Internet Computing, IEEE, 5*(4), 56-65.

Candan, K. S., Li, W., Luo, Q., Hsiung, W., & Agrawal, D. (2001, May 21-24). Enabling dynamic content caching for database-driven Web sites. In T. Sellis (Ed.), In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA (pp. 532-543). New York: ACM Press.

Challenger, J., Dantzig, P., & Iyengar, A. (1998). A scalable and highly available system for serving dynamic data at frequently accessed Web sites. In *Proceedings of ACM/IEEE Supercomputing 98 (SC 98).* New York: ACM Press.

Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., & Doyle, R. P. (2001, October 21-24). Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, SOSP '01*, Banff, Alberta, Canada (pp. 103-116). New York: ACM Press.

Coulter, T. (2005). *Utility computing reality.* Retrieved September 10, 2005, from http://www.line56.com/articles/default.asp?ArticleID=6467

Davison, B. D. (2001). A Web caching primer. *Internet Computing, IEEE, 5*(4) 38-45, Jul/Aug 2001

Eckstein, R. (Ed.). (2003). *Java enterprise best practices.* Sebastopol: O'Reilly & Associates, Inc.

Entopia (2005). *Entopia—solutions for information discovery.* Retrieved June 7, 2005, from http://www.entopia.com/

Florins, M., & Vanderdonckt, J. (2004, January 13-16). Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI '04)*, Funchal, Madeira, Portugal (pp. 140-147). New York: ACM Press.

Fowler, M. (2000). *Refactoring: Improving the design of existing code.* Upper Saddle River, NJ: Addison-Wesley.

Goodman, B. (2002). *Accelerate your Web services with caching: Drive your solutions into the fast lane.* Retrieved June 7, 2005, from http://www-106.ibm.com/developerworks/webservices/library/ws-cach1/?ca=degrL19wscaching

Goodman, B., & Kebinger, J. (2004). *Increase stability and responsiveness by short-circuiting code: Keep your Web applications running when tasks lock up.* IBM Developer Works. Retrieved June 7, 2005, from http://www-106.ibm.com/developerworks/web/library/wa-shortcir/?ca=dgr-lnxw09ShortCircuit

Google (2005a). *Google Gmail Beta*. Retrieved September 10, 2005, from http://mail.google.com/

Google (2005b). *Google Maps Beta*. Retrieved September 10, 2005, from http://maps.google.com

Google (2005c). *Google Suggest Beta*. Retrieved September 10, 2005, from http://maps.google.com

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns.* Upper Saddle River, NJ: Addison-Wesley Professional.

Grand, M. (1999). *Patterns in Java, (vol. 2): Lazy initialization* (pp. 233-237). New York: John Wiley and Sons, Inc.

Grand, M. (2002). *Java enterprise design patterns, pp 159-500*. New York: John Wiley and Sons, Inc.

Gurguis, S. A., & Zeid, A. (2005, May 21-21). Towards autonomic Web services: Achieving self-healing using Web services. In *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software (DEAS '05)*, St. Louis, MO (pp. 1-5). New York: ACM Press.

Hammar Cloyd, M. (2001). Designing user-centered Web applications in Web time. *IEEE Software, 18*(1), 62-69.

Hassan, A. E., & Holt, R. C. (2002, May 19-25). Architecture recovery of Web applications. In *Proceedings of the 24th International Conference on Software Engineering, ICSE '02*, Orlando, FL (pp. 349-359). New York: ACM Press.

Herlihy, M. P., & Wing, J. M. (1987, August 10-12). Specifying graceful degradation in distributed systems. In F. B. Schneider (Ed.), In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, Vancouver, BC, Canada, (pp. 167-177). New York: ACM Press.

IBM. (2005a). *IBM Software: WebSphere Extended Deployment.* Retrieved September 10, 2005, from http://www.ibm.com/software/webservers/appserv/extend

IBM. (2005b). *IBM Tivoli Provisioning Manager—Product overview.* Retrieved September 10, 2005, from http://www.ibm.com/software/tivoli/products/prov-mgr/

Internet Society (1999). *Hypertext Transfer Protocol—HTTP/1.1.* Retrieved September 10, 2005, from http://www.w3.org/Protocols/rfc2616/rfc2616.html

Iyengar, A., & Challenger, J. (1997). Improving Web server performance by caching dynamic data. In *Proceedings of the Usenix Symposium of Internet Technology and Systems*, Usenix Association: Berkeley, CA (pp. 49-60).

Iyengar, A., Challenger, J., Dias, D., & Dantzig, P. (2000). High performance Web site design techniques. *IEEE Internet Computing, 4*(2), 17-26.

Jadav, D., & Gupta, M. (1997). Caching of large database objects in Web servers. In *Proceedings of the Seventh International Workshop on Research Issues in Data Engineering, April 7-8, 1997*, (pp.10-19).

Kabutz, H. (2001). *Blocking queue. The Java™ Specialists' Newsletter* (016). Retrieved September 10, 2005, from http://www.enterprisedeveloper.com/developers/javaspecialist/Issue016.html

Kaplan, J., Loffler, M., & Roberts, R. (2004). Managing next-generation IT infrastructure. *McKinsey on IT, 3.*

Kephart, J. O. (2005). Research challenges of autonomic computing. In *Proceedings of the 27th International Conference on Software Engineering, ICSE '05, St. Louis, MO, May 15-21, 2005* (pp. 15-22). New York: ACM Press

Krishnamurthi, S., & Bultan, T. (2005). Discussion summary: Characteristics of Web services and their impact on testing, analysis, and verifi-

cation. *SIGSOFT Softw. Eng. Notes, 30*(1) (Jan. 2005), 5.

Liebmann, E., & Dustdar, S. (2004). Adaptive data dissemination and caching for edge service architectures built with the J2EE. In *Proceedings of the 2004 ACM Symposium on Applied Computing, SAC '04, Nicosia, Cyprus, March 14-17, 2004* (pp. 1717-1724). New York: ACM Press.

Liu, Y., Ngu, A. H., & Zeng, L. Z. (2004). QoS computation and policing in dynamic Web service selection. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, WWW Alt. '04, New York, NY, May 19-21, 2004* (pp. 66-73). New York: ACM Press.

Malveau, R., & Mowbray, T. J. (2001). *Software architect bootcamp.* Upper Saddle River, NJ: Addison-Wesley.

Murry, G. (2005). *Asynchronous JavaScript Technology and XML (AJAX) with Java 2 Platform, Enterprise Edition*. Retrieved September 10, 2005, from http://java.sun.com/developer/technicalArticles/J2EE/AJAX

Neville, S (2003). Emerging standards and futures in enterprise integration. *Patterns and Best Practices for Enterprises.* Retrieved September 10, 2005, from http://enterpriseintegrationpatterns.com/Future.html

Nielsen, J. (2000). *Designing Web usability.* Indianapolis, IN: New Riders Publishing.

Pacitti, E., & Simon, E. (2000). Update propagation strategies to improve freshness in lazy master replicated databases. *The VLDB Journal, 8*(3-4) (Feb. 2000), 305-318.

Paulson, L. D. (2005). Building rich Web applications with Ajax. *Computer, 38*(10) (Oct. 2005), 14-17.

Preece, J. (1994). *Human computer interaction.* (pp. 159-163) Reading, MA: Addison Wesley.

Raskin, J. (2000). *The humane interface.* (pp. 178-183) Reading, MA: Addison Wesley.

Reynal, M. (2005). A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News, 36*(1) (Mar. 2005), 53-70.

Rodriguez, P., Spanner, C., & Biersack, E. W. (2001). Analysis of Web caching architectures: Hierarchical and distributed caching. *IEEE/ACM Transactions on Networking,* 9(4) (Aug. 2001) 404-418.

Schmid, H.A., & Rossi, G. (2004). Modeling and designing processes in e-commerce applications. *IEEE Internet Computing, 8*(1), 19- 27.

Stelting, S., & Maassen, O. (2002). *Applied Java patterns.* Palo Alto, CA: Prentice Hall.

Strong, P. (2005). Enterprise grid computing. *ACM Queue, 3*(6), 50-59.

Sun Microsystems (2002). *Designing Enterprise Applications with the J2EE™ Platform* (2nd ed.), *Web-Tier MVC Controller Design*. Retrieved September 10, 2005, from http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html

Tacit (2005). Connect the dots to make the right connections happen. *Tacit.* Retrieved June 7, 2005, from http://www.tacit.com

W3C (2003). *SOAP Version 1.2 Part 1: Messaging Framework: SOAPFault.* Retrieved September 10, 2005, from http://www.w3.org/TR/soap12-part1/#soapfault

Weiss, A. (2005). WebOS: Say goodbye to desktop applications. *netWorker, 9*(4) (Dec. 2005), 18-26.

Whatis.com (2005). General computing terms: Utility computing. *TechTarget.* Retrieved September 8, 2005, from http://whatis.techtarget.com/definition/0,,sid9_gci904539,00.html

Zhang, J., & Buy, U. (2003). A framework for the efficient production of Web applications. In *Proceedings of the Eighth IEEE International Symposium on Computers and Communication*, ISCC, 2003. *30*(1) (pp. 419- 424).

This work was previously published in Architecture of Reliable Web Applications Software, edited by M. Radaideh & H. Al-Ameed, pp. 112-146, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 1.28
A Survey of Development Methods for Semantic Web Service Systems

Terje Wahl
Norwegian University of Science and Technology, Norway

Guttorm Sindre
Norwegian University of Science and Technology, Norway

ABSTRACT

Semantic web services (SWS) hold the promise of enabling dynamic discovery of candidate web services fitting a particular specified need. One interesting question is what impact this will have on software and systems engineering methods – will mainstream methods like RUP still be suitable, or will new or adapted methods be needed? This article surveys the state-of-the-art in methods specifically tailored for the engineering of SWS systems, looking at development methods trying to cover the entire lifecycle as well as methods covering only one or two phases. Some of the surveyed methods are specifically meant to deal with semantics, others are for the engineering of service-oriented systems in general. The survey reveals that there are many proposals being made in this area, some extensions of mainstream methods like RUP, others more experimental. [Article copies are available for purchase from InfoSci-on-Demand.com]

INTRODUCTION

During recent years, the software industry has increasingly acknowledged that using discrete building blocks of software provides an effective way of engineering such systems (Stojanovic & Dahanayake, 2005). The 1990's saw an increased use of component based development (CBD), but this is now giving way to the increasingly popular technologies of Web Services and Service Oriented Architectures (SOA) (Stojanovic & Dahanayake, 2005). In the future, Semantic

Web Services (SWS) may provide an even more powerful way of rapidly building flexible and advanced Service Oriented Systems (SOS).

Technology versus Development Methods

When information systems are developed based on some specific technology, this implies the use of some underlying basic concepts such as *service, requester,* and *provider,* and a certain way of thinking about the structure and organization of the information systems. Hence, it is important that the system engineering methods match the underlying technology, to be able to create systems of high quality in a more effective manner. To effectively develop or adapt methods for engineering of SWS systems, there is a need for understanding what special concerns arise when engineering these systems. New engineering methods or adaptations of general methods may be desirable if special concerns are to be taken into consideration.

The service concept elevates the abstraction level and aims at empowering business users to create and execute software systems based on SOA (Erl, 2005; Stojanovic *et al*., 2004). Many technology standards for SOS and Web Services-systems have emerged, but little attention has been given to methods for modeling and designing such systems (Cabral *et al*., 2004). The amount of new technology is growing and it changes over time, making development complex and error-prone. As a result, increased application adaptability, widespread reuse and commercialization is slowed without proper support from methods and development tools (Cabral et al., 2004).

Motivation and Purpose

This article surveys the current state-of-the-art in engineering methods specifically tailored for the requirements specification, modeling, design and development of SWS systems. Methods may also be in the form of specially designed tools for supporting the development of SWS systems. The motivation for providing such a survey in terms of its utility for the reader is as follows:

- Giving an overview of methodological differences that have been envisioned between SWS systems development and current mainstream software development.
- Giving an overview of, and some comparison of, different methods and techniques that have been proposed to deal with the specific challenges of developing SWS systems, so that the reader can more easily navigate the plethora of existing proposals and possibly find a method which is useful in the reader's own context (whether this be research or practice).

We try to analyze what makes the development of SWS systems different and unique compared to the development of other kinds of information systems. From the survey we also try to identify what special concerns the methods take into consideration, and compare these concerns in the analysis. We perform a simple classification and comparison of the different methods and tools. In the end, we identify some suggested areas for future research into methods and tools supporting the engineering of SWS-based systems.

Scope

This article focuses on requirements specification, modeling, design / development, and testing / evaluation methods for creating SWS-based systems for end-users. It does not focus on methods for developing new technologies or frameworks for such systems, but rather attempts to look at what becomes important if assuming that the proper technical infrastructure is in place. As a result, we are focused on methods that involve users, with more or less technical skills, to see what systematic approach, actions, processes and

tools are required to create a system for end-users based on an existing infrastructure framework.

Engineering of SWS systems may be seen from two perspectives: Engineering of individual SWS (as a service provider), or engineering of SOS utilizing SWS (SWS-based systems, as a service requester). When building the latter kinds of systems, there is often a need for simultaneously developing services - not only utilizing existing ones. This article tries to cover both these perspectives. On the other hand, the design, development and utilization of general domain ontologies are important aspects of developing SWS systems, but are considered beyond the scope of this article.

Many methods for engineering of information systems in general or SOS in general, e.g., (Stojanovic et al., 2004) might be useful for the engineering of SWS systems. This might especially be true for methods supporting modeling and elicitation of requirements, such as interviewing, workshops, etc. These methods are however considered beyond the scope of this article.

Research Questions and Methods

The research questions of this study are as follows: 1) What methods have been proposed specifically for the development of semantic web service systems? 2) How mature are these methods, e.g., in terms of empirical evaluations and tool support?

In the framework of (Glass *et al.*, 2004) the research method of this article would be classified as Literature Review / Analysis. In a more recent taxonomy by (Mora *et al.*, 2008) where IS research is divided into four quadrants according to the dimensions conceptual / empirical and behavioral / design, this article would fall into the category conceptual behavioral research, as it does not design anything new, rather reviewing existing proposals, and this review is on a conceptual level rather than performing empirical comparisons of various software engineering methods. As discussed by (Kitchenham, 2004) there are two types of reviews, the *traditional* / *conventional* and the *systematic*. Systematic reviews have several advantages in terms of the thoroughness of procedure and confidence in the final result, but also disadvantages in a much heavier workload due to its formality, therefore requiring much longer time. The review reported in this article would not fully satisfy all of the strict criteria for a systematic review as formulated by Kitchenham, but has many aspects of a systematic review in that it makes an unbiased analysis of a large collection of works rather than considering a more limited and biased collection of sources.

The literature search for this study was performed by searching in Thomson ISI, IEEE and ACM Digital Libraries, and Google Scholar with phrases “web service” and “service-oriented” combined with development stage terms such as “analysis”, “requirements”, “design”, “development”, “composition”, and “testing”. The sources that emerged from this search were quickly evaluated by reading the abstract or browsing through the full paper (if available) to determine its relevance to our study. The most relevant ones were then selected for further analysis, the inclusion/exclusion criterion being whether the paper really proposed or discussed a *method* for service-oriented systems development (rather than merely discussing advantages of service-orientation or proposing components or architecture of such systems). The reference list of found papers was also used to harvest further papers that for some reason did not pop up in the initial search.

Organization of this Article

Section 2 presents related work and explains how the current article is different from previous surveys in the field. Section 3 discusses the special challenges in the engineering of SWS systems, emphasizing how this is different from more traditional software development, and motivating why specific methods are needed although mainstream

development methods can partially be used or adapted to this purpose. Section 4 presents the surveyed methods and tools, which are compared and discussed in section 5. Section 6 concludes and suggests further research directions.

RELATED WORK

Some work has been done on surveying the state of the art in SWS technologies and frameworks, e.g., (Cabral et al., 2004), and for the underlying technologies of service-oriented architecture, there is a thorough survey in (Papazoglou & van den Heuvel, 2007). (Küngas & Matskin, 2006) perform a survey of commercial and governmental web services to see which data they use, but this is more an investigation of the services as such, not of software development methods related to semantic web services.

(Srivastava & Koehler, 2003) have performed a survey of current solutions and open problems regarding web service composition. They focus on solutions used in industry vs. solutions proposed by academia. (Rao & Su, 2004) make a survey of automated approaches to service composition, referencing some of the same articles as this one, but of course missing those published after 2004. (Zimmermann *et al.*, 2005) discuss analysis and design techniques for service-oriented development in a survey-like manner, however, with fairly few references the paper is more a survey of issues than of sources in the field. (Ramollari *et al.*, 2007) present a survey of service-oriented development methodologies. The survey is most focused on the analysis and design stages, but does not focus specifically on semantic web services.

Hence, the main differences of the current survey compared to these previous ones is a more specific focus on engineering methods for SWS systems without restricting ourselves to any specific development stage, as well as looking at a broader range of methods.

SEMANTIC WEB SERVICES AND SOFTWARE ENGINEERING

Potential Benefits of Semantics

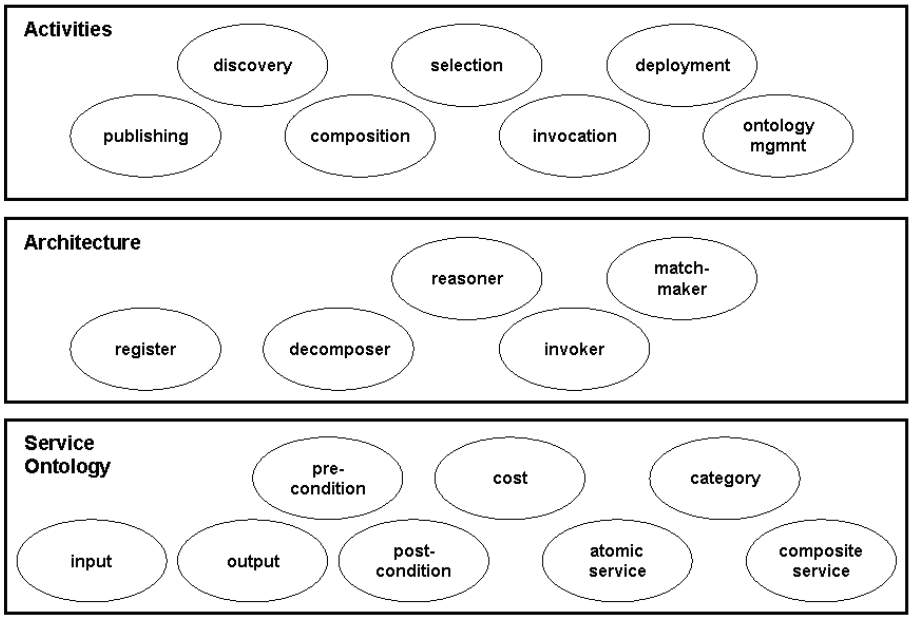
Existing technologies for traditional web services only allow descriptions at the syntactic level (Cabral *et al.*, 2004). This makes it difficult for requesters and providers to understand the meaning of the inputs and outputs and the real functionality of the service. This limitation may be overcome with the use of SWS. Semantic descriptions of web services are needed to enable automated discovery, composition and execution across heterogeneous users and domains (Cabral et al., 2004). Each SWS is described in a service ontology, enabling machines to understand its capabilities and reason over it related to other domain knowledge.

According to (Cabral et al., 2004), SWS infrastructures can be described along three different dimensions, as shown in Figure 1: *Usage activities*, *architecture* and *service ontology*. These dimensions are about the business, physical and conceptual level requirements for SWS respectively.

Special Lifecycle Concerns for SWS Systems

(Blake, 2007) discusses lifecycle issues in the development of service-oriented systems. Two different lifecycles are proposed. The development of single services is seen as similar to more traditional development, thus possible to do within mainstream lifecycles, for instance of RUP-like conceptualization and analysis, followed by iterative cycles of design, development, and testing, and finalized by deployment and retirement. However, the development of larger systems composed by such services, called service-centric software systems management, is seen as quite different,

Figure 1. SWS infrastructure dimensions, adapted from (Cabral et al., 2004)



thus needing a new kind of lifecycle with several different roles participating. Business-process engineers will be responsible for business-process conceptualization and domain analysis. Software engineers will then be responsible for the subsequent design stage. This has an iterative cycle much similar to the previous lifecycle, but the nature of the steps is different: service discovery, service composition, and evaluation. Finally, the operation stage is also different in nature, with the steps on-demand composition and rebinding, indicating a much higher dynamicity than for the traditional lifecycle.

(Gu & Lago, 2007) propose a more complex picture, illustrating the roles of several actors in one lifecycle picture. The main roles, however, are the service provider and application provider, which correspond to the two different kinds of lifecycle suggested by (Blake, 2007), and many similar aspects can be seen covered in both proposals. A somewhat more traditional lifecycle, much inspired by RUP, can be seen in (Papazoglou & van den Heuvel, 2006), identifying four different approaches to service-oriented applications: green-field, top-down, bottom-up, and meet-in-the-middle development.

(Tsai *et al.*, 2006) mention four challenges for service-oriented software engineering, namely decomposition challenge, utility challenge, quality challenge, and certification challenge, and state that service-oriented development has somewhat different methodological needs from the current mainstream focus on OOA / OOD with methods such as RUP and MSF. Yet, as pointed out by (Zimmermann et al., 2005) many of the general principles found in such mainstream methods still apply also in service-oriented software engineering, and extensions or adaptations of mainstream methods may therefore be one approach to better address service-oriented development. Some examples of RUP extensions are IBM's RUP for SOA (IBM, 2006) and the Service-Oriented Unified Process (Mittal, 2007). Sun's SOA RQ method (Sun, 2006) and the method proposed in

(Papazoglou & van den Heuvel, 2006) are also partly based on or have many similarities with RUP. Another example of a relationship between mainstream and service-oriented software development is OMG's effort to combine MDA with SOA, and to provide a meta-model for defining ontologies to help the construction of semantic web applications in an MDA context (OMG, 2006). The UML SOA profile proposed by (Johnston, 2005) can be seen as a related effort to align service-oriented development with mainstream techniques.

Another view on lifecycle can be found in situations where the idea is not to build new software but to offer services to a customer, for instance in a sourcing situation. The Information Technology Infrastructure Library, version 3 (ITIL v3) has made an increasing focus on service-oriented approaches in IT service management (Greiner, 2007; Keel *et al.*, 2007). From the perspective of the service-provider, the following lifecycle activities are envisioned: Service Strategy, Service Design, Service Transition, Service Operation, and Continual Service Improvement (Greiner, 2007). ITIL provides a common vocabulary for these activities, as well as guidelines to how they should be conducted.

Special Phase Concerns for SWS Systems

To find suitable specialized engineering methods, there is a need for understanding what special concerns arise when engineering systems based on SWS. On the requirements level, service-orientation poses both challenges and opportunities. The introduction of a new architectural paradigm might mean that some traditional development methods are no longer appropriate, and that new methods are needed. On the other hand, the presence and possible reuse of existing services might make requirements engineering easier (Jones *et al.*, 2005) – for instance, functionality of available services might inspire stakeholders to become clear about requirements they were not aware of, and could also enable rapid prototyping for elicitation and validation purposes.

Moreover, requirements engineering challenges can be found on two different levels, requirements for single services (which should, for instance, be as generic as possible to make the service competitive for usage in a wide range of applications), and requirements for applications or systems composed of various services, where the goal is to satisfy the needs of the stakeholders of that particular application. This is quite similar to the old distinction between development for reuse and development with reuse (Sindre *et al.*, 1995).

To understand challenges in design and later development stages we need to know what parts of a system must be implemented if we assume that there already exists a suitable infrastructure framework on which to build the system? Answering this question requires us to analyze what components and functions of the system are (or are envisioned to be) fully automated elements that need no involvement from the (business) user or system developer that are creating the end-user system.

All the *architecture* components for SWS (as shown in Figure 1) are meant to be automatic functions in an SWS framework. For example, *reasoning* is supposed to be performed without human intervention. Regarding the *activities* performed in an SWS system, most of these are clearly desired to be automatically performed, e.g., *publishing* and *invocation* of services. However, some of the activities may need to be performed manually or at least semi-automatically, depending on the complexity and type of system. This especially applies to *composition* and *selection* of services (D. Martin *et al.*, 2004). A system performing simple tasks within a well known domain may perform fully automated service composition and selection if the underlying framework is sufficiently robust and advanced (D. Martin et al., 2004). But some tasks may be so complicated

that they require a user to manually specify how different services are to be composed, or manually select desired services from lists of possible matches. In addition, sometimes users will want to be able to influence the composition process, e.g., if they only have a basic overall idea of their requirements for the system (Kim & Gil, 2004). One method of manually specifying a composition may be to define a process flow for the execution of a combination of different activities.

The *service ontology*, describing the service extensively, will usually have to be manually specified. This is done when implementing a service as a service provider, and must assure that the capabilities of the service are aligned with its service description (its service ontology). When specifying requirements for a SWS system, an overall service ontology may be created manually to specify the goal and requirements of the system, or a business process may be manually created and service ontologies defined for each of the activities in the process.

Service-oriented systems also have special challenges in the testing and evaluation phase. As pointed out by (Offutt & Xu, 2004) the testing of web service applications is difficult because they are distributed applications with runtime behaviors that differ from more traditional applications. For instance, the fact that service providers may change the implementation of a service at any time, introduces new challenges to regression testing (Ruth, 2008) and quality evaluation (Bianculli & Ghezzi, 2007).

Empowering the End User

Since the first software systems were created, systems have been developed at an increasing level of abstraction, making it easier for non-technical users to participate. SOS engineering (including SWS systems engineering) takes aim at providing modeling, design and development methods that map high-level business requirements to the technical implementation (Stojanovic & Dahanayake, 2005). Doing this will empower end-users to specify, design and develop their own systems, and thus "bridge the gap between business and IT" (Stojanovic & Dahanayake, 2005). Hence it is important that the methods and tools require as little technical skills as possible. Properly set up, and assuming an adequate supply of various services from providers, it could be possible for non-technical users to develop SWS systems in a fashion similar to Model Driven Development (MDD).

OVERVIEW OF METHODS AND TOOLS

In this section we survey methods and tools that support the engineering of SWS systems. We focus on methods and tools from the user perspective, i.e. involving user interaction.

Requirements for SWS Systems

The EU-funded IP project "Service Centric Systems Engineering" (SeCSE) has addressed various aspects of the development of service-oriented applications, including the requirements process (Jones et al., 2005; Zachos *et al.*, 2007). This is envisioned as an iterative cycle between requirements formulation and searching for services. Starting with an incomplete requirements specification, some possible services may be found in a service repository, these may again provide ideas – either because of functionality they contain, or properties they are lacking – for more detailed requirements. Tool support is developed for the approach, including a requirements assistant (UCaRE), a service discovery tool (EDDiE), and a service exploration tool. An evaluation of UCaRE is presented in (Zachos et al., 2007), while the EDDiE tool is presented in more detail in (Dourdas *et al.*, 2006).

While the efforts around UCaRE have mainly addressed the requirements specification of larger

systems (to be composed of services), another effort within SeCSE has addressed the specification of single services (Walkerdine *et al.*, 2007). A faceted specification approach is used to aid the later retrieval and comparison of services according to system specification. Also here, tool support has been developed and evaluated through a case study.

(Barn *et al.*, 2007) propose a tool workbench (SOA-MDK) for model-based development of service-oriented systems. The approach is inspired by component-based development, and the development method starts by developing business process models and information models, which can be considered RE activities on the system level. These models are then factored, and services allocated to the various parts. Finally, WSDL specifications are generated for the various services. A somewhat similar approach, but considering use cases plus non-functional requirements rather than business process models in the early stages, is proposed by (Dexter *et al.*, 2007).

Modeling and Design of SWS

(Timm & Gannod, 2005) recognize that the learning curve for the OWL-S descriptions language is steep, and that the current state of tool support is limited. They have created an automated software tool that uses model-driven architecture (MDA) techniques to generate OWL-S descriptions. The manual input to this tool is the description of an SWS in the form of a UML model specified in a standard UML modeling tool. Conversion to OWL-S is done via an XML representation of the UML model and XSLT.

(Jaeger *et al.*, 2005) propose a methodology for developing OWL-S descriptions that consists of three main steps: Generation of an OWL-S template, identification of available ontologies, and performing classification. Tasks in this methodology that must be performed manually are adapting or designing an ontology that describes the relevant domain, and classifications of the service, its inputs/outputs and any optional additions.

(Elenius *et al.*, 2005) have created a development tool for SWS, in the form of an OWL-S editor which aims to be easy and intuitive. The editor is implemented as a plug-in to the Protégé OWL ontology editor, and allows for engineering of all aspects of SWS by providing specialized views and design features wherever deemed necessary by the authors.

(Scicluna *et al.*, 2004) discuss how the difficult task of developing an OWL-S specification can result in incorrect specifications of the description or lead the user to rather utilize some other type of description language. They have created the OWL-S editor tool *OwlsWiz*, which enables creation of OWL-S ontologies from a WSDL description by presenting the user to a step-by-step process of inputting information. It allows a fast way of creating SWS descriptions while using well known standards such as WSDL and UML Activity Diagrams. Their main objective with the tool is to abstract away the underlying complexity of the OWL-S format and offer a relatively user-friendly way of building complex descriptions.

(Klein & Koenig-Ries, 2003) present an approach to improving the usability of DAML-S (now named OWL-S) by suggesting a process and a tool. The process guides through the activities needed to create adequate SWS descriptions by introducing a layered ontology of services, consisting of an upper service ontology, an ontology for the service category, and several domain ontologies. Their graphical tool *DINST* implements this process and claim to offer a comfortable way of editing SWS descriptions.

(Heß *et al.*, 2004) state that manually creating semantic metadata for services is tedious and error-prone. They propose a semi-automatic tool (*ASSAM*) for assisting a user in creating semantic metadata for SWS in the OWL-S format. *ASSAM* consists of two parts; a WSDL annotator appli-

cation, and OATS (Operation Aggregation Tool for Web Services), a data aggregation/schema matching algorithm. The WSDL annotator application uses machine learning to provide the user with a point-and-click interface containing suggestions on how to semantically annotate a web service. The tools have been evaluated, and findings indicate their usefulness.

(Brambilla *et al.*, 2007) propose a framework for developing semantic web service systems cross-cutting several enterprises. The proposed approach utilizes several available techniques, such as BPMN for the specification of business process models and WebML for the development of web applications. WSMO ontologies are semi-automatically elicited from the design of the applications. Their model-driven design process is seen as consisting of four steps: process design, data design, hypertext design, and semantic description. For this approach, a CASE-tool has also been presented (Brambilla *et al.*, 2006).

Composition of Semantic Web Services

(Sirin *et al.*, 2004) have developed a goal-oriented and interactive composition approach that utilizes matchmaking algorithms to assist users in filtering and selecting services while building the composition. They have created a semi-automatic tool that uses OWL-S service descriptions and filters and selects services using matchmaking algorithms. The system has two separate components: One component is an inference engine that stores service requirements and processes match requests. The other is the composer that generates the service composition workflow. The composer communicates with the inference engine to discover possible matching services, and presents them to users that may create a workflow of services based on the choices presented at each step. Users enter constraints on the service parameters from a form generated by the system using the ontologies that define those parameters. Users make the final selection of the specific services at each step.

(Kim & Gil, 2004) have developed a framework for interactive composition of SWS that guide users in sketching a composition of services. They argue that users benefit when employing intelligent tools that help them specify complete and correct pathways, even if there is only a small number of services that need to be put together. Their approach is to first take existing service descriptions and extend them with domain ontologies and task ontologies that attend to various task types in the domain. Their analysis tool *CAT* (Composition Analysis Tool) then uses these ontologies when examining and giving specific suggestions regarding the user's solution. They demonstrate how *CAT* may be useful in a case study regarding the domain of earthquake science.

The Service Component Architecture (SCA) (Chappell, 2007) is a joint effort in by a number of American companies, defining a platform for how to program web services and how to assemble them. However, it focuses more on defining necessary standards and infrastructure, and less on software engineering methods.

IRS-III (Hakimpour *et al.*, 2005) is a framework for structured web service composition. Their approach is supported by a tool which automatically recommends candidate web services to satisfy the specified composition context.

Testing / Evaluation of Service-Oriented Systems

(Offutt & Xu, 2004) present an approach for automated generation of test cases for web service systems, based on data perturbation. Automated test generation is also the topic of (Sinha & Paradkar, 2006), proposing an approach based on finite state machines. In this approach, WSDL-S specifications are taken as input, hence utilizing some provided semantics of services.

(E. Martin *et al.*, 2007) presents a prototype tool for robustness testing of web service systems,

but this is for web services in general, not specifically for semantic web services.

(Ruth, 2008) presents algorithms for safe test selection when performing regression testing on web service applications. (Penta *et al.*, 2007) propose the usage of genetic algorithms to produce test data for examining whether web services adhere to service level agreements.

As for evaluation, (Bianculli & Ghezzi, 2007) propose an approach for run-time quality monitoring of conversational web services, based on algebraic specifications. (Zhu *et al.*, 2006) present an approach for model-driven generation of benchmarks to evaluate service-oriented systems.

Methods Spanning Several Development Phases

(Gomez-Perez & González-Cabero, 2004) propose a framework for design and semi-automatic composition of SWS at a language-independent and knowledge level. The framework is directly based on the stack of ontologies that describe all the features of SWS. The framework specifies how to create SWS with the capabilities required by an external agent or user, focusing on three models: An instance model (specifying the SWS at the knowledge level), a checking model (utilizing design rules - called axioms - to check that there are no inconsistencies or errors in ontology instances) and a translation model (translating the service modeled at the knowledge level into an SWS-oriented language such as OWL-S combined with WDSL/SOAP). A prototype tool called *ODE SWS* has been implemented as an integrated part of *WebODE*, which is a platform for ontology development. The tool allows users to manually design descriptions and compositions of services at a conceptual level using a graphical user interface. The service model and composition is automatically checked for errors before performing an automatic translation into the OWL-S specification.

(Howard & Kerschberg, 2004) recognize the need for a comprehensive and overarching framework that deals with the processing and workflow requirements of Virtual Organizations, maps these into a group of service-oriented activities, dynamically configures these activities from available services and manages the choreography and execution of these services. They propose such a framework, which they call the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework. The framework deals with specification of service requirements, mapping of these requirements to composable services, and coordination of the execution of services conforming to enterprise workflow requirements. The KDSWS Framework has a functional architecture containing a user layer. The idea is that users would visit a portal of the e-enterprise, compose their request as a high-level task, and cooperate with a planning agent to decompose the task into a plan that is automatically processed further. A framework implementation is planned in the future.

(Papazoglou & van den Heuvel, 2006) propose a method which is inspired by RUP, combined with business process modeling and component-based development, covering all phases from planning to deployment and execution of the service-oriented system. However, while all phases are mentioned, some of them (e.g., testing) are discussed in rather general terms, not containing any suggestions of specific service-oriented techniques for that phase. The development of an integrated tool set supporting the approach is mentioned as future work in the article.

COMPARISON AND DISCUSSION

In this section we perform a simple classification and comparison of the different methods and tools, and discuss the result.

Table 1 below gives an overview of the methods and tools identified in the survey dealing with stages before testing. It indicates what development phase the methods are intended for, whether tool support is documented for the method (the first Y/N in the Tool & Eval column), and if the usefulness of the method/tool is evaluated (the second Y/N in the Tool & Eval column). Note that an N in one of these columns does not necessarily mean that tool support or evaluations has not been established, only that this was not found documented in the publication. Furthermore, the "Lang" column indicates which modeling languages are utilized by the approach, and the final "Proc" column whether it makes any particular assumptions about the

Table 1. An overview of methods and tools supporting the engineering of SWS systems

Approach	Phase	Tool & eval		Lang	Proc
SOA-MDK (Barn et al., 2007; Dexter et al., 2007)	reqs	Y	Y	BPM, info. mod. Use cases	Inspired by CBD
SeCSE (Walkerdine et al., 2007; Zachos et al., 2007)	reqs, design, comp.	Y	Y	Use cases, Struct natural lang, WSDL	Extended RUP
(Timm & Gannod, 2005)	design	Y	N	UML, OWL-S	no
(Jaeger et al., 2005)	design	N	N	OWL-S	no
(Elenius et al., 2005)	design	Y	N	OWL-S	no
(Scicluna et al., 2004)	design	Y	N	OWL-S, WSDL, UML AD	no
(Klein & Koenig-Ries, 2003)	design	Y	N	OWL-S	no
(Heß et al., 2004)	design	Y	Y	OWL-S, WSDL	no
(Brambilla et al., 2007)	design	Y	Y	BPMN, WebML, WSMO	Own process
(Gomez-Perez & González-Cabero, 2004)	design, comp.	Y	N	OWL-S	no
(Sirin et al., 2004)	comp.	Y	N	OWL-S	no
(Kim & Gil, 2004)	comp.	Y	Y	WSDL	no
IRS-III (Hakimpour et al., 2005)	comp.	Y	Y	WSMO, UPML	no
(Howard & Kerschberg, 2004)	all	N	N	KDL, KDSPL	Own process
(Papazoglou & van den Heuvel, 2006)	all	N	N	BPEL	Inspired by RUP

overall development process, e.g., being related to RUP or component-based development (CBD). Note here that a "no" does not necessarily mean that the approach could not be used together with, e.g., RUP, only that the publication does not make any particular assumption about this. Hence a "no" here may in some ways be a disadvantage (no overall process provided) but also in some ways an advantage (the proposed technique can potentially be used with many different larger-scale development methods).

Table 2 similarly gives an overview of the surveyed approaches to testing and evaluation of service-oriented systems. The columns indicate what kind of testing the technique supports, what kind of specifications it takes as input, what it delivers as output, and what approach is used for the intermediate processing.

The list of methods might not be comprehensive, but despite of this it may be argued that much more research has been done on the enabling technologies of SWS than research regarding methods for requirements engineering, modeling, design and composition of systems based on these technologies.

Most of the methods (all but two) are accompanied by a software tool. This might indicate that the need for tool support is recognized and reckoned with. However, most of the tools are at an early stage of development and undergoing continued development, showing that this research area is still somewhat experimental and immature. Yet, there is a positive trend that publications from the last 2-3 years tend to include evaluations of the proposed methods, while earlier publications did not.

Several methods focus on modeling & design or composition or both phases of development, but only two of the methods focus explicitly on requirements specification for SWS systems. This may be because this has not received a lot of attention in research, or because more general methods apply well to specifying requirements for SWS systems. Many methods talk about cre-

Table 2. An overview of test and evaluation methods

Source	Kind of testing	input	output	approach
(Offutt & Xu, 2004)	functional	XML	test cases	Data perturbation
(Sinha & Paradkar, 2006)	functional	WSDL-S	Test cases	Extended finite state machines
(E. Martin et al., 2007)	robustness	WSDL	Test cases, test results	Java code generation
(Ruth, 2008)	regression	WSDL	Test selections	agents
(Penta et al., 2007)	Service level agreement	WS-BPEL	Test data	Genetic algorithms
(Bianculli & Ghezzi, 2007)	Perceived quality (run-time monitoring)	BPEL	Quality Evaluation	Algebraic specifications
(Zhu et al., 2006)	Performance / benchmarking	UML	benchmark	MDA

ating specifications of SWS systems, however it is important to note that a specification/description of a system is not the same as the system requirements.

Several of the suggested approaches underline the importance of user friendliness for the users that will make use of the methods and tools, by minimizing the needed amount of user input and the technical skills needed to provide it. This might point towards the underlying technologies being complex, and a wish for relatively non-technical business users to be able to create SWS systems in a fashion similar to Model Driven Development (MDD).

Most of the methods and tools utilize the OWL-S specification, indicating that it is very popular compared to other alternative technologies. Although most of the papers in this survey briefly rationalize the usefulness and argue the general benefits of their methods or correctness of their tools, only six of the methods have been evaluated for their usefulness. This hints at the relative immaturity of this research area, and a large potential for further research.

CONCLUSION AND SUGGESTIONS FOR FUTURE RESEARCH

Engineering of Semantic Web Services (SWS) systems include two important tasks that usually must be performed with input from a human user; requirements engineering, and modeling and design. In addition, manual input might often be needed when composing SWS into a workflow. Most of the other parts of SWS systems engineering are at least envisioned to be performed automatically. This indicates that engineering of SWS systems might become similar in fashion to Model Driven Development (MDD).

Our findings indicate that although some research have been done regarding methods for requirements engineering, modeling, design and composition of SWS systems, this research area is still somewhat experimental and immature. The need for tool support is recognized and reckoned with, and especially the approaches published in 2007 or later tend to come with working tool support, although this is often in the form of research prototypes that may not always be easily integrated with other mainstream software development tools. The OWL-S specification seems to be very popular in use compared to other alternative technologies. Methods for requirements engineering specifically aimed at SWS systems have not received a lot of attention. This might indicate that more general methods apply well to specifying requirements for SWS systems.

Research is being done to allow business users to be able to create SWS systems with as little technical knowledge as possible. This might also lead to engineering of SWS systems being performed in a fashion similar to MDD. There is also an evident convergence between business process modeling, enterprise application integration and service-orientation, for instance exemplified by the fact that many BPMS tools take a service-oriented approach. This convergence can also be seen in the survey, as languages such as BPMN and BPEL are included in several of the surveyed approaches in Table 1 and 2. The typical picture, however, is that the largest and most comprehensive tools tend to address service-oriented development in general, without any particular focus on semantics, while the tools that are ambitious on semantics tend to be smaller research prototypes with a more limited scope, dealing with only one or a few development phases. Hence, better tool support can be envisioned when ambitious semantic-based functionality also reaches the mainstream tools.

Future research is needed on methods for engineering of SWS systems. In particular, there is a need for further refinement of tools supporting the methods, and there is a strong need for research evaluating the usefulness of the proposed methods and tools. It will also prove valuable to do more research on requirements engineering for

SWS systems, to examine if general requirements engineering methods are adequate, or if more suitable methods can be found when considering the distinctive character of SWS systems.

ACKNOWLEDGMENT

This work was performed in the context of the project WISEMOD, funded by the Norwegian Research Council.

REFERENCES

Barn, B., Dexter, H., Oussena, S., & Sparks, D. (2007). SOA-MDK: Towards a method development kit for service oriented system development. In G. Magyar, G. Knapp, W. Wojtkowski, W. G. Wojtkowski & J. Zupančič (Eds.), *Advances in information systems development: New methods and practice for the networked society* (pp. 191-201). Berlin: Springer.

Bianculli, D., & Ghezzi, C. (2007). *Monitoring conversational web services.* Paper presented at the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07), Dubrovnik, Croatia. Retrieved 21.1., 2008, from http://www.selab.isti.cnr.it/IW-SOSWE07/program.html.

Blake, M. B. (2007). Decomposing composition: Service-oriented software engineers. *IEEE Software, 24*(6), 68-77.

Brambilla, M., Ceri, S., Comai, S., & Fraternali, P. (2006). A CASE tool for modelling and automatically generating web service-enabled applications. *Int. J. Web Engineering and Technology, 2*(4).

Brambilla, M., Ceri, S., Facca, F. M., Celino, I., Cerizza, D., & Della Valle, E. (2007). Model-driven design and development of semantic web service applications. *ACM Transactions on Internet Technology, 8*(1), 3:1-3:31.

Cabral, L., Domingue, J., Motta, E., Payne, T. R., & Hakimpour, F. (2004). *Approaches to semantic web services: An overview and comparisons.* Paper presented at the ESWS'04.

Chappell, D. (2007). *Introducing sca*: Chappell & Associates.

Dexter, H., Petch, J., & Powley, D. (2007, 11-12 Jan). *Establishing an SOA composite applications development process for work-based learning and competency progression management.* Paper presented at the 2nd TENCompetence Open Workshop, Manchester, UK. Retrieved 20.1., 2008, from http://www.medicine.manchester.ac.uk/helm/TENCompPaper.pdf

Dourdas, N., Zhu, X., Maiden, N., Jones, S., & Zachos, K. (2006). Discovering remote software services that satisfy requirements: Patterns for query reformulation. In Dubois, E. & Pohl, K. (Eds.): *Advanced Information Systems Engineering, 18th International Conference*, CAiSE 2006, Luxembourg, June 5-9, 2006, Proceedings. Lecture Notes in Computer Science 4001 (pp. 239-254), Berlin: Springer.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., et al. (2005). The OWL-S editor: A development tool for semantic web services. In Gómez-Pérez, A. & Euzenat, J. (Eds.): *The Semantic Web: Research and Applications*, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings. Lecture Notes in Computer Science 3532 (pp. 78-92). Berlin: Springer.

Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design.* Upper Saddle River, NJ: Prentice-Hall.

Glass, R. L., Ramesh, V., & Vessey, I. (2004). An analysis of research in computing disciplines. *Communications of the ACM, 47*(6), 89-94.

Gomez-Perez, A., & González-Cabero, R. (2004, 22-24 Mar). *A framework for design and composition of semantic web services.* Paper presented at the First International Semantic Web Services Symposium, Palo Alto, CA. Retrieved 22.4., 2007, from http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf

Greiner, L. (2007). ITIL: The international repository of it wisdom. *Business: the 8th layer*, 9-11.

Gu, Q., & Lago, P. (2007). *A stakeholder-driven service life cycle model for soa.* Paper presented at the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07), Dubrovnik, Croatia. Retrieved 21.1., 2008, from http://www.selab.isti.cnr.it/IW-SOSWE07/program.html.

Hakimpour, F., Sell, D., Cabral, L., Domingue, J., & Motta, E. (2005). *Semantic web service composition in IRS-III: The structured approach.* Paper presented at the 7th IEEE International Conference on E-Commerce Technology (CEC'05).

Heß, A., Johnston, E., & Kushmerick, N. (2004). *Assam: A tool for semi-automatically annotating semantic web services.* Paper presented at the International Semantic Web Conference (ISWC04), Hiroshima, Japan.

Howard, R., & Kerschberg, L. (2004). A framework for dynamic semantic web services management. *International Journal of Cooperative Information Systems, 13*(4), 441-485.

IBM. (2006). IBM RUP for service-oriented modeling and architecture v2.4. Retrieved 2.4., 2008, from http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/

Jaeger, M. C., Engel, L., & Geihs, K. (2005). *A methodology for developing OWL-S descriptions.* Paper presented at the Workshop on Web Services and Interoperability at First International Conference on Interoperability of Enterprise Software and Applications, Geneva, Switzerland.

Johnston, S. (2005). UML 2.0 profile for software services. Retrieved 4.4., 2008, from http://www-128.ibm.com/developerworks/rational/library/05/419_soa/

Jones, S. V., Maiden, N. A. M., Zachos, K., & Zhu, X. (2005). *How service-centric systems change the requirements process.* Paper presented at the 11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'05), Porto, Portugal, 13-14 Jun.

Keel, A. J., Orr, M. A., Hernandez, R. R., Patrocinio, E. A., & Bouchard, J. (2007). From a technology-oriented to a service-oriented approach to IT management. *IBM Systems Journal, 46*(3), 549-564.

Kim, J., & Gil, Y. (2004). *Towards interactive composition of semantic web services.* Paper presented at the First International Semantic Web Services Symposium.

Kitchenham, B. (2004). *Procedures for performing systematic reviews* (Technical Report No. TR/SE-04-01). Keele, Staffordshire, UK: Keele University.

Klein, M., & Koenig-Ries, B. (2003). *A process and a tool for creating service descriptions based on DAML-S.* Paper presented at the 4th International Workshop Technologies for E-Services (TES 2003).

Küngas, P., & Matskin, M. (2006, 19-25 Feb). *Web services roadmap: The semantic web perspective.* Paper presented at the AICT/ICIW'06.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., et al. (2004). *Bringing semantics to web services: The OWL-S approach.* Paper presented at the First International Workshop on Semantic Web Services and Web Process Composition.

Martin, E., Basu, S., & Xie, T. (2007). WebSob: A tool for robustness testing of web services. *29th International Conference on Software Engineer-*

ing (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007, *Companion Volume* (pp. 65-66). Los Vaqueros, CA: IEEE Computer Society.

Mora, M., Gelman, O., Paradice, D., & Cervantes, F. (2008). The Case for Conceptual Research in Information Systems. In: Janczewski, L. & Gutierrez, J.: *Proceedings of the 2008 International Conference in Information Resources Management* (pp 1-10), Hershey, Pennsylvania: IRMA.

Mittal, K. (2007). Service oriented unified process. Retrieved 3.3., 2008, from http://www.kunalmittal.com/html/soup.shtml

Offutt, J., & Xu, W. (2004). Generating test cases for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes, 29*(5), 1-10.

OMG. (2006). Ontology definition metamodel. Retrieved 3.4., 2008, from http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf

Papazoglou, M. P., & van den Heuvel, W.-J. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology, 2*(4), 412-442.

Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service-oriented architectures: Approaches, technologies and research issues. *VLDB Journal, 16*, 389-415.

Penta, M. D., Canfora, G., Esposito, G., Mazza, V., & Bruno, M. (2007). *Search-based testing of service level agreements.* In Thierens, D. et al. (Eds.), *GECCO´07 - Genetic and Evolutionary Computation Conference*, (pp. 1090-1097), New York: ACM Press.

Ramollari, E., Dranidis, D., & Simons, A. J. H. (2007). *A survey of service oriented development methodologies.* Paper presented at the 2nd European Young Researchers Workshop on Service Oriented Computing, Leicester, UK. Retrieved 3.2., 2008, from http://www.dcs.shef.ac.uk/~ajhs/research/papers/soasurvey.pdf

Rao, J., & Su, X. (2004). A survey of automated web service composition methods. In J. Cardoso & A. Sheth (Eds.), *Semantic web services and web process composition* (pp. 43-54): Springer (LNCS 3387).

Ruth, M. E. (2008). *Concurrency in a decentralized automatic regression test selection framework for web services.* Paper presented at the 15th ACM Mardi Gras conference (MG'08), Baton Rouge, LA.

Scicluna, J., Abela, C., & Montebello, M. (2004). *Visual modeling of OWL-S services.* Paper presented at the IADIS International Conference WWW/Internet, Madrid, Spain.

Sindre, G., Conradi, R., & Karlsson, E.-A. (1995). The reboot approach to software reuse. *Journal of Systems and Software, 30*(3), 201-212.

Sinha, A., & Paradkar, A. (2006). *Model-based functional conformance testing of web services operating on persistent data.* In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications* (TAV-WEB'06), Portland, Maine (pp. 17-22). New York: ACM.

Sirin, E., Parsia, B., & J., H. (2004). Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems, 19*(4), 42–49.

Srivastava, B., & Koehler, J. (2003). *Web service composition - current solutions and open problems.* Paper presented at the ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy.

Stojanovic, Z., & Dahanayake, A. (2005). Service-oriented software system engineering: Challenges and practices: Preface. In Z. Stojanovic & A. Dahanayake (Eds.), *Service-oriented software system engineering: Challenges and practices* (pp. vii-xv). Hershey, PA: Idea Group.

Stojanovic, Z., Dahanayake, A., & Sol, H. (2004). *Modeling and design of service-oriented architecture.* In International Conference on Systems, Man and Cybernetics (pp. 4147 - 4152), IEEE.

Sun. (2006). SOA RQ methodology: A pragmatic approach. Retrieved 2.4., 2008, from http://www.sun.com/products/soa/soa_methodology.pdf

Timm, J. T. E., & Gannod, G. C. (2005, 11-15 Jul). *A model-dr*iven *approach for specifying semantic web services.* Proceedings of the IEEE International Conference on Web Services (pp. 313-320), Los Vaqueros, CA: IEEE Computer Society Press.

Tsai, W. T., Malek, M., Chen, Y., & Bastani, F. (2006, May 27-28). *Perspectives on service-oriented computing and service-oriented system engineering.* In Proceedings International Workshop on Service Oriented Software Engineering (pp. 3-10), IEEE.

Walkerdine, J., Hutchinson, J., Sawyer, P., Dobson, G., & Onditi, V. (2007). *A faceted approach to service specification.* Paper presented at the Second International Conference on Internet and Web Applications and Services (ICIW'07).

Zachos, K., Maiden, N. A. M., Zhu, X., & Jones, S. (2007, 11-15 Jun). *Discovering web services to specify more complete system requirements.* In Opdahl, A.L., Sindre, G. & Krogstie, J.: 19th Conference on Advanced Information Systems Engineering (CAiSE'07), Proceedings, Lecture Notes in Computer Science 4495 (pp. 142-157), Berlin: Springer.

Zhu, L., Gorton, I., Liu, Y., & Bui, N. B. (2006). *Model driven benchmark generation for web services.* In *International Workshop on Service Oriented Software Engineering* (pp. 33-39), IEEE.

Zimmermann, O., Schlimm, N., Waller, G., & Pestel, M. (2005). Analysis and design techniques for service-oriented development and integration. *GI Jahrestagung*(2), 606-611.

This work was previously published in the International Journal of Information Systems in the Service Sector, edited by J. Wang, Volume 1, Issue 2, pp. 1-16, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 1.29
Knowledge Management Software

Rodrigo Baroni de Carvalho
FUMEC University, Brazil

Marta Araújo Tavares Ferreira
Federal University of Minas Gerais (UFMG), Brazil

INTRODUCTION

Due to the vagueness of the concept of knowledge, the software market for knowledge management (KM) seems to be quite confusing. Technology vendors are developing different implementations of the KM concepts in their software products. Because of the variety and quantity of KM tools available on the market, a typology may be a valuable aid to organizations that are searching and evaluating KM software suitable to their needs.

The objective of this article is to present a typology that links software features to knowledge processes described in the SECI (socialization, externalization, combination, internalization) model developed by Nonaka and Takeuchi (1995). KM solutions such as intranet systems, content-management systems (CMSs), groupware, work flow, artificial intelligence- (AI) based systems, business intelligence (BI), knowledge-map systems, innovation support, competitive intelligence (CI) tools, and knowledge portals are discussed in terms of their potential contributions to the processes of socialization, externalization, internalization, and combination.

BACKGROUND

KM intends to be an area of research and practice that deepens the understanding of knowledge processes in organizations and develops procedures and instruments to support the transformation of knowledge into economic and social progress. In fact, different aspects of these issues have been studied for decades in many different disciplines as R&D (research and development) and innovation management, information systems management, information science, computer science, library studies, innovation economics, science and technology social studies, epistemology, and many others. Maybe one of the most important contributions of the KM concept is the creation

of a space (in academy and in the business world) where these many groups and points of view may discuss and work together.

KM studies analyze people, organizations, processes, and technology. Although technology is not the main component of KM, it would be naive to implement KM without considering any technological support. According to Stewart (1998), the intellectual capital of an organization has three dimensions: human capital, structural capital, and client capital. Structural capital is defined as the organizational systems and structures that store and transfer knowledge, and it includes the quality and extent of information systems, databases, patents, written procedures, and business documents. From this perspective, KM software should be considered as an important component of the structural capital of organizations.

This article assumes that IT has a supporting role, not the main role, in a KM program. According to Terra (2000), KM has seven dimensions: strategy, culture and organizational values, organizational structure, human resource skills, IT, measuring, and environmental learning. Therefore, IT is only one of the dimensions of KM, and technology alone does not transform information into knowledge. The KM ultimate challenge is to increase the chances of innovation through knowledge creation. The role of IT in this context is to extend the human capacity of knowledge creation through the speed, memory extension, and communication facilities of technology.

Nonaka and Takeuchi (1995) have analyzed the knowledge-creation process of Japanese organizations and developed a framework (SECI model). This model relates the knowledge creation of firms to four knowledge conversion processes.

- **Socialization (S):** the process of sharing tacit knowledge through shared experiences. As apprentices learn the craft of their masses through observation, imitation, and practice, so do employees of a firm learn new skills through on-the-job training.
- **Externalization (E):** where tacit knowledge is articulated into explicit knowledge with the help of metaphors and analogies. Externalization is triggered by dialog and collective reflection.
- **Combination (C):** the process of converting explicit knowledge into more systematic sets of explicit knowledge.
- **Internalization (I):** where explicit knowledge is converted into tacit knowledge. This usually occurs when explicit knowledge is put into practice. It is also related to shared mental models and work practices.

These interactions build a continuous spiral from the individual to organizational level. Ponzi (2004) used bibliometric techniques to analyze 2,240 source records obtained from scientific citation indexes. His research revealed that Nonaka and Takeuchi (1995) is the top most cited reference in the KM area and the most influential work. Due to this popularity, we have decided to use the SECI model to help individuals who already know this framework but need a better understanding of the KM software market.

There are some related works concerning KM software categorization: Barnes (2001), Bellaver and Lusa (2002), Davenport and Prusak (1998), Fernandez, Gonzalez, and Sabherwal (2004), Maier (2004), Malhotra (2000), Rollet (2003), Ruggles (1997), and Tiwana (2002). None of these academic works establish a direct relationship between the KM systems and the SECI model. The authors usually prefer to use their own KM framework to analyze the link between knowledge processes and KM systems. There is also another type of proposal for categorization, that is, Microsoft (2000), which has been developed by vendors and is very IT based. It is not the objective of this article to discuss the differences and the similarities among these proposals, but they have been considered in the development of the typology presented here.

MAIN FOCUS OF THE ARTICLE

The main objective of this article is to present a typology of KM solutions present on the market that comprehends 10 categories, each of which emphasizes specific KM aspects. It also intends to identify which of the knowledge-conversion processes (Nonaka & Takeuchi, 1995) is supported by each software category. This article concludes by presenting some trends in KM software development and suggesting some guidelines for the launching of KM programs supported by IT.

To accomplish our objective, it was necessary to explore the software market in order to classify KM tools. The major difficulty in accomplishing this task was the establishment of limits on a growing market. A sample of KM software was constructed through information collected on KM-related sites selected in Nascimento and Neves (1999), on advertisements in KM magazines (*KM World*, *KM Magazine*, and *DM Review*), and in digital libraries (http://brint.com). The exploratory research resulted in a list of 26 software vendors that were contacted, from which 21 sent folders, technical briefings, and demo versions of their software. The analysis of each KM system basically consisted of an installation and feature checkup. It was tested if the features advertised by the vendor were really supported by the KM system. After the analysis of these tools, it was possible to identify some common features among them, which originated the typology's first version. This version (Carvalho, 2000) was composed of eight categories.

After this period, Collins (2003), Detlor (2004), Firestone (2003), Kim, Chaudhury, and Rao (2002), and Raol, Koong, Liu, and Yu (2002) published research related to the evaluation of KM software and the emergence of knowledge portals. Due to the development of the KM software market and influenced by the previously mentioned works, this typology was reviewed and updated in 2004. As a result, two new categories have been incorporated: competitive intelligence tools and knowledge portals. The KM systems are then discussed in terms of their contributions to the four knowledge conversion modes developed by Nonaka and Takeuchi (1995).

As a result of this research, 10 KM software categories are presented as follows:

- Intranet-based systems
- Content management systems
- Groupware
- Work flow
- Artificial intelligence-based systems
- Business intelligence
- Knowledge map systems
- Innovation support tools
- Competitive intelligence tools
- Knowledge portals

Intranet-Based Systems

An intranet is an appropriate tool to systematize and add the explicit knowledge that is dispersed through departments. Intranets are organizational assets and an important part of the structural dimension of intellectual capital, as described by Stewart (1998). The communication in intranets is usually passive because the user has to pull the information. Nevertheless, the efficient usage of intranets is closely related to a wider comprehension of information management contribution to organizational performance. An intranet, like other systems described in this article, should be understood as a part of the organizational information context, and its usefulness is influenced by culture, values, and principles concerning strategic information management. This explains why, despite the wide and varied set of features made possible by intranets, they have been used in most organizations primarily for basic information access, that is, the retrieval of corporate documents (Choo, Detlor, & Turnbull, 2000).

According to Choo et al. (2000), intranets are quite helpful in promoting the externalization, combination, and internalization processes. The

combination process is supported by unified access to multiple content sources, and internalization occurs when there is a dissemination of success stories and best practices on the intranet. Part of the intranet content is generated by employees who have decided to document their experiences and externalize their knowledge. Web server software, such as Apache HTTP (hypertext transfer protocol) Server, offers the basic features for intranet deployment.

Content Management Systems

CMSs manage repositories of important corporate documents and contribute to the organization of the vast amount of documents generated by office activities. Paperwork is still a reality, and each document is a source of nonstructured information that could be lost if not well organized. According to Rollet (2003), existing paper documents are brought into the CMS through scanning, and optical character recognition (OCR) software analyzes the resulting image files and translates them into computer-readable text. Bennet (1997) states that CMSs provide more efficient retrieval, and better security and version control of documents. File Net and Excalibur Retrieval Ware are examples of CMSs. These systems have many features-cataloging, metadata, searching, versioning, and indexing-that were inherited from the traditional information retrieval (IR) systems, which are studied in the field of library science.

CMSs deal only with the explicit dimension of knowledge, supporting then the combination process. The focus of CMS is primarily on providing access to existing documents in any available media: fax, e-mails, HTML (hypertext markup language) forms, computer reports, paper, video, audio, or spreadsheets.

Groupware

Organizations are searching for flexible structures that can easily adapt to a changing environment. The need of cooperation between geographically dispersed work groups is a critical issue to global organizations. Groupware systems have a push style where information is sent to the user. Groupware is a blend of synchronous (like chat), asynchronous (like e-mail), and community-focused tools (like e-groups). Informal communication predominates in a groupware environment. Groupware systems are well suited to support communities of practice, where specialists of a given domain of knowledge, who may be dispersed all over the world, exchange their expertise in order to find solutions to specific problems.

According to Nonaka and Takeuchi (1995), the externalization of tacit knowledge is induced by dialog and collective reflection. Groupware helps this process by permitting collaboration and the exchange of nonstructured messages. Discussion groups and chats are common groupware features that make possible the gradual articulation of tacit knowledge. The development of technologies, such as videoconferencing and instant messaging, has contributed to a better quality of interaction among groupware users. These enriched virtual environments provide a suitable context for the socialization and internalization processes. Choo et al. (2000) present online apprenticeship as an example of socialization supported by groupware. The Microsoft groupware suite (MS Exchange, MS Outlook, MS Messenger) and the Lotus family (Notes, Sametime, Lotus Workplace) are examples of groupware packages.

Work Flow

Work flow systems support standardized business processes. These systems regulate the information flow from person to person, place to place, and task to task in processes that require ordered and structured information. The objective of work flow systems is to establish and accelerate the process flow, following its steps and tracking each activity that composes the process. They make explicit the knowledge that is embedded in standard pro-

cesses, mainly supporting the formal codification of existing knowledge (externalization).

Cruz (1998) defines the three basic elements of work flow, also called the three *R*s model.

- **Roles:** set of skills to execute a specific task
- **Rules:** features that define how the data should be processed
- **Routes:** logical paths for the knowledge flow through the process

Work flow systems, like ARIS Toolset, present features that support the graphical representation of existing processes. These systems are also used for business process reengineering (BPR) because they make explicit who does what in what order, and what products or services are produced. Another interesting feature of work flow systems is simulation, which permits the dynamic analysis of business processes. The simulation supplies information about the execution of processes, process weak points, and resource bottlenecks. Work flow systems are usually integrated with groupware and CMSs in order to provide an organized document flow across knowledge workers, supporting then the execution of a business process.

Artificial Intelligence-Based Systems

AI is the computer-science field that has produced the first studies relating information to knowledge. Expert systems, CBR (case-based reasoning) systems, and neural networks are some types of systems that use AI techniques. An expert system is built on the observation of a specialist at work and on the mapping of part of his or her knowledge into derivation rules.

As Davenport and Prusak (1998) explain, CBR systems support learning from a set of narratives or cases related to a problem. When a user has a problem, he or she can check in the case database in order to find if it is related to a problem that has already been solved. CBR systems use pattern matching algorithms to retrieve cases that are more similar to the problem stated before. The user can interact with the system by analyzing the solutions of existing cases and refining the search. CBR systems have been successfully used in help desk and call-center applications. They help contributors to externalize what has been learned from experience through the narrative of cases.

On the other hand, frequent users of the CBR system can internalize the knowledge that is represented into the system. This knowledge can also be restructured and represented in another manner. According to Rollet (2003), fundamental prerequisites for AI methods are suitable ways of representing knowledge, and automated reasoning typically uses highly formalized knowledge bases consisting of explicit rules, using, for instance, predicate logic. AI is also important for the development of software components, like intelligent agents, that can be used in a wide range of information systems, helping search and retrieval features.

Neural networks, like CA-Neugents, are more sophisticated systems that use statistical instruments to process cause-effect examples and to learn the relationships involved in the solution of problems. Neural networks are very flexible and intelligent systems because each new input results in an automatic reprogramming and consequent addition of new relationships.

Business Intelligence

Business intelligence is a set of tools used to manipulate a mass of operational data and to extract essential business information from them. BI systems comprehend the following:

- **Front-end systems:** They consist of a comprehensive set of data analysis tools like OLAP (online analytical processing), data mining, query, and reporting.

- **Back-end systems:** DBMSs (database management systems), data warehouses, and data marts.

DBMSs are the basis of a BI solution. First, the operational data generated by business transactions are extracted from the DBMS, filtered by some criteria, and then moved to the data warehouse. After this BI back-end loading step, the front-end tools are able to identify hidden patterns inside the data, and the user is free to build his or her own queries and strategic reports. BI systems, like Business Objects and Oracle 10g BI, provide end users with self-service access to information stored in data marts, data warehouses, and online transaction processing (OLTP) systems. As Choo (1998) explains, organizations are using analysis tools to reveal patterns that would otherwise remain buried in their huge operational databases; software for OLAP, a front-end BI tool, allow users to create multidimensional views of large amounts of data as they "slice and dice" the data in various ways to discover patterns and trends.

The focus of BI is decision making. BI systems excel in the job of sorting, categorizing, and structuring information, and facilitating the reconfiguration of existing information (combination) as well as creating new information.

Knowledge Map Systems

Also known as expertise locators, knowledge maps work like yellow pages that contain a "who-knows-what" list. A knowledge map does not store knowledge. The map just points to people who own it, creating opportunities for knowledge exchange.

A standard knowledge map is fed with profiles of competencies of the members of an organization. The knowledge map provides an expert locator feature that helps users to find the expert's best suited to work on a specific problem or project. A knowledge map categorizes an organization's expertise into searchable catalogs. By using a knowledge map, it is easier to identify people in terms of whom they know, what they know, and how proficient they are at a given task.

Human resource specialists use knowledge maps to match existing competencies with strategic targets and to identify what kinds of know-how, essential for growth, are currently available. According to Terra (2000), knowledge maps facilitate tacit knowledge exchange because they provide faster expert search and increase the chance of personal meetings. This approximation can probably result in face-to-face contacts that promote shared experiences and learning by observation, imitation, and praxis (socialization).

Innovation Support Tools

Amidon (2000) defines innovation as the application of new ideas to products or services. The result of innovation can be observed by the number of new patents, the design modifications of existing products, and the development of new products. Innovation support tools are systems that contribute to knowledge generation along the product design process. These tools intend to create a virtual environment that stimulates the multiplication of insights and are especially used in industrial R & D. An innovation support tool may include different features.

- A technical database where patents, articles, and research projects are recorded
- Graphic simulation features, which can facilitate internalization
- Combinatory tools, which help to consider unusual possibilities in the design of innovations

Innovation support tools, like Goldfire Innovator from Invention Machines, are generally based on a scientific content or patent database that allows users to conceive new products, correct

product defects, design feature modifications to existing products, identify technology trends and future product road maps, or improve production processes. The combination process is also supported because an engineer can combine existing explicit knowledge to generate new patents or product specifications.

Competitive Intelligence Tools

FULD & Company Inc. (2000) describes the CI cycle in five steps.

- **Planning and direction:** the identification of questions that will drive the information gathering phase
- Published information collection
- **Primary source collection:** information gathering from people rather than from published sources
- **Analysis and production:** the transformation of the collected data into meaningful assessment
- **Report and inform:** the delivery of critical intelligence in a coherent manner to corporate decision makers

FULD and Company Inc. (2000) has evaluated the CI software offered on the market and has concluded that they offer better support to the second and fifth steps of the CI cycle. The other steps are very human based and are only slightly benefited by technology. In the second step, software agents perform the automatic collection of timely information from news feeds and search the Internet and corporate intranets. In the fifth step, CI tools accelerate the dissemination of reports by sending e-mail reports according to users' preferences. CI tools concentrate on the combination process of the knowledge conversion spiral. They act like a probe on information sources: The information that is obtained is filtered and classified before dissemination so it is disseminated in an adequate format to facilitate combination. On the other hand, CI tools contribute to sense making, which is related to the internalization process. According to Choo (1998), organizations first have to make sense of what is happening in their environments in order to develop a shared interpretation that can serve as a guide to action.

Knowledge Portals

In an attempt to consolidate various departmental intranets, organizations are constructing corporate intranets or portals (Choo et al., 2000). A great contribution of portals is the integration of heterogeneous information sources, providing a standard interface to the users. According to the authors, a portal's primary function is to provide a transparent directory of information already available elsewhere, not to act as a separate source of information itself. Common elements contained in corporate portal design include an enterprise taxonomy or classification of information categories that helps to ease retrieval, a search engine, and links to internal and external Web sites and information sources.

The personalization feature of portals enables end users to organize their work by community, interest, task, or job focus. Besides providing personal access to knowledge, portals help users in the job of building community places. Online awareness and real-chat capabilities are available throughout the portal. Therefore, the user can see who is online, connect with them instantly, and get immediate answers.

But portals are evolving into more complex and interactive gateways so that they may integrate in a single solution many KM tools' features presented before. They are becoming single points of entry through which end users and communities can perform their business tasks, and evolving into virtual places where people can get in touch with other people who share common interests. Knowledge portals are the next generation of

EIPs (enterprise information portals). Knowledge portals support all knowledge processes described in the SECI model because portals are in fact the amalgamation of many KM systems presented before.

Table 1 presents the 10 classes of KM software discussed in this article, their main contribution to knowledge conversion processes, the disciplinary origin of their main concepts, and some examples. The examples are merely illustrative and do not represent a recommendation or preference for any technology vendor.

It is interesting to notice how KM software covers a large spectrum of features, information resources, and users. For instance, CMSs are made to retrieve documents while knowledge map systems exist to find people. Like CMSs, BI supports the combination process. However, CMSs deal basically with documents that are usually nonstructured and appear in a great va-

Table 1. Categories of knowledge management software: Summary table

Category	Dominating Knowledge Conversion Processes	Origin of Concepts	Examples
Intranet-Based Systems	Externalization, Combination, Internalization	Computer Networks (Web Technology)	Apache HTTP Server
Content Management Systems	Combination	Information Science	Excalibur Retrieval Ware and File Net
Groupware	Socialization, Externalization, Internalization	CSCW (Computer-Supported Cooperative Work)	Lotus Family (Notes, Sametime) and MS Suite (Exchange, Outlook, Messenger)
Work Flow	Externalization	Organization and Methods	ARIS Toolset (IDS Scheer)
Artificial Intelligence-Based Systems	Externalization, Combination, Internalization	Artificial Intelligence	Neugents (Computer Associates)
Business Intelligence	Combination	Database Management	Business Objects and Oracle 10g BI
Knowledge Maps	Socialization	Information Science and Human Resource Management	Gingo (Trivium) and Lotus Discovery Server
Innovation Support Tools	Combination, Internalization	Innovation and Technology Management	Goldfire Innovator (Invention Machines)
Competitive Intelligence Tools	Combination	Strategic Management and Information Science	Knowledge Works (Cipher Systems) and Vigipro (CRIQ/CGI)
Knowledge Portals	Socialization, Externalization, Combination, Internalization	Computer Networks and Information Science	Hummingbird and Plumtree

riety of formats, while the basic BI structure is a database record with specific attributes and a standardized format. Finally, the users of innovation support tools are usually technicians, engineers, or scientists who are involved in some creative design process inside an R & D department, while managers are BI's typical users.

FUTURE TRENDS

There seems to be a trend of functional convergence in KM systems. Preserving initial features, vendors are incorporating extra features from others categories described in the typology presented in this article, transforming their products into KM-integrated suites. For instance, a BI system may start to offer a knowledge map feature in a new version. So, it seems that increasingly, KM software will be classified in more than one of the presented categories, which can be alternatively considered as an array of features for KM systems.

The portal technology is the materialization of this convergence trend. Scientific research about portal features and types of KM systems seems to merge, following the movement of the KM software market. Collins (2003) and Detlor (2004) are examples of this recent approach. According to Collins, the knowledge portal's basic features are BI, collaboration, and content management. As a result, a basic portal will require the integration of at least four KM systems presented in this article: the intranet, CMS, groupware, and BI.

Integration can be a cumbersome task and may not be seen as a short-term project, especially in the case where there are heterogeneous systems scattered all over the organization. Firestone (2003) emphasizes the role of XML (extensible markup language) in portal architecture and presents relations between portal-integration efforts and the current research concerning EAI (enterprise application integration). The better choice is to consider a knowledge portal as a gradual project, allowing organizations to expand later the capabilities and functionalities delivered through the portal.

CONCLUSION

The wise selection of KM software requires a previous analysis of an organization's knowledge needs. Among the considerations to be addressed in some organizations is the fact, for instance, that a low level of socialization may be the critical point; in other ones, externalization may need to be improved.

As to the adoption process, it is interesting to notice the differences between KM software and ERP (enterprise resource planning) systems. ERPs are usually implemented in a top-down style, and the organization generally has to adjust its processes to the system in a short period of time. It is impossible to do the same with a KM system. The commitment and motivation of members are crucial to any KM program, much more than better KM software. KM requires a long term strategy to involve people and break paradigms. Also, policies referring to participation, flexibility, autonomy, and career evolution must surely be adapted.

KM software can be considered an interdisciplinary business because their development requires not only technical skills, but also a deep understanding of social and managerial aspects. In this sense, Choo et al. (2000) suggest that intranet designers look for the lessons learned from the field of CSCW. Recommendations include the need to ensure that everyone benefits from groupware, the need to understand the current work practice, and the involvement of users in design.

As a result of the research presented in this article, we conclude that KM software is evolving in order to offer an integrated platform for organizational knowledge conversion processes. But this does not mean that the resources of KM software are already well exploited by the orga-

nizations that have adopted them. As reported in literature and as we have ourselves learned from the study of two Brazilian organization systems (Carvalho, 2000), their potential is most frequently underevaluated and unexplored. In fact, their actual utilization stresses mainly their support of information access and retrieval, while their communication and collaboration dimensions are yet to be discovered.

The implementation of KM systems is a complex process. The KM software needs not only to be integrated to the existing IT infrastructure, but to the organizational culture, procedures, and human resource policy as well. The correct balance between managerial and technical aspects constitutes one of KM-tool adoption's greatest challenges. According to Detlor (2004), culture and user behaviors are the key drivers and inhibitors of internal sharing, and organizations should develop ways of stimulating people to use and contribute to KM systems.

Many organizations that are implementing KM programs focus exclusively on the conversion of human capital into structural capital. They think of KM as an opportunity to extract part of the knowledge of their employees and store it in knowledge bases. This approach misunderstands the dynamic and complex characteristics of knowledge, its tacit-prevailing nature, and the fact that, more than existing knowledge, the incessant creation of knowledge is the distinctive feature.

The KM concept has recently been severely criticized (Berkman, 2001), and one of the reasons for this may be the excessive emphasis on software and methodologies per se. This argument emphasizes the importance of considering technology in its context, that is, of relating it to the complexity of knowledge processes in order not to over (or under) estimate technology, or to miss the opportunity of bringing knowledge to where it belongs: the center of organizational attention.

REFERENCES

Amidon, D. (2000). *Knowledge innovation.* Retrieved from *http://www.entovation.com*

Barnes, S. (2001). *Knowledge management systems: Theory and practice*. London: Thomson Learning Europe.

Bellaver, R., & Lusa, J. (2002). *Knowledge management strategy and technology.* Norwood, MA: Artech House.

Bennet, G. (1997). *Intranets: Como implantar com sucesso na sua empresa*. Rio de Janeiro, Brazil: Campus.

Berkman, E. (2001). When bad things happen to good ideas. *Darwin Magazine*. Retrieved from *http://www.darwinmag.com*

Carvalho, R. B. (2000). *Aplicações de softwares de gestão do conhecimento: Tipologia e usos.* MSc dissertation, Programa de Pós-Graduação em Ciência da Informação da UFMG, Belo Horizonte, Brazil.

Choo, C. W. (1998). *The knowing organization.* Oxford, UK: Oxford University Press.

Choo, C. W., Detlor, B., & Turnbull, D. (2000). *Web work: Information seeking and knowledge work on the World Wide Web.* Dordrecht, Germany: Kluwer Academic Publishers.

Collins, H. (2003). *Enterprise knowledge portals.* New York: Amacon.

Cruz, T. (1998). *Workflow: A tecnologia que vai revolucionar processos.* São Paulo, Brazil: Atlas.

Davenport, T., & Prusak, L. (1998). *Working knowledge: How organizations manage what they know.* Boston: HBS Press.

Detlor, B. (2004). *Towards knowledge portals.* Boston: Kluwer Academic Publishers.

Fernandez, I., Gonzalez, A., & Sabherwal, R. (2004). *Knowledge management and KM software package*. Harlow, UK: Pearson.

Firestone, J. (2003). *Enterprise information portals and knowledge management.* Burlington, UK: Butterworth-Heinemann.

FULD & Company Inc. (2000). *Intelligence software report.* Retrieved from *http://www.fuld.com*

Kim, Y., Chaudhury, A., & Rao, H. (2002). A knowledge management perspective to evaluation of enterprise information portals. In *knowledge and process management* (Version 9, pp. 57-71). Indianapolis, IN: John Wiley & Sons.

Maier, R. (2004). *Knowledge management systems: Information and communication technologies for knowledge management.* Heidelberg, Germany: Springer-Verlag.

Malhotra, Y. (2000). *Knowledge management and virtual organizations.* Hershey, PA: IGP.

Microsoft. (2000). Knowledge management: Produtividade organizacional. *ComputerWorld, 319,* 11-12.

Nascimento, N., & Neves, J. T. R. (1999). A gestão do donhecimento na World Wide Web: Reflexões sobre a pesquisa de informações na rede. *Perspectivas em Ciência da Informação, 4,* 29-48.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge creating company.* New York: Oxford Press.

Ponzi, L. (2004). Knowledge management: Birth of a discipline. In *Knowledge management lessons learned*. Medford, NJ: Information Today.

Raol, J., Koong, K., Liu, L., & Yu, C. (2002). An identification and classification of enterprise portal functions and features. *Industrial Management Data Systems, 102*. Retrieved from *http://www.emerald insight.com/0263-55777.htm*

Rollet, H. (2003). *Knowledge management: Processes and technologies.* Boston: Kluwer Academic Publishers.

Ruggles, R. (1997). *Knowledge management tools.* Burlington, UK: Butterworth-Heinemann.

Stewart, T. (1998). *Capital intelectual*. Rio de Janeiro, Brazil: Campus.

Terra, J. C. C. (2000). *Gestão do conhecimento: O grande desafio empresarial.* São Paulo, Brazil: Negócio Editora.

Tiwana, A. (2002). *The knowledge management toolkit: Practical techniques for building a knowledge management system*. New York: Prentice Hall.

KEY TERMS

Competitive Intelligence: Set of interrelated measures that aim at systematically feeding the organizational decision process with information about the organizational environment in order to make it possible for people to learn about it, to anticipate its evolution, and to make better decisions in consequence.

CSCW (Computer-Supported Cooperative Work): Branch of computer science dedicated to the study of groupware technologies.

Enterprise Information Portal (EIP): Single Web interface to corporate information.

Expert System: A special type of artificial intelligence system that contains a limited domain knowledge base, an inference mechanism to manipulate this base, and an interface to permit the input of new data and user dialog.

Groupware: Type of software that is designed to help teams that are geographically dispersed who need to work together.

Intelligent Agent: Software component capable of acting autonomously by perceiving the environment, evaluating choices, and deciding on actions without checking the user.

Knowledge Portal: A personalized interface to online resources for knowledge workers to integrate applications and data. It is an evolution of EIP.

Neural Networks: A system composed of a large number of software nodes connected by weighted links. The system learns by adjusting those weights through repetitive learning cycles.

OLAP (Online Analytical Processing): Front-end tool that allows the analysis of multidimensional data. It is commonly used in business intelligence systems.

This work was previously published in Encyclopedia of Knowledge Management, edited by D. Schwartz, pp. 410-418, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 1.30
Malicious Software

Thomas M. Chen
Southern Methodist University, USA

Gregg W. Tally
SPARTA, Inc., USA

INTRODUCTION

Malicious software (malware) allows an intruder to take over or damage a target host without the owner's consent and often without his or her knowledge. Over the past thirty years, malware has become a more serious worldwide problem as Internet-connected computers have proliferated and operating systems have become more complex. Today, the average PC user must be more cognizant of computer security than ever before due to the constant threat of possible infection. Although exact costs are difficult to determine, there is little doubt that malware has widespread impact on equipment damages, loss of data, and loss of productivity. According to surveys, malware is one of the most common and costly types of attack on organizations (CERT, CSO, & ECTF, 2005).

In the early days of computing, malware was predominantly viruses and Trojan horses that spread among computers mainly by floppy disks and shared files (Grimes, 2001). The typical virus writer was a young male experimenting by himself and looking for notoriety. Today, malware is largely worms, viruses, spyware, bots, and Trojans proliferating through computer networks. Worms are a particular concern due to their ability to spread by themselves through computer networks. They can exploit weaknesses in operating systems or common applications such as Web and e-mail clients. They are often used as vehicles to install other types of malware onto hosts. Many thousands of worms and viruses are constantly tracked by the WildList (Wildlist Organization International, 2006) and antivirus companies.

Naturally, host-based and network-based defenses have also evolved in sophistication in response to growing threats. Surveys have found that organizations almost universally use antivirus software, firewalls, intrusion detection systems, and other means of protection (Gordon, Loeb, Lucyshyn, & Richardson, 2005). These defenses certainly block a tremendous amount of malware and prevent global disasters. However, their effectiveness is widely known to be limited

by their ability to accurately detect malware. Detection accuracy is critical because malware must be blocked without interfering with legitimate computer activities or network traffic. This difficulty is compounded by the creativity of attackers continually attempting to invent new methods to avoid detection.

BACKGROUND

Self-Replicating Malware

Malware can be classified into self-replicating or nonself-replicating. Self-replicating malware consists of viruses and worms. Fred Cohen originated the term virus after biological viruses for their manner of parasitically injecting their RNA into a normal cell, which then hijack the cell's reproductive process to produce copies of the virus (Cohen, 1994). Analogously, computer viruses attach their code to a normal program or file, which takes over control of execution of the infected program to copy the virus code to another program.

Polymorphism was a major development in virus evolution around 1990. Polymorphic viruses are able to scramble their form to have at most a few bytes in common between copies to avoid detection by virus scanners. In 1991, the dark avenger's mutation engine was an easy to use program for adding polymorphism to any virus. A number of other "mutation engines" were subsequently created by other virus writers.

A new wave of mass-mailing viruses began with Melissa in 1999. It was a macro virus infecting Microsoft Word normal templates. On infected computers, it launched Microsoft Outlook and e-mailed copies of itself to 50 recipients in the address book. It demonstrated the effectiveness of e-mail as a propagation vector, infecting 100,000 computers in 3 days. Since then, e-mail has continued to be a popular vector for viruses and worms because e-mail is used by everyone across different operating systems (Harley, Slade, & Gattiker, 2001). Mass-mailing worms today often carry their own SMTP engines to mail themselves and circumvent security features in e-mail programs.

Whereas viruses are program fragments dependent on execution of a host program, worms are standalone programs capable of spreading by themselves (Nazario, 2004; Skoudis, 2004). A worm searches for potential targets through a computer network and sends a copy of itself if the target is successfully compromised. Worms take advantage of networks and have proliferated as Internet connectivity has become ubiquitous.

One of the earliest and most famous worms was written by Robert Morris Jr. in 1988. Perhaps released accidentally, it disabled 6,000 hosts, which was 10% of the ARPANET (the predecessor to the Internet). A number of fast worms, notably Code Red I, Code Red II, and Nimda appeared in 2001. Two years later, another wave of fast worms included SQL Slammer/Sapphire, Blaster, and Sobig.F. The following year was dominated by MyDoom, Netsky, and Bagle worms (Turner et al., 2006).

Nonself-replicating malware classification of nonself-replicating malware into disjoint subcategories is difficult because many types of nonself-replicating malware share similar characteristics. Perhaps the largest category is Trojan horses defined as programs with hidden malicious functions. A Trojan horse may be disguised as a legitimate program to avoid detection. For example, a Trojan horse could be installed on a host with the name of a legitimate system file (displacing that file). Alternatively, the intention of the disguise could be to deceive users into executing it. For example, a Trojan horse could appear to be a graphic attachment in an e-mail message but in actuality be a malicious program. Trojans do not replicate by themselves but could spread by file sharing or downloading.

Remote administration or access trojans (RATs) are a well-known type of trojan horse

giving covert remote control to attackers. One of the first was Netbus written in 1998. It works in a client-server fashion with the server component installed on the target machine responding to the attacker's client. Another well-known RAT was Back Orifice released by Cult of the Dead Cow in 1998, which was later released as an open source version Back Orifice 2000.

A backdoor is software giving access to a system bypassing normal authentication mechanisms (Skoudis, 2004). Programmers have written backdoors sometimes to allow convenient access for legitimate testing or administrative purposes, but backdoors can be installed and exploited by attackers to maintain covert remote control after a target has been compromised. For example, the Nimda worm dropped a backdoor on infected hosts.

Relatively recently, bots such as Spybot and Gaobot have become a major problem (Turner et al., 2006). Bots installed on a group of hosts act as a large bot net to carry out a remote attacker's instructions which are typically communicated via Internet relay chat (IRC). Bot net sizes in the thousands to hundreds of thousands have been observed. Bot nets have been rented or sold as platforms for spamming, distributed denial of service, and other criminal activities (Lewis, 2005).

A rootkit is low-level software, possibly at the kernel level, designed to conceal certain files and processes. Rootkits are sometimes bundled as part of malware such as worms (Hoglund & Butler, 2006) because the concealment allows attackers to maintain longer control over their targets.

Spyware is software that collects and sends personal information through the network to a remote attacker (Evans, 2005). Spyware may be bundled with a legitimate program, and its presence may be mentioned in a end user license agreement (EULA). Commonly, a type of spyware called adware is bundled for the purpose of collecting information about user behavior to customize delivery of advertising. Accepting the EULA is considered explicit agreement to installation of the spyware, but many people neglect to read EULAs carefully. More pernicious types of spyware deliberately hide their presence and attempt to steal personal data by recording data to a file which is transmitted to or retrieved by a remote attacker.

MALICIOUS SOFTWARE

Malware involves an ongoing conflict between attackers and defenders. Worms are a prime example of a malware attack. Computers are typically protected by a combination of host-based and network-based defenses.

Self replication basics worms actively select and attack their targets through a network automatically. The capability for self replication is enabled by certain functions in the worm code (Skoudis, 2004). First, a function for target location chooses the next host for attack. The simplest algorithm chooses random IP address as pseudorandomly generated 32-bit numbers. Random target selection is not completely effective because the B and C class address spaces are more populated. Hence, some worms target B and C class addresses more often. Also, some worms favor targets on the same local area network as the victim because they are easier to reach. Another common way to identify targets is to harvest e-mail addresses from the victim host.

Second, a function in the worm code must contain the infection mechanism to compromise a selected target. The most common method is an exploit of a vulnerability. Most operating systems and applications software have vulnerabilities or weaknesses discovered over time. The most common type of vulnerability is a buffer overflow, which can lead to running arbitrary malicious code on a target host if attacked successfully (Foster, Osipov, Bhalla, & Heinen, 2005). When a vulnerability is discovered, the software developer is usually notified privately and given a chance

to develop a patch or update. The vulnerability may be publicly disclosed later along with the patch. Vulnerabilities are regularly published in Microsoft security bulletins, CERT advisories, Bugtraq, MITRE CVEs, and other places. This process allows users to update their systems before attackers can write the exploit code that takes advantage of the vulnerability. Other vulnerabilities may be discovered by attackers but not disclosed, in hopes of catching targets unprotected against so-called zero-day exploits.

Exploits are not the only way for worms to spread. Social engineering takes advantage of human gullibility to trick users into taking an action to help the worm (e.g., opening an e-mail attachment). Password attacks attempt to compromise a target by trying default passwords, easily guessed passwords, or cracking the password file. Another way to spread is to look for backdoors left by other worms.

Worms can easily include multiple exploits to compromise more targets faster. The Morris worm was an example using a combination of different exploits to attack targets: a buffer overflow exploit of the Unix finger daemon; an exploit of the debug mode of the sendmail program; and cracking the password file by a dictionary attack. Another prominent example of a blended threat was Nimda in 2001, using five different vectors.

A third function in the worm code enables replication of the worm to a compromised target. Replication might be combined with the exploit. For example, SQL Slammer/Sapphire carried a buffer overflow exploit and a copy of the worm within a single 404-byte UDP packet.

Finally, worm code may optionally contain a payload. The payload is executed on the target and might be virtually anything such as data theft, data deletion, or installation of other malware.

Host-Based Defenses

The most common suite of host-based defenses includes antivirus software, spyware detection software, and a personal firewall. Antivirus and antispyware software aim to identify specific malware, disinfect, or remove infected files, and prevent new infections if possible. Antivirus and antispyware programs largely work by signatures, which are sets of characteristics that will identify a specific malware (Szor, 2005). Signatures are preferred for their accuracy in identifying known malware, but new malware without a matching signature can escape detection. Antivirus software typically include heuristic rules to detect suspicious new malware based on their behavior or construction. For example, behavior blocking looks at the behavior of programs and raises a warning if the behavior appears suspicious. The disadvantage of heuristics is a possibly high rate of false positives (false alarms).

Another defense against malware is software patching. Software developers often publicize new vulnerabilities along with patches for them. This works for known vulnerabilities but not all vulnerabilities are known by the developers. Also, it can be inconvenient for users to keep up with regular patching.

Host-based intrusion detection systems are processes that observe system activities and raise alarms for suspicious activities. For example, if someone fails several consecutive login attempts, that would be a suspicious activity suggesting that the person does not know the correct password.

Lastly, computers typically include personal firewalls, implemented as software at the network interface. Incoming and outgoing traffic is blocked according to the firewall policies. There might be firewalls on the perimeter of a user's network, but a personal firewall allows packet filtering to be customized to individual preferences.

Network-Based Defenses

Compared to host-based defenses, network-based defenses have the advantage of providing broad protection to groups of users without any special requirements on hosts (Nazario, 2004).

Firewalls are perhaps the best known network defense (Northcutt, Zeltser, Winters, Fredrick, & Ritchey, 2002). Firewalls apply filtering rules to block malicious traffic including malware. Rules are often based on fields in packet header fields such as source and destination addresses, source and destination ports, and protocol.

Routers with access control lists (ACLs) can block traffic similarly to firewalls. Routers must process packet headers for the purpose of forwarding packets along the correct routes. ACLs are simply additional rules to specify which packets are dropped.

Network-based intrusion detection systems (IDS) are specialized equipment to observe and classify traffic as normal, suspicious, or malicious. IDS raise alarms for suspicious traffic but do not take active actions (intrusion prevention systems have that additional capability to block malicious traffic). Like antivirus software, IDS typically work by a combination of signature-based and behavior-based detection (also called misuse and anomaly detection). Signatures are traffic characteristics that unique identify malware traffic and are preferred for accurate detection. However, not all malware traffic is known, and therefore malware might escape signature-based detection (Riordan, Wespi, & Zamboni, 2005). Behavior-based or anomaly detection aims to identify all suspicious traffic that deviates in some sense from normal traffic.

Honeypots are decoy computers intentionally set up to look vulnerable to attackers (Spitzner, 2003). They are not used for legitimate services so all traffic received by a honeypot is unsolicited and inherently suspicious. Their general purpose is to learn about attacker behavior but can be configured to collect malware, particularly worms that choose their targets automatically and randomly. The risks associated with malware impose the necessity for special precautions to limit possibly compromised honeypots from spreading malware to other computers.

CHALLENGES

New vulnerabilities are constantly being discovered in operating systems and applications software, giving rise to new exploits for malware. Turner et al. (2006) reported an average of 10 new vulnerabilities discovered per day. Accurate detection of new exploits requires signatures, but signatures usually takes a few hours to days to develop. In the absence of a signature, the effectiveness of defenses will depend on the accuracy of anomaly (or behavior-based) detection. Anomaly detection based on unique behavioral traits of worms is an active area of research (Al-Hammadi & Leckie, 2005; Gu, Sharif, Qin, Dagon, Lee, & Riley, 2004; Kawaguchi, Azuma, Ueda, Shigeno, & Okada, 2006). For example, random worms might be inferred by the observation of a large number of failed connection messages (Berk, Bakos, & Morris, 2003). Another active research problem is automated defenses after detection such as automatic generation of worm signatures (Newsome, Karp, & Song, 2005; Simkhada, Tsunoda, Waizumi, & Nemoto, 2005) or dynamic quarantine (Moore, Shannon, Voelker, & Savage, 2003).

The situation is complicated by the many means of self-preservation that malware today often use. First, malware attempts to be stealthy through polymorphism or rootkit techniques. Second, malware can actively attack defenses. It is not uncommon for viruses and worms to disable antivirus software on targets by stopping antivirus processes and disabling registry keys. Third, malware has the capability to dynamically download new code or plug-ins, changing its functionality.

FUTURE TRENDS

Malware is always seeking new propagation vectors in addition to the Internet. Recently, malware

has begun to spread via wireless networks to mobile devices such as cell phones and PDAs and is increasingly targeting instant messaging (Turner et al., 2006). E-mail and social engineering will continue to be popular propagation vectors.

The changing nature of payloads, increasingly towards remote control and data theft, suggests that malware is become more used for cybercrimes. Malware for profit has been called crimeware. This trend is also suggested by increasing use of stealth techniques.

Finally, worm outbreaks have become faster than humans can respond. For example, SQL Slammer/Sapphire is reported to have infected 90 percent of the vulnerable hosts within 10 minutes. This trend means more dependence on automated defenses in the future. However, the effectiveness of automated defenses will depend on a solution to the problem of accurate detection.

CONCLUSION

Current defenses based on signatures and anomaly detection are imperfect. Signatures are preferred for accuracy but take time to develop and distribute. On the other hand, anomaly detection has the difficult challenge of differentiating normal from malicious behavior. In the future, malware attacks will be carried out faster, and we will depend more on automated defenses. These defenses will need solutions to automating signature development and making anomaly detection more accurate.

Finally, users are an important part of security. Since malware often use social engineering, user education and awareness of secure practices (such as patching and antivirus updating) are essential. Just as with anything valuable, users must be constantly vigilant to protect their computers and data.

REFERENCES

Al-Hammadi, Y., & Leckie, C. (2005). Anomaly detection for Internet worms. In *Proceedings of IEEE IM 2005* (pp. 133-146).

Berk, V., Bakos, G., & Morris, R. (2003). Designing a framework for active worm detection on global networks. In *Proceedings of the 1st IEEE International Workshop on Info. Assurance* (pp. 13-23).

CERT, CSO, and ECTF. (2005). *2005 e-crime watch survey.* Retrieved April 24, 2006, from http://www.cert.org/archive/pdf/ecrimesummary05.pdf

Cohen, F. (1994). *A short course on computer viruses.* New York: Wiley & Sons.

Evans, G. (2005). *Spyware study and reference guide.* Marina Del Rey, CA: Ligatt Publishing.

Foster, J., Osipov, V., Bhalla, N., & Heinen, N. (2005). *Buffer overflow attacks: Detect, exploit, prevent.* Rockland, MA: Syngress Publishing.

Gordon, L., Loeb, M., Lucyshyn, W., & Richardson, R. (2005). *CSI/FBI computer crime and security survey.* Retrieved April 24, 2006, from http://www.gocsi.com

Grimes, R. (2001). *Malicious mobile code.* Sebastopol, CA: O'Reilly & Associates.

Gu, G., Sharif, M., Qin, X., Dagon, D., Lee, W., & Riley, G. (2004). Worm detection, early warning, and response based on local victim information. In *Proceedings of the 20th IEEE Annual Computer Security Applications Conf*erence (pp. 136-145).

Harley, D., Slade, R., & Gattiker, R. (2001). *Viruses revealed.* New York: McGraw-Hill.

Hoglund, G., & Butler, J. (2006). *Rootkits: Subverting the windows kernel.* Upper Saddle River, NJ: Addison-Wesley.

Kawaguchi, N., Azuma, Y., Ueda, S., Shigeno, H., & Okada, K. (2006). ACTM: Anomaly connection tree method to detect silent worms. In *Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Application* (pp. 901-908).

Lewis, J. (2005). *McAfee virtual criminology report: North American study into organized crime and the Internet.* Retrieved April 24, 2006, from http://www.mcafeesecurity.com/us/local_content/misc/mcafee_na_virtual_criminology_report.pdf

Moore, D., Shannon, C., Voelker, G., & Savage, S. (2003). Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of IEEE INFOCOM 2003.*

Nazario, J. (2004). *Defense and detection strategies against Internet worms.* Norwood, MA: Artech House.

Newsome, J., Karp, B., & Song, D. (2005). Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the 2005 IEEE Symp. on Security and Privacy* (pp. 226-241).

Northcutt, S., Zeltser, L., Winters, S., Fredrick, K., & Ritchey, R. (2002). *Inside network perimeter security: The definitive guide to firewalls, vpns, routers, and intrusion detection systems.* Indianapolis, IN: New Riders.

Riordan, J., Wespi, A., & Zamboni, D. (2005). How to hook worms. *IEEE Spectrum, 42*(5), 32-36.

Simkhada, K., Tsunoda, H., Waizumi, Y., & Nemoto, Y. (2005). Differencing worm flows and normal flows for automatic generation of worm signatures. In *Proceedings of IEEE International Symp. on Multimedia.*

Skoudis, E. (2004). *Malware: Fighting malicious code.* Upper Saddle River, NJ: Prentice-Hall PTR.

Spitzner, L. (2003). *Honeypots: Tracking hackers.* Boston, MA: Pearson Education.

Szor, P. (2005). *The art of computer virus research and defense.* Upper Saddle River, NJ: Addison-Wesley.

Turner, D., Entwisle, S., Friedrichs, O., Ahmad, D., Blackbird, J., & Fossi, M. (2006). *Symantec Internet security threat report: Trends for July 2005-December 2005.* Retrieved April 24, 2006, from http://www.symantec.com.

Wildlist Organization International. (2006). Retrieved April 24, 2006 from http://www.wildlist.org/WildList/.

KEY TERMS

Antivirus: Software to detect viruses and worms, clean infected files, and prevent new infections.

Exploit: Software written to take advantage of a specific vulnerability.

Firewall: A device or software to selectively filter packets.

Intrusion Detection System: A device or software to detect suspicious or malicious activities.

Malware: Software intended to perform a malicious action.

Rootkit: Low-level software designed to avoid detection on a compromised host.

Spyware: A type of malware that collects personal user information and transmits to a remote attacker.

Trojan Horse: A type of malware with a hidden malicious function.

Virus: A type of self-replicating malware that infects other files or programs.

Vulnerability: A security weakness in operating system or application software.

Worm: A standalone program capable of automated replicating itself through a computer network.

This work was previously published in Encyclopedia of Internet Technologies and Applications, edited by M. Freire & M. Pereira, pp. 284-290, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 1.31
Current Challenges in Intrusion Detection Systems

H. Gunes Kayacik
Dalhousie University, Canada

A. Nur Zincir-Heywood
Dalhousie University, Canada

INTRODUCTION

Along with its numerous benefits, the Internet also created numerous ways to compromise the security and stability of the systems connected to it. In 1995, 171 vulnerabilities were reported to CERT/CC © while in 2003, there were 3,784 reported vulnerabilities, increasing to 8,064 in 2006 (CERT/CC©, 2006). Operations, which are primarily designed to protect the availability, confidentiality, and integrity of critical network information systems are considered to be within the scope of security management. Security management operations protect computer networks against denial-of-service attacks, unauthorized disclosure of information, and the modification or destruction of data. Moreover, the automated detection and immediate reporting of these events are required in order to provide the basis for a timely response to attacks (Bass, 2000). Security management plays an important, albeit often neglected, role in network management tasks.

Defensive operations can be categorized in two groups: static and dynamic. Static defense mechanisms are analogous to the fences around the premises of a building. In other words, static defensive operations are intended to provide barriers to attacks. Keeping operating systems and other software up-to-date and deploying firewalls at entry points are examples of static defense solutions. Frequent software updates can remove the software vulnerabilities, which are susceptible to exploits. Firewalls provide access control at the entry point; they therefore function in much the same way as a physical gate on a house. In other words, the objective of a firewall is to keep intruders out rather than catching them. Static defense mechanisms are the first line of defense, they are relatively easy to deploy and provide significant defense improvement compared to the

initial unguarded state of the computer network. Moreover, they act as the foundation for more sophisticated defense mechanisms.

No system is totally foolproof. It is safe to assume that intruders are always one step ahead in finding security holes in current systems. This calls attention to the need for dynamic defenses. Dynamic defense mechanisms are analogous to burglar alarms, which monitor the premises to find evidence of break-ins. Built upon static defense mechanisms, dynamic defense operations aim to catch the attacks and log information about the incidents such as source and nature of the attack. Therefore, dynamic defense operations accompany the static defense operations to provide comprehensive information about the state of the computer networks and connected systems.

Intrusion detection systems are examples of dynamic defense mechanisms. An intrusion detection system (IDS) is a combination of software and hardware, which collects and analyzes data collected from networks and the connected systems to determine if there is an attack (Allen, Christie, Fithen, McHugh, Pickel, & Stoner, 1999). Intrusion detection systems complement static defense mechanisms by double-checking firewalls for configuration errors, and then catching the attacks that firewalls let in or never perceive (such as insider attacks). IDSs are generally analyzed from two aspects:

- **IDS deployment:** Whether to monitor incoming traffic or host information.
- **Detection methodologies:** Whether to employ the signatures of known attacks or to employ the models of normal behavior.

Regardless of the aspects above, intrusion detection systems correspond to today's dynamic defense mechanisms. Although they are not flawless, current intrusion detection systems are an essential part of the formulation of an entire defense policy.

DETECTION METHODOLOGIES

Different detection methodologies can be employed to search for the evidence of attacks. Two major categories exist as detection methodologies: misuse and anomaly detection. Misuse detection systems rely on the definitions of misuse patterns, which are the descriptions of attacks or unauthorized actions (Kemmerer & Vigna, 2002). A misuse pattern should summarize the distinctive features of an attack and is often called the signature of the attack in question. In the case of signature based IDS, when a signature appears on the resource monitored, the IDS records the relevant information about the incident in a log file. Signature-based systems are the most common examples of misuse detection systems. In terms of advantages, signature-based systems, by definition, are very accurate at detecting known attacks, which are included in their signature database. Moreover, since signatures are associated with specific misuse behavior, it is easy to determine the attack type. On the other hand, their detection capabilities are limited to those within signature database. As the new attacks are discovered, a signature database requires continuous updating to include the new attack signatures, resulting in potential scalability problems. Furthermore, attackers are known to alter their exploits to evade signatures. Work by Vigna, Robertson, Balzarotti (2004) described a methodology to generate variations of an exploit to test the quality of detection signatures. Stochastic modification of code was employed to generate variants of exploits to render the attack undetectable. Techniques such as packet splitting, evasion, and polymorphic shellcode were discussed.

As opposed to misuse IDSs, anomaly detection systems utilize models of the acceptable behavior of the users. These models are also referred to as normal behavior models. Anomaly-based IDSs search for the deviations from the normal behavior. Deviations from the normal behavior are consid-

ered as anomalies or attacks. As an advantage over signature-based systems, anomaly-based systems can detect known and unknown (i.e., new) attacks as long as the attack behavior deviates sufficiently from the normal behavior. However, if the attack is similar to the normal behavior, it may not be detected. Moreover, it is difficult to associate deviations with specific attacks since the anomaly-based IDSs only utilize models of normal behavior. As the users change their behavior as a result of additional service or hardware, even the normal activities of a user may start raising alarms. In that case, models of normal behavior should be redefined to maintain the effectiveness of the anomaly-based IDS. Similar to the case of misuse IDSs, attackers are known to alter their exploits to be recognized as normal behavior by the detector, hence evading detection. The general approach employed for evading anomaly detectors is based on the generation of mimicry attacks to perform evasion. A mimicry attack is an exploit that exhibits legitimate normal behavior while performing malicious actions. Methodologies exist to create mimicry attack automatically (Giffin, Jha, & Miller, 2006; Kayacik, Zincir-Heywood, & Heywood, 2007) or manually (Kruegel, Kirda, Mutz, 2005; Tan, Killourhy, & Maxion, 2002; Wagner & Soto, 2002).

In today's intrusion detection systems, human input is essential to maintain the accuracy of the system. In the case of signature-based systems, as new attacks are discovered, security experts examine the attacks to create corresponding detection signatures. In the case of anomaly systems, experts are needed to define the normal behavior. Therefore, regardless of the detection methodology, frequent maintenance is essential to uphold the performance of the IDS.

Given the importance of IDSs, it is imperative to test them to determine their performance and eliminate their weaknesses. For this purpose, researchers conduct tests on standard benchmarks (Kayacik & Zincir-Heywood, 2003; Pickering, 2002). When measuring the performance of intrusion detection systems, the detection and false positive rates are used to summarize different characteristics of classification accuracy. In simple terms, false positives (or false alarms) are the alarms generated by a nonexistent attack. For instance, if an IDS raises alarms for the legitimate activity of a user, these log entries are false alarms. On the other hand, detection rate is the number of correctly identified attacks over all attack instances, where correct identification implies the attack is detected by its distinctive features. An intrusion detection system becomes more accurate as it detects more attacks and raises fewer false alarms.

IDS DEPLOYMENT STRATEGIES

In addition to the detection methodologies, data is collected from two main sources: traffic passing through the network and the hosts connected to the network. Therefore, according to where they are deployed, IDSs are divided into two categories: those that analyze network traffic and those that analyze information available on hosts such as operating system audit trails. The current trend in intrusion detection is to combine both host-based and network-based information to develop hybrid systems and therefore not rely on any one methodology. In both approaches however, the amount of audit data is extensive, thus incurring large processing overheads. A balance, therefore, exists between the use of resources, and the accuracy and timeliness of intrusion detection information.

Network-based IDSs inspect the packets passing through the network for signs of an attack. However, the amount of data passing through the network stream is extensive, resulting in a trade off between the number of detectors and the amount of analysis each detector performs. Depending on throughput requirements, a network-based IDS may inspect only packet headers or include the content. Moreover, multiple detectors are typi-

cally employed at strategic locations in order to distribute the task. Conversely, when deploying attacks, intruders can evade IDSs by altering the traffic. For instance, fragmenting the content into smaller packets causes IDSs to see one piece of the attack data at a time, which is insufficient to detect the attack. Thus, network-based IDSs, which perform content inspection, need to assemble the received packets and maintain state information of the open connections, where this becomes increasingly difficult if a detector only receives part of the original attack or becomes "flooded" with packets.

A host-based IDS monitors resources such as system calls made by critical applications, logs, file systems, processor, and disk resources. Example signs of intrusion on host resources are unusual system call sequences, critical file modifications, segmentation fault errors, crashed services, or extensive usage of the processors. As opposed to network based IDSs, host-based IDSs can detect attacks, which are transmitted over an encrypted channel. Moreover, information regarding the software that is running on the host is available to host-based IDS. For instance, an attack targeting an exploit on an older version of a web server might be harmless for the recent versions. Network-based IDSs have no way of determining whether the exploit has a successful chance, or of using a priori information to constrain the database of potential attacks. Moreover, network management practices are often critical in simplifying the IDS problem by providing appropriate behavioral constraints, thus making it significantly more difficult to hide malicious behaviors (Cunningham, Lippmann, & Webster, 2001).

CHALLENGES

The intrusion detection problem has three basic competing requirements: speed, accuracy, and adaptability. The speed problem represents a quality of service issue. The more analysis (accurate) the detector, the higher the computational overhead. Conversely, accuracy requires sufficient time and information to provide a useful detector. Moreover, the rapid introduction of both new exploits and the corresponding rate of propagation require that detectors be based on a very flexible/scalable architecture. In today's network technology, where gigabit Ethernet is widely available, existing systems face significant challenges merely to maintain pace with current data streams (Kemmerer & Vigna, 2002).

An intrusion detection system becomes more accurate as it detects more attacks and raises fewer false alarms. IDSs that monitor highly active resources are likely to have large logs, which in turn complicate the analysis. If such an IDS has high false alarm rate, the administrator will have to sift through thousands of log entries, which actually represent normal events, to find the attack related entries. Therefore, increasing false alarm rates will decrease the administrator's confidence in the IDS. Moreover, intrusion detection systems are still reliant on human input in order to maintain the accuracy of the system. In case of signature-based systems, as new attacks are discovered, security experts examine the attacks to create corresponding detection signatures. In the case of anomaly systems, experts are needed to define the normal behavior. This leads to the adaptability problem. The capability of the current intrusion detection systems for adaptation is very limited. This makes them inefficient in detecting new or unknown attacks or adapting to changing environments (i.e., human intervention is always required). Although a new research area, incorporation of machine learning algorithms provides a potential solution for accuracy and adaptability of the intrusion detection problem.

CURRENT EXAMPLES OF IDS

Intrusion detection systems reviewed here are by no means a complete list but a subset of open

source and commercial products, which are intended to provide readers different intrusion detection practices.

- **Snort:** Snort is one of the best-known lightweight IDSs, which focuses on performance, flexibility, and simplicity. It is an open-source intrusion detection system that is now in quite widespread use (Roesch, 1999). Snort is a network-based IDS which employs signature-based detection methods. It can detect various attacks and probes including instances of buffer overflows, stealth port scans, common gateway interface attacks, and service message block system probes (Roesch, 1999). Hence, Snort is an example of active intrusion detection systems that detects possible attacks or access violations while they are occurring (CERT/CC ©, 2001).
- **Cisco IOS (IDS Component):** Cisco IOS provides a cost-effective way to deploy a firewall with network-based intrusion detection capabilities. In addition to the firewall features, Cisco IOS Firewall has 59 built-in, static signatures to detect common attacks and misuse attempts (Cisco Systems, 2003). The IDS process on the firewall router inspects packet headers for intrusion detection by using those 59 signatures. In some cases, routers may examine the whole packet and maintain the state information for the connection. Upon attack detection, the firewall can be configured to log the incident, drop the packet or reset the connection.
- **Tripwire:** When an attack takes place, attackers usually replace critical system files with their versions to inflict damage. Tripwire (Tripwire Web Site, 2004) is an open-source host-based tool, which performs periodic checks to determine which files are modified in the file system. To do so, Tripwire takes snapshots of critical files. Snapshot is a unique mathematical signature of the file where even the smallest change results in a different snapshot. If the file is modified, the new snapshot will be different than the old one, therefore critical file modification would be detected. Tripwire is different from the other intrusion detection systems because rather than looking for signs of intrusion, Tripwire looks for file modifications. Tripwire also offers a commercial version of the open source detector.
- **Stide:** Stide (Forest et al.,1996) employs a methodology based on immune systems where the problem is characterized as distinguishing self from nonself (normal and abnormal behaviors respectively). An event horizon is built from a sliding window applied to the sequence of system calls made by an application during normal use. The sequences formed by the sliding window are then stored in a table, which comprises the normal database. During the detection phase, if a pattern from the sliding window is not in the normal database, it is flagged as an anomaly. Recent work proposed improvements on Stide by employing finite state automata (Sekar et al., 2001), virtual path tables (Feng et al., 2003) and static analysis of the source code of the application (Wagner et al., 2001).

FUTURE TRENDS

As indicated above, various machine learning approaches have been proposed in an attempt to improve on the generic signature-based IDS. The basic motivation is to measure how close a behavior is to some previously established gold standard of misuse or normal behavior. Depending on the level of a priori or domain knowledge, it may be possible to design detectors for specific categories of attack (e.g., Denial of Service, User to Root, Remote to Local). Generic machine learning approaches include clustering or data-mining

in which case the data is effectively unlabeled. The overriding assumption is that behaviors are sufficiently different for normal and abnormal behaviors to fall into different "clusters." Specific examples of such algorithms include artificial immune systems (Hofmeyr & Forrest, 2000) as well as various neural network (Kayacik, Zincir-Heywood, & Heywood, 2003; Lee & Heinbuch, 2001) and clustering algorithms (Eskin, Arnold, Prerau, Portnoy, & Stolfo, 2002).

Naturally, the usefulness of machine learning systems is influenced by the features on which the approach is based (Lee & Stolfo, 2001). Domain knowledge that has the capability to significantly simplify detectors utilizing machine learning often make use of the fact that attacks are specific to protocol-service combinations. Thus, first partitioning data based on the protocol-service combination significantly simplifies the task of the detector (Ramadas, Ostermann, & Tjaden, 2003).

When labeled data is available, then supervised learning algorithms are more appropriate. Again, any number of machine learning approaches have been proposed, including: decision trees (Elkan, 2000), neural networks (Hofmann & Sick, 2003), and genetic programming (Song, Heywood, & Zincir-Heywood, 2003). However, irrespective of the particular machine learning methodology, all such methods need to address the scalability problem. That is to say, datasets characterizing the IDS problem are exceptionally large (by machine learning standards). Moreover, the continuing evolution of the base of attacks also requires that any machine learning approach also have the capability for online or incremental learning. Finally, to be of use to network management practitioners, it would also be useful if machine learning solutions were transparent. That is to say, rather than provide "black box solutions," it is much more desirable if solutions could be reverse engineered for verification purposes. Many of these issues are still outstanding, with cases that explicitly address the computational overhead in learning against large datasets only just appearing (Song, Heywood, & Zincir-Heywood, 2003).

CONCLUSION

Intrusion detection system is a crucial part of the defensive operations, which complements the static defenses such as firewalls. Essentially, intrusion detection is searching for signs of attacks and when an intrusion is detected, intrusion detection system can take an action to stop the attack by closing the connection or report the incident for further analysis by administrators. According to the detection methodology, intrusion detection systems can be categorized as misuse detection and anomaly detection systems. According to the deployment, they can be classified as network-based or host-based, although such distinction is coming to an end in today's intrusion detection systems where information is collected from both network and host resources. In terms of performance, an intrusion detection system gets more accurate, as it detects more attacks and raises fewer false alarms. However, no intrusion detection is infallible, attackers use detector weaknesses and blind spots to evade intrusion detection systems. Fortunately, penetration testing and ethical hacking became a part of the field of research.

REFERENCES

Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., & Stoner, E. (1999). *State of the practice of intrusion detection technologie* (CMU/SEI Tech. Rep. No. CMU/SEI-99-TR-028). Retrieved December 1, 2007, from http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html

Bass, T. (2000). Intrusion detection systems and multisensor data fusion. *Communications of the ACM, 43*(4), 99-105.

CERT/CC © (2006). *Incident statistics 1988-2006.* Retrieved December 1, 2007, from http://www.cert.org/stats/

CERT/CC © (2001). *Identifying tools that aid in detecting signs of intrusion.* Retrieved December 1, 2007, from http://www.cert.org/security-improvement/implementations/i042.07.html

Cisco Systems Inc. (2003). *Cisco IOS firewall intrusion detection system documentation.* Retrieved December 1, 2007, from http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t5/iosfw2/ios_ids.htm

Cunningham, R. K., Lippmann, R. P., & Webster S. E. (2001). Detecting and displaying novel computer attacks with macroscope. *IEEE Transactions on Systems, Man, and Cybernetics – Part A, 31*(4), 275-280.

Elkan C. (2000). Results of the KDD'99 classifier learning. *ACM SIGKDD Explorations, 1,* 63-64.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo S. (2002) A geometric framework for unsupervised anomaly detection: Detecting attacks in unlabeled data. In D. Barbara & S. Jajodia (Eds.), *Applications of Data Mining in Computer Security* (chapter 4). Kluwer. ISBN 1-4020-7054-3.

Feng, H., Kolesnikov, O., Fogla, P., Lee, W., & Gong, W. (2003). Anomaly detection using call stack information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (pp. 62-74), Washington, D.C. IEEE Computer Society Press.

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (pp. 120-128). Los Alamitos, CA: IEEE Computer Society Press.

Giffin, J. T., Jha, S., & Miller, B. (2006). Automated discovery of mimicry attacks. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID),* Hamburg, Germany.

Hofman, A., & Sick, B. (2003). Evolutionary optimization of radial basis function networks for intrusion detection. In *Proceedings of the International Joint IEEE-INNS Conference on Neural Networks* (pp. 415-420).

Hofmeyr, S. A., & Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation, 8*(4), 443-473.

Kayacik, G., & Zincir-Heywood, N. (2003). A case study of three open source security management tools. In *Proceedings of International Symposium on Integrated Network Management.*

Kayacik, G., Zincir-Heywood, N., & Heywood, M. (2003). On the capability of an som based intrusion detection system. In *Proceedings of International Joint Conference on Neural Networks.*

Kayacik, G., Zincir-Heywood, N., & Heywood, M. (2007). Automatically evading IDS using GP authored attacks. In *Proceedings of the IEEE Computational Intelligence for Security and Defense Applications.*

Kemmerer, R. A., & Vigna, G. (2002, April). Intrusion detection: A brief history and overview. *IEEE Security and Privacy,* pp. 27-29.

Kruegel, C., Kirda, E., Mutz, D., Robertson, W., & Vigna, G. (2005). Automating mimicry attacks using static binary analysis. In *Proceedings of the USENIX Security Symposium* (pp. 161-176).

Lee, S. C., & Heinhuch, D. V. (2001). Training a neural-network based intrusion detector to recognize novel attacks. *IEEE Transactions on Systems, Man, and Cybernetics – Part A, 31*(4), 294-299.

Pickering, K. (2002). *Evaluating the viability of intrusion detection system benchmarking.* B.S. thesis submitted to The Faculty of the School of Engineering and Applied Science, University of Virginia. Retrieved December 1, 2007, from http://www.cs.virginia.edu/~evans/students.html

Ramadas, M., Ostermann, S., & Tjaden, B. (2003). Detecting anomalous network traffic with self-organizing maps. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection* (LNCS 2820, pp. 36-54). Springer-Verlag.

Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference* (pp. 229-238).

Sekar, R., Bendre, K., Dhurjati, D., Bollineni, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of 2001 IEEE Symposium on Security and Privacy* (pp. 144-155), Oakland, CA. IEEE Computer Society Press.

Song, D., Heywood, M. I., & Zincir-Heywood, A. N. (2003). A linear genetic programming approach to intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03).*

Tan, K. M. C., Killourhy, K. S., & Maxion, R. A. (2002). Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*. (LNCS 2820, pp. 54-73).

Tripwire Web Site. (2004). *Home of the Tripwire open source project.* Retrieved December 1, 2007, from http://www.tripwire.org/

Vigna, G., Robertson, W., & Balzarotti, D. (2004). Testing network based intrusion detection signatures using mutant exploits. In *Proceedings of the ACM Conference on Computer Security* (pp. 21-30).

Wagner, D., Dean, D. (2001). Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (pp. 156-169), Oakland, CA. IEEE Computer Society Press.

Wagner, D., & Soto, P. (2002). Mimicry attacks on host based intrusion detection systems. In *Proceedings of the ACM Conference on Computer and Communications Security* (pp. 255-264).

Wenke, L., Stolfo, S.J., Chan, P.K., Eskin, E., Wei, F., Miller, M., Hershkop, S., & Junxin, Z. (2001). Real time data mining-based intrusion detection. In *Proceedings of DARPA Information Survivability Conference & Exposition II* (vol.1, pp. 89-100), Anaheim, CA.

KEY TERMS

Attack vs. Intrusion: A subtle difference—intrusions are the attacks that succeed. Therefore, the term *attack* represents both successful and attempted intrusions.

CERT / CC ©: CERT Coordination Center. Computer security incident response team, which provide technical assistance, analyze the trends of attacks, and provide response for incidents. Documentation and statistics are published at their web site: http://www.cert.org.

Exploit: Taking advantage of a software vulnerability to carry out an attack. To minimize the risk of exploits, security updates, or software patches should be applied frequently.

Fragmentation: When the data packet is too large to transfer on given network, it is divided into smaller packets. These smaller packets are reassembled on destination host. Among with

other methods, intruders can deliberately divide the data packets to evade IDSs.

Light Weight IDS: An intrusion detection system, which is easy to deploy and have smaller footprint on system resources.

Logging: Recording vital information about an incident. Recorded information should be sufficient to identify the time, origin, target, and if applicable, characteristics of the attack.

Machine Learning: A research area of artificial intelligence, which is interested in developing algorithms to extract knowledge from the given data.

Open Source Software: Software with its source code available for users to inspect and modify to build different versions.

Penetration Testing: A part of computer security research, where the objective of an "ethical hacker" is to discover the weaknesses and blind spots of the security software such as intrusion detection systems.

Security Management: In network management, the task of defining and enforcing rules and regulations regarding the use of the resources.

This work was previously published in Encyclopedia of Multimedia Technology and Networking, Second Edition, edited by M. Pagani, pp. 305-311, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.32
A Comparison and Scenario Analysis of Leading Data Mining Software

John Wang
Montclair State University, USA

Xiaohua Hu
Drexel University, USA

Kimberly Hollister
Montclair State University, USA

Dan Zhu
Iowa State University, USA

ABSTRACT

Finding the right software is often hindered by different criteria as well as by technology changes. We performed an analytic hierarchy process (AHP) analysis using Expert Choice to determine which data mining package was best suitable for us. Deliberating a dozen alternatives and objectives led us to a series of pair-wise comparisons. When further synthesizing the results, Expert Choice helped us provide a clear rationale for the decision. The issue is that data mining technology is changing very rapidly. Our article focused only on the major suppliers typically available in the market place. The method and the process that we have used can be easily applied to analyze and compare other data mining software or knowledge management initiatives.

INTRODUCTION

Based on the *knowledge life cycle* model, four stages of knowledge creation, knowledge storage/retrieval, knowledge transfer, and knowledge application have been proposed by Alavi and Leidner

(2001) and confirmed by Jennex (2006). "To be effective knowledge management systems, KMS, must support the various knowledge management functions of knowledge capture, storage, search, retrieval, and use" (Jennex, 2006, p.3). Knowledge discovery is generally one of the important stages or phases of KM. And while this incorporates identifying critical knowledge (this may also be what this stage is called), using data mining to aid in knowledge discovery is appropriate as being a useful KM tool.

Data mining is a promising tool that assists companies to uncover patterns hidden in their data. These patterns may be further used to forecast customer behavior, products and processes. It is important that managers who understand the business, the data, and the general nature of the analytical methods are involved. Realistic expectation can yield rewarding results across a wide range of applications, from improving revenues to reducing costs (Davenport & Harris, 2007; Porter & Miller, 2001). It is crucial to properly collect and prepare the data, and to check the models against the real figures. The best model is often found after managers build models of several different types or by trying different technologies or algorithms. This alone demonstrates the active role managers play in the data mining or other knowledge management processes.

Selecting software is a practical yet very important problem for a company (James, Hakim, Chandras, King, & Variar, 2004). However, not enough attention is given to this critical task. Current literature is quite limited because selecting software is such a complex problem, due to many criteria and frequent technology changes (Elder IV & Abbott, 1998; Giraud-Carrier & Povel, 2003). Haughton, Deichmann, Eshghi, Sayek, Teebagy, and Topi (2003) generally reviewed several computer *software packages* for *data mining, including* SPSS Clementine, XLMiner, Quadstone, GhostMiner, and SAS Enterprise Miner. Corral, Griffin, and Jennex (2005) examined the potential of knowledge management in data warehousing from an expert's perspective. Jennex (2006) introduced technologies in support of knowledge management systems.

Firstly, this article will take a brief look at data mining today, through describing some of the opportunities, applications and available technologies. We will then discuss and analyze several of the most powerful data mining software tools available on the market today. Ultimately, we will also attempt to provide an analytical analysis and comparison among the brands we have selected. Our selection is based, in part, on our own experience using data mining software as well as writing data mining code, SQL code and our work as relational database administrators. For our analytical comparison we will be using *Expert Choice* (Version 11) advanced decision support software.

DATA MINING SOFTWARE

Data mining software analyzes- based on open-ended user queries- relationships and patterns that are stored in transaction data. Available are several types of analytical software: statistical, machine learning and neural networks, decision trees, Naive-Bayes, K-Nearest Neighbor, rule induction, clustering, rules based, linear and logistical regression time sequence, and so forth. Along the lines of Mena (1998) and Martin (2005), the basic steps of data mining for knowledge discoveries are:

1. Define business problem
2. Build data mining data base
3. Explore data
4. Prepare data for modeling
5. Build model
6. Evaluate model
7. Deploy model
8. Results

Note: Each of these steps contains managerial issues which must be addressed.

The key to knowledge discovery is a true understanding of your data and your business. Without this understanding, no algorithm is going to provide you with a result in which you should confide. Moreover, without this background you will not be able to identify the problems you are trying to solve, prepare the data for mining, or correctly interpret the results. There are many tasks involved in the construction of a database: data collection, data description, selection, data quality assessment and data cleansing, consolidation and integration, metadata construction, and maintaining the database. In exploring the data, the manager must choose the appropriate hardware to accomplish this feat. The goal is to identify the most important fields in predicting an outcome, and determine which derived values may be useful. According to O Chan (2005), a good interface and fast computer response are very important in this phase because the very nature of your exploration is changed when you often have to wait up to 30 minutes for some graphs to be created.

Preparing data for modeling consists of four main parts: selecting variables, selecting rows, constructing new variables and transforming variables. The managerial decision in this case focuses on identifying key variables to examine, nonfully functional variables inclusive. The time it takes to build a model increases with the number of variables while blindly including extraneous columns can lead to incorrect models. The most important thing to remember about data model building is that it is an interactive process. Many alternative models may have to be examined to find one that is most appropriate in solving your business problem. A manager searching for a good model may go back and amend the data he or she is using or even modify his or her problem statement. In the evaluation and interpretation process, the accuracy rate found during testing applies only to the data on which the model was built. The accuracy may vary if the data to which the model is applied differ in important and unpredictable ways from the original data set.

Once a data mining model is built and validated, it can be used in two main ways. First, the manager may recommend actions based on simply viewing the model and its results. For example, the manager may look at the clusters the model has identified, the rules that define the model or the lift and ROI charts that depict the effect of the model. The second process involves applying the model to different datasets. The manager may use the model to flag records based on their classification or assign a score such as the probability of an action.

Data Mining Software Alternatives

As stated earlier in our introduction, there are numerous data mining software alternatives that vary in the number of modeling and visualization nodes as well as in price. We have elected the following eight software vendors for comparison due to a limitation of trial version of *Expert Choice* (Version 11):

- Clementine from SPSS
- DB2 Intelligent Miner from IBM
- Enterprise Miner from SAS
- GhostMiner by Fujitsu
- Insightful Miner V5.2 for Insightful
- Megaputer PolyAnalyst
- Microsoft SQL Server 2005 Enterprise Edition
- Oracle Data Miner

Although there are various other comparable programs available, we were limited in our selection. One of the limiting factors was the inadequacy of alternatives in our decision tools' evaluation copy.

Decision Tool

To aid in comparing our software choices, we used an evaluation copy of *Expert Choice* version 11, a leading software solution construed to analyze, categorize, prioritize, select, allocate and choose a selection based on relevant criteria (Expert Choice Inc, 2007). *Expert Choice* incorporates a process known as *Analytical Hierarchical Process* (AHP) into its software (Saaty & Vargas, 2006; Saaty, 1980, 1996, 2001, 2005). Research has demonstrated that AHP is a powerful decision-making tool that can help organizations avoid making costly mistakes caused by bad decisions (Hemaida & Schmits, 2006). AHP was developed by Saaty and Kearns and consists of four stages (Roper-Lowe & Sharp, 1990). The first stage is to construct a hierarchy where the primary objective, or goal, is at the highest level. Criteria, which can also be subdivided, follow in decreasing order. At the bottom of the hierarchy are the alternatives to be evaluated. The second stage calculates weights for the criteria using pair-wise comparisons. In the next stage, the alternatives are also compared to each other in respect to each criterion. Finally, all weighted scores are tallied to yield a final score. The alternative with the highest score is considered the best alternative.

SAMPLE PRODUCTS

In this section, we will analyze the various features and benefits offered by each of our software alternatives that are to be considered, as well as researching product reviews and professional opinions. The information gathered from this analysis will serve as the basis for the pair-wise comparisons of the software alternatives with respect to each of our criteria In other words, we will investigate how important one choice is over the other alternative given a specific criterion, when comparing any two software alternatives,

CART by Salford Systems

CART is an easy to use decision tree tool that uses the CART algorithm and boosting. Its main objective is to rifle through databases, identifying significant patterns and relationships, which are then used to generate predictive models. CART uses an exhaustive, recursive partitioning routine to generate binary splits by posing a series of yes-no questions. It searches for questions that split nodes into relatively homogenous child nodes. As the tree evolves, the nodes become more homogenous, identifying segments. CART supports more than 80 file formats, including SAP, SPSS databases such as Oracle and Informix, Excel spreadsheets, and Lotus 1-2-3 spreadsheets.

CART was formulated from the original CART code developed by Stanford University and University of California at Berkeley statisticians. The frequent addition of new features and capabilities continually enhances the procedure, strengthening the accuracy and reliability of the results. CART has a no-stopping rule, which makes it unique. This means that more data are read and compared, and it ensures that important data are not overlooked by stopping too soon. It produces an over-grown tree, and immediately prunes it back for the most optimal results. CART also uses a powerful binary split search approach. This means the trees are more sparing with data and detect more structure before too little data are left for learning. Next, CART uses automatic self-validation procedures, which are essential in avoiding the trap of finding patterns that apply only to the training data.

CART was designed for both technical and nontechnical users. It can quickly identify important data relationships. It offers users some flexibility, with the choice of how to split criteria. It also offers different models for scalability. The results are easy to read and understand, with decision tree diagrams drawn out. Salford Systems understands that expert and timely technical

support is a critical part of the business, which is why they offer many means of customer training and support. The company offers both public and private on-site instruction, user seminars, hand-on training courses, consultation services, and e-mail, NetMeeting, and phone support from offices worldwide.

SPSS—Clementine

Clementine data mining software by SPSS is useful for organizations using SPSS infrastructure as well as those with mixed platforms. This program supports both client and server platforms, including the Windows family of products and Sun Solaris, HP-UX11i, IBM AIX, and OS/400 server platforms.

With regard to reliability, Clementine by SPSS supports decision trees, neural networks, regression, self-organizing maps, clustering, and association rules (Lampe & Garcia, 2004). However, the author states that Clementine implements "a broad set of statistical algorithms, but fewer than in the SAS and IBM packages" (Lampe & Garcia, 2004, p. 18).

In the area of efficiency, Clementine works with SPSS, SAS and SQL and can export to C code and Predictive Model Markup Language (PPML) (Angus, 2006). It can also handle critical data preparation, rapid modeling, and model scoring tasks. These tasks are all performed using GUI graphical layouts, workflow diagrams, scatterplots, distribution, histogram, multiplot and Web charts (which are unique to Clementine (Haughton et al., 2003)).

Training and support for Clementine are available from SPSS in the form of online tutorials, downloadable overview and demos of the program, along with online technical support and excellent help screens. The price of Clementine starts at $75,000 (Angus, 2006).

Enterprise Miner by SAS

Enterprise Miner was developed by SAS Corporation, which was originally called Statistical Analysis System. Enterprise Miner is an integrated software product that provides widespread business solutions for data mining based on SEMMA (Sample, Explore, Modify, Model, Assess) methodology. It has many different statistical tools including decision trees, clustering, linear and logistic regression and neural networks. Data preparation tools include outlier detection, variable transformations, random sampling, and the partitioning of data sets into training, test, and validation data sets. Its advanced GUI allows you to review large amounts of data in multidimensional histograms with ease, as well as compare modeling results graphically.

Enterprise Miner includes several procedures that automate traditional analysis tasks, such as choosing the variables to include in a model and applying the appropriate transformations. The system also provides extensive visualization to help users explore the data to decide on additional manipulations that the system itself does not recommend. Enterprise Miners' graphical user interface and automated framework indicate that the user does not have to know how the tools work to use them. Release 8.2 provides cross-platform national language support that is especially important to international customers.

GhostMiner by Fujitsu

GhostMiner is a data mining software product from Fujitsu that not only supports common databases (or spreadsheets) and mature machine learning algorithms, but also assists with data preparation and selection, model validation, multimodels (like committees or k-classifiers), and visualization. GhostMiner provides a large range of **data preparation** techniques and a broad

scope of **selection of featured** methods. Choice of data mining algorithms and v**isualization techniques** are integrated.

GhostMiner offers several project features unique to their platform, which enables users to create simple interfaces for their specific needs. GhostMiner has a human machine interface (HMI) that is fairly user friendly and easy to start up right out of the box. The system is so easy to use that it actually has a feel of being too user friendly and may be missing some of the power of the larger server based data mining software tool such as Darwin IBM and Oracle. GhostMiner can be loaded directly onto a Windows based PC and is equally adept at data mining a system database as it is a series of spreadsheets, text or ASCII files.

GhostMiner contains both data preprocessing capabilities as well as data visualization capabilities. Data preprocessing includes data normalization, standardization and many preliminary statistical analysis functions, such as variance, standard deviation, mean and median across the entire database. GhostMiner does not have the inherent flexibility of some of the larger, more robust products, and also does not offer the same levels of support as the products offered by Oracle and IBM. GhostMiner is a product marketed to small to mid size users who are looking for a simple to use product at a lower price than some of its larger, well known counterparts.

Insightful-Insightful Miner

Insightful Miner is a cost effective data mining software program. The software has numerous model types, algorithms and visualizers, including decision trees, *Block Model Averaging*, linear and logistic regression, neural networks, Naïve Bayes, Cox proportional hazard models, K-means clustering and others. Insightful Miner offers highly scalable algorithms, which train models on very large data sets without the need for sampling or aggregation. Insightful also offers data preprocessing and data cleansing as well as exploratory data analysis and visualization. Insightful Miner's cost is typically $12,000.

According to a product review in *DM Review* (Lurie, 2004) the main strength of Insightful Miner is its ability to scale large data sets in an accessible manner. It provides the analytic tools required to transform fragmented raw data into "actionable knowledge" (Lurie, 2004, p. 88). Insightful Miner provides cutting edge analytics and reporting tools to identify patterns, trends and relationships in data. Insightfuls' simplicity allows users to quickly aggregate, clean and analyze data. Its powerful reporting and modeling capabilities allow users to deliver clear, usable analytics to designers and producers. Simple visual work maps make it easy for users to become productive relatively quickly. Insightful Miner provides excellent product support and its documentation is complete and easy to understand.

In another product review of Insightful Miner (Deal, 2004), the software was found to be a comprehensive data mining application that includes extensive data input, data manipulation, and analysis capabilities. Insightful Miner can efficiently process large amounts of data by using a chunking and processing algorithm that is intended to be scalable to the mass of data used for each analysis. Insightfuls' ability to integrate S-Plus strengthens and extends its functionality. Deal (2004) stated that Insightful Miner "is a very simple and intuitive process" (p. 46).

IBM-Intelligent Miner

IBM Intelligent Miner V7R1 is very user friendly software. It is an essential e-commerce tool, as it can aid in handling transactions as they come in. It has business intelligence applications, which allow it to make decisions that would be good for any business, large or small. The intelligence part of the software could cut costs and increase profits. The data screens help with decision making and improvement on processes that are out of

date. It also maximizes the business to customer relationship, because of the personalization the software can provide for each client. This software package is also compatible with Windows, AIX, Solaris and Linux servers.

IBM Intelligent Miner V7R1 has IM Scoring, in which the user has an advantage, because scores and ranks are done in real time. This means that as a new transaction takes place, it would then reorganize the scores/ranks of the customers' information. For example, when a customer buys an item off the Internet, the software would update for the payment due and when it should be posted. The same would apply for a dentist visit: after 6 months it would indicate that it is time for another checkup for the particular patient. As the appointment would approach, the higher the person would be on the list, that is, moving up the ranks. Another advantage with the IM Scoring is the high performance and scalability of mining functions, thus making sampling obsolete.

The best aspect of this product is its user friendliness. The whole staff would be content with it. It can also be updated easily, without any disruption to the business. IBM is currently promoting the DB2 Query Management Facility version 8.1, because in March 2006, IBM withdrew Intelligent Miner from all marketing and ended all its support for Intelligent Miner tools.

Megaputer-PolyAnalyst

Another data mining software package is PolyAnalyst (the newest version is 4.6) made by the Megaputer Company. This company is quite small, especially compared to some of the other companies we have profiled. Megaputer Intelligence Inc. is a leading developer and distributor of advanced software tools for data mining, text mining, and intelligent e-commerce personalization. The tools help reveal knowledge hidden in data. They add intelligence and insight to every step of the business decision-making process.

Because the Megaputer Company focuses primarily on data mining programs, they can offer a more comprehensive program than other companies who simply have a data mining component to existing products. They offer a vast array of algorithms from which a consumer can choose the ones they need specifically, making the product ready to be customized. The price for an older version of PolyAnalyst (the most recent pricing data found) is an affordable $2,300 for the base version and can go up to $14,900 with all the algorithms. Also, the developer kit for PolyAnalyst is available for $16,000.

PolyAnalyst can be run either on a stand-alone system or in a client/server configuration, where the server would handle the data processing. It only works with the Microsoft Windows O/S, which shows that it is not as portable as some other products analyzed. Also, Megaputer offers possible users a free evaluation version to decide if this is the software right for them. The program offers a rich set of features. PolyAnalyst by Megaputer seems to be a feature rich data mining software package. The price and ala carte feature set seem more suited for a smaller company that cannot afford to use a more expensive data mining solution that would require the use of highly trained employees.

Oracle–Oracle Data Mining

Oracle Data Miner is the graphical user interface for Oracle Data Mining (Release 10.1 and above) that helps data analysts mine their Oracle data to find valuable hidden information, patterns, and new insights. Oracle Data Mining is a powerful data mining software embedded in the Oracle Database that enables you to discover new information hidden in your data and helps businesses target their best customers and find and prevent fraud.

Oracle provides unique portability across all major platforms including Windows, Solaris, HP-

UX, IBM AIX, Compaq Tru64, and Linux and ensures that applications run without modification after changing platforms. There are two common ways to architect a database: client/server or multitier. Two basic memory structures are associated with Oracle software: the system global area and the program global area.

Oracle Data Miner facilitates interactive data preparation, data mining model creation, evaluation, refinement and model scoring. Oracle Data Mining provides the following supervised data mining algorithms: Naïve Bayes, Adaptive Bayes Network, decision trees, Support Vector Machines, and attribute importance. Unsupervised algorithms are: clustering, association rules, feature selection, anomaly detection, text mining and unstructured data, and life sciences algorithm. Mining Activity Guides provide structured templates for all users to explore and mine their data.

Oracle Data Mining (ODM) enables companies to extract information efficiently from the very largest databases, and build integrated business intelligence applications and support data mining problems such as: classification, prediction, regression, clustering, associations, attribute importance, feature extraction and sequence similarity searches and analysis. When the capabilities of Oracle Data Mining are combined with the ability of the RDBMS to access, preprocess, retrieve and analyze data, they create a very powerful platform for data analysis.

Oracle Data Mining can generate valuable new insights and reports that can help proactively manage your business, according to the Oracle Discoverer report. Oracle Data Miner models can be visualized graphically and can be display in tables, histograms, line graphs and pie graphs. Data may be in either Excel or the Database. Significant productivity enhancements are achieved by eliminating the extraction of data from the database to special-purpose data mining tools (Berger & Haberstroh, 2005).

Data size is unlimited. The expert analyst can adjust some or all of the parameters manually, but the option to allow the algorithms to optimize the parameters intelligently, with no intervention, is available. There are free demos available: Oracle Data Mining, Integration with Oracle BI EE, Spreadsheet Add-in for Predictive Analytics, and Text Mining. The tutorial Oracle by Example series and online training provides valuable hands-on experience, step-by-step instructions on how to implement various technology solutions to business problems. Oracle Data Mining significantly reduces the cost of data mining. Savings are realized in the avoidance of additional hardware purchases for computing and storage environments, redundant copies and multiple versions of the data and duplication of personnel who perform similar functions. Database analytics includes: engine, basic statistics (free), data mining, and text mining.

SQL Server 2005

SQL server 2005 is Microsoft's solution to database management and data mining. SQL Server database platform provides enterprise-class data management with integrated business intelligence (BI) tools. SQL Server 2005 combines analysis, reporting, integration, and notification. SQL server is closely integrated with Microsoft Visual Studio, the Microsoft Office System, and a suite of new development tools, including the Business Intelligence Development Studio (Bednarz, 2005).

Microsoft SQL Server series utilizes the Windows operating system and features four discrete algorithms. HMI features include a Windows' interface, as well as complete integration with the Microsoft Office suite. Reports that are served by the Report Server in Reporting Services can run in the context of Microsoft SharePoint Portal Server and Microsoft Office System applications such as Microsoft Word and Microsoft Excel (Fontana, 2005). SharePoint can be used to subscribe to

reports, create new versions of reports, and distribute reports. SQL Server 2005 also supports rich, full-text search applications. Query performance and scalability have been improved dramatically, and new management tools will provide greater insight into the full-text implementation.

SQL Server also features an online restore function, database encryption and a fast recovery option. It also has a system with built-in scalability features such as parallel partition processing, creation of *remote relational online analytical processing* (ROLAP) or *hybrid online analytical processing* (HOLAP) partitions, distributed partitioned cubes, persisted calculations, and proactive caching.

COMPARISON

We use *Expert Choice* in the evaluation process and will attempt to analytically quantify the aspects of data mining software that best define overall product quality. Before we describe the decision making process, we would like to present several assumptions on which our decision will be based:

1. In addition to our experience, we will rely on manufacture specifications, descriptions and described attributes, along with third party reviews where available.
2. We will base our needs on fundamental business goals such as business-related decision making and business-driven information analysis. Although this definition may seem overly broad, we will attempt to further limit our scope by eliminating research and development, educational and political as well as most human resource applications.
3. Because we are using a trial version of *Expert Choice* advanced decision making software, there will be limits with respect to importing and exporting data as well as with printing and possibly some advanced analytical tools. Therefore, we will utilize screen captures embedded into this document, and will manually write any necessary data as opposed to systematic imports or exports.

Criteria Revisited

Our selection process will be centered on the below mentioned software quality criteria. We will attempt to compare all of our selections based on the specified criteria. Using *Expert Choice*, we will make objective ratings of each product, comparing in a pair-wise manner, attributes that define each element.

- **Portability:** the amount of platform independence; the number of support platforms and supported software architectures as well as any software requirements needed to run the software.
- **Reliability:** the degree of completeness, accuracy and consistency, any stated warranty and support provided by the vendor. The number of data models and algorithms available with the software as well as any templates or custom models available for creation of projects.
- **Efficiency:** the degree of efficiency and accessibility; the degree in which the product supports the general business goal assumptions and the number of tools available for data preprocessing.
- **Human engineering:** how well the software interfaces and communicates with the outside world, plus the quality of the human machine interface (HMI). Testability – how well the software is structured; how results are displayed and how results are reintroduced into the process if applicable.
- **Understanding:** degree of self-descriptiveness; the degree of simplicity of the machine

interface, the use of graphical user interfaces, visual programming ability, summary reports, and data model visualization.

- **Modifiability:** the degree of augmentation ability and the ability to change over time and expand; the use of batch processing and any expert options as well as data size limitations.
- **Price, training and support:** price of product, availability of evaluation or demo versions, and the amount of post purchase support included in the package.

Evaluation Model

Our evaluation criteria, as entered in the Expert Choice, are as follows:

- Portability: evaluated in terms of:
  - Hardware platform (PC, Unix/Solaris workstation, etc.).
  - Software Architecture (standalone, client/server, thin client).
  - Software requirements (DB2, SAS, Base, Java/JRE, Oracle, and so forth.
- Reliability - evaluated in terms of:
  - What model classes does the tool support?
  - How many algorithms does the tool use?
  - Does the tool allow custom model creation or simply uses templates?
  - What is the reputation of the vendor supplying the tool?
- Efficiency evaluated in terms of:
  - How well does the product support our general business goal assumption?
  - Ability to perform data preprocessing.
- Human Engineering evaluated in terms of:
  - Simplicity of HMI (human machine interface)
  - Graphical layout
  - Visual programming ability
- Testability evaluated in terms of dissemination and deployment:
  - How well the results are reintroduced into the process "closing the loop"
  - How results are displayed
- Understanding in terms of evaluation and interpretation of data:
  - Are summary reports available?
  - Can the model be visualized graphically?
- Modifiability in terms of scalability and upgrades:
  - What is the data set size limit?
  - Are there expert options or batch processing?
- Training and support evaluated in terms of:
  - Is a free demo available?
  - Is any free training or support available with the purchase?
- Price (where available) – if pricing is not available we will note our evaluation as price neutral.

PROCEDURE OF EXPERT CHOICE

Let's use five products to demonstrate the whole process of *Expert Choice* on a small scale. We commence with pair-wise comparisons for each of our criteria. Figure 1 is a screen capture of our initial results of priorities. As can be seen in Table 1, we placed a great deal of importance on Human Engineering (weight of .220), slightly less on Training and Support (w=. 193) and then on Understanding (w=. 190). Our main driver was that for the software to be successful, people had to know and understand it. Our next highest priority was Reliability, with a relative weight of .142, followed by Portability, which is platform and hardware independence, with a relative weight of .128.

Next, we perform a pair-wise comparison of each software tool for each criterion; that is, we compare the components of each criterion on a case-by-case basis, assigning relative strengths and weaknesses to each product. Although this process is quite tedious, it provides an accurate measurement for each product. Table 2 shows an example of a pair-wise comparison for the contribution of hardware platform independence to overall platform independence, which is a contributor to overall portability within our quality structure.

Table 3 shows a graphical representation from the pair-wise comparison between all products for the hardware contribution, to overall portability. The screen capture shows a weight for each product with SQL Server as the best in class (with a weight of .269) and GhostMiner as last in class (with a weight of .109). These criteria are also weighted individually so as to roll up into the overall contribution toward portability.

Table 1. Weights assigned to criteria

Category	Priorities
Human Engineering	0.22
Training and Support	0.193
Understandability	0.19
Reliability	0.142
Portability	0.128
Modifiability	0.051
Efficiency	0.039
Testability	0.022
Price	0.016

Overall Results

Table 4 shows the overall results from *Expert Choice* advanced decision support software. From the first iteration of our selection process, the best solution for our chosen attributes and assigned priorities is the CART product, with an overall weight of .268, followed by SAS Enterprise with an overall weight of .223. We also performed several iterations, changing the weights of our criteria.

Table 5 shows the assigned weights of each category along with the overall score for all of the objects. This tool allows dynamic sensitivity

Table 2. Pair-wise comparison grid WRT hardware platform

	CART by Salford Systems	SAS Enterprise Miner	Oracle 8i	GhostMiner	SQL Server 2005
CART by Salford Systems		3.0	3.0	3.0	7.0
SAS Enterprise Miner			2.0	3.0	4.0
Oracle 8i				3.0	3.0
GhostMiner					4.0
SQL Server 2005					
Inconsistency: 0.73					

Table 3. Class weightings for overall hardware platform independence

Vendor	Class Weighting
CART by Salford Systems	0.268
SAS Enterprise Miner	0.223
Oracle 8i	0.215
GhostMiner	0.141
SQP Server 2005	0.152
Overall inconsistency: 0.28	

Table 4. Results from "Choosing a data mining software vendor"

Vendor	Overall Weight
CART by Salford Systems	0.191
SAS Enterprise Miner	0.215
Oracle 8i	0.222
GhostMiner	0.109
SQP Server 2005	0.263
Inconsistency: 0.73	

Table 5. Dynamic sensitivity analysis

Category	Category Weight	Vendor	Vendor Preference Weight
Portability	12.8%	CART by Salford Systems	26.8%
Reliability	14.2%	SAS Enterprise Miner	22.3%
Efficiency	3.9%	Oracle 8i	21.5%
Human Engineering	22.0%	GhostMiner	14.1%
Testability	2.2%	SQL Server 2005	15.2%
Understandability	19.0%		
Modifiability	5.1%		
Training and Support	19.3%		
Price	1.6%		

analysis with respect to changing priorities. We used this tool to look at how much a change in one weight changes the overall goal. Using this tool is similar to the sensitivity analysis performed in Excel Solver; however, instead of listed ranges the Expert Choice tool allows for dynamic manipulation. From the chart, we can see our weighted emphasis on Human Engineering (22%), Training

Table 6. Dynamic sensitivity analysis with different constraints

Category	Category Weight	Vendor	Vendor Preference Weight
Portability	0.1%	CART by Salford Systems	24.0%
Reliability	2.6%	SAS Enterprise Miner	18.5%
Efficiency	3.2%	Oracle 8i	17.1%
Human Engineering	5.8%	GhostMiner	20.8%
Testability	3.3%	SQL Server 2005	19.6%
Understandability	21.5%		
Modifiability	4.1%		
Training and Support	15.7%		
Price	33.5%		

and Support (19.3%) and Understandability. The window on the right shows which system best fits our stated criteria.

In Table 6 we change our requirements in order to verify the strength of our decision. We increase the importance of Price from 1.6% all the way up to 33.5%. We also change our Human Engineering requirement from 22% down to 5.8%, and also reduce Training and Technical Support, Portability, Reliability, Efficiency and Modifiability (flexibility) substantially and still came up with CART systems as our best overall choice (24% weight).

SCENARIO ANALYSIS

We now start to compare eight leading data mining packages based on seven criteria. Determining the best software is a multiple objective decision-making process because different companies may have completely different needs. An array of software may each be the best choice because their design and performance are defined within a certain type of institution. Usually, one data mining software cannot be the best for every scenario. This is because specific software cannot meet the expectations of every type of institution; therefore, the creation of scenarios is a very important tool in term of decision-making process.

In order to make this project more accurate and realistic, we have combined different scenarios. Factors, such as size, budget, type of business, and the type of data we have to manipulate, can affect the software we attempt to choose and the reasons why we choose it. Having simulated many scenarios, we found that size is a decisive factor. For instance, if we choose two companies with different sizes and follow a traditional reasoning, the tentative result may contradict with our intuition.

After researching all the alternatives, we used our decision tool, *Expert Choice*, to make pairwise comparisons for each of them. A total of 392

pair-wise comparisons were required to compare each of the alternatives with respect to each of the criteria (196 comparisons for each scenario). This was in addition to the 42 pair-wise comparisons required to assign weights to each criterion with respect to the goal (21 comparisons for each scenario). After completing all of the pair-wise comparisons the software synthesized all of the weights of the alternatives with the weights of the criteria and selected the best alternative of each of the two scenarios. Table 7 below summarizes the weights of the pair-wise comparisons for each of the alternatives with respect to each criterion. As with the weights of the criteria, there is also a direct relationship between the calculated weights in each column and the respective criterion. In other words, the higher a number is in a given column the more important that sized company views that particular software for that specific criterion.

Based on our research of all the alternatives and the weighted criteria calculated by our decision tool, the software has determined that for a small-sized company the top three alternatives are Insightful Miner, Megaputer PolyAnalyst, and SAS Enterprise Miner. These results were somewhat unexpected. We had anticipated that Insightful, Megaputer, and GhostMiner would

Table 7. Weights assigned to each alternative for both a small and large-sized company

Synthesized Weights - with respect to criteria	Efficiency		Human Engineering\ Understandability		Modifiability		Portability	
	Large	Small	Large	Small	Large	Small	Large	Small
Clementine	0.213	0.153	0.154	0.154	0.159	0.159	0.200	0.059
Enterprise Miner	0.176	0.174	0.155	0.155	0.151	0.151	0.221	0.059
GhostMiner	0.110	0.102	0.124	0.124	0.069	0.069	0.069	0.235
Insightful Miner	0.142	0.168	0.135	0.135	0.108	0.108	0.097	0.235
Intelligent Miner	0.106	0.130	0.112	0.112	0.155	0.155	0.097	0.059
Megaputer	0.086	0.091	0.090	0.090	0.060	0.060	0.067	0.235
Oracle Data Miner	0.102	0.103	0.104	0.104	0.151	0.151	0.148	0.059
SQL Server 2005	0.066	0.078	0.126	0.126	0.147	0.147	0.079	0.059

Synthesized Weights – continued	Reliability		Training and Support\Price		Testability	
	Large	Small	Large	Small	Large	Small
Clementine	0.194	0.194	0.040	0.036	0.156	0.156
Enterprise Miner	0.206	0.206	0.027	0.025	0.149	0.149
GhostMiner	0.069	0.069	0.170	0.194	0.116	0.116
Insightful Miner	0.107	0.107	0.261	0.266	0.147	0.147
Intelligent Miner	0.110	0.110	0.029	0.024	0.099	0.099
Megaputer	0.147	0.147	0.261	0.349	0.151	0.151
Oracle Data Miner	0.101	0.101	0.108	0.051	0.096	0.096
SQL Server 2005	0.066	0.066	0.104	0.054	0.086	0.086

be among the top three or four alternatives for a small-sized company, primarily because each of these vendors offer stand-alone versions of their software and are also the least expensive among all the alternatives. However, GhostMiner ranked lower than expected while SAS Enterprise Miner ranked higher. A closer analysis of the pair-wise comparisons shows that SAS was more efficient and had better human engineering than GhostMiner. Because of the weights given to these two criteria, SAS Enterprise Miner was able to beat all of the other alternatives despite its cost.

According to our decision tool, the top three alternatives for a large company are SPSS Clementine, followed by SAS Enterprise Miner and Insightful Miner. There was not one overwhelming choice for a large company. The differences in weight among the top three alternatives were relatively small (16.7%, 15.9%, and 13.5%, respectively). These results were also a little unanticipated. All of the software reviews that we have read rated SPSS and SAS among the top leading data mining software that are commercially available. We had not expected Insightful Miner to rank among the top three alternatives for a large company. We anticipated that Microsoft, IBM, or Oracle would round out the top three alternatives because these vendors offer enterprise class DBMS. Upon closer examination of the software analysis, Insightful Miner's visualization and modeling nodes were comparable to those of an enterprise class data mining software program such as Enterprise Miner and Clementine. In addition, a closer review of the pair-wise comparisons showed that Insightful Miner ranked higher than the other alternatives (with the exception of SAS and SPSS) in human engineering and efficiency. It also tied for top weight for Training and Support/Price. Table 8 below summarizes the results of the Expert Choice software.

Certainly, different companies may have different priorities, preferences, and prerequisites. We have explored a few individual scenarios.

Special Case 1: This is a large international company, with thousands of employees, doing business between U.S., Mexico and other countries in Latin America. It is an import and export company. The main goal of using data mining software is to determine the best distribution methods in order to maximize profits. Keeping costs down so it can compete with other companies is always a concern. The three main criteria this company is looking for in a software package are: Training and Support with multilingual support because of the language difference between the countries,

Table 8. Summary of synthesized results for each sized company

Synthesized Weights - with respect to goal	Large Company	Small Company
Clementine	0.167	0.138
Enterprise Miner	0.159	0.140
GhostMiner	0.102	0.123
Insightful Miner	0.135	0.160
Intelligent Miner	0.112	0.104
Megaputer	0.106	0.146
Oracle Data Miner	0.115	0.098
SQL Server 2005	0.102	0.092
Overall inconsistency ratio	**0.010**	**0.020**

Human Engineering to assure that employees in different countries with possibly different computer skill levels, will be able to adapt and use the software, and Portability because it is an established company with an IT department and different platforms that include Microsoft, IBM and Sun Microsystems, as well as a range of desktop operating systems that includes Windows 2000 and XP, Linux, and old legacy equipment. They must be certain that the software is compatible with all existing platforms. Because large amounts of data are processed, software must be robust and reliable as well.

- **Our goal is:** To find the best data mining software.
- **Our criteria are:** Listed seven factors. The three most important criteria are Training and Support, Human engineering, and Portability.
- **Our alternatives are:** Clementine, Enterprise Miner, Oracle, Microsoft SQL, IBM DB2, Salford CART, Megaputer, and Insightful Miner.

Criteria	Weights
Portability	.215
Modifiability	.181
Training and Support	.147
Human Engineering/ Testability	.130
Understandability	.119
Reliability	.109
Efficiency	.100

Software	Ranking
Clementine	.136
Enterprise Miner	.126
Oracle	.162
Microsoft SQL	.109
IBM DB2	.117
Salford CART	.123
Megaputer	.111
Insightful Miner	.117

Special Case 2: This is a large national corporation, between 500 and 1,000 employees, in the retail industry with many branches throughout the country. They already have an IT department and different types of platforms, including Unix Servers and Microsoft 2000 and 2003, as well as XP and 2000 Workstations. Portability is very important to make sure that the software is able to run with the platforms already in place. This company already has a well-established customer base, so the goal of choosing data mining software is to find the best way to maximize customer retention, while lowering costs. The three most important criteria that this company is looking for in a software package are: Portability, Efficiency to assure it supports the general business goal assumption, and Modifiability because it is a growing business, and they want to be sure that they can go back and customize the software if necessary.

Ideal mode	PAIRWISE	PAIRWISE	PAIRWISE	PAIRWISE	PAIRWISE	PAIRWISE	PAIRWISE
Alternative	**Portability (L: .055)**	**Reliability (L: .145)**	**Efficiency (L: .130)**	**Human Engineering/ Testability (L: .201)**	**Understandability (L: .156)**	**Modifiability (L: .076)**	**Training and Support (L: .226)**
☑ Clementine	.793	.819	.806	.974	.988	.671	.861
☑ Enterprise Miner	.684	.725	1.000	.920	.944	.625	.677
☑ Oracle	1.000	1.000	.993	.984	.946	1.000	1.000
☑ Microsoft SQL	.236	.763	.862	.857	.951	.444	.968
☑ IBM DB2	.727	.806	.796	.714	.923	.389	.814
☑ Salford CART	.527	.738	.855	.783	1.000	.777	.767
☑ Megaputer	.360	.529	.268	1.000	.925	.826	.898
☑ Insightful Miner	.371	.607	.473	.858	.974	.923	.899

- **Our goal is:** To find the best data mining software.
- **Our criteria are:** Listed seven factors. The three most important criteria are portability, efficiency, and modifiability.
- **Our alternatives are:** Clementine, Enterprise Miner, Oracle, Microsoft SQL, IBM DB2, Salford CART, Megaputer, and Insightful Miner.

Criteria	Weights
Modifiability	.217
Portability	.171
Efficiency	.153
Reliability	.132
Training and Support	.126
Human Engineering/ Testability	.123
Understandability	.078

Software	Ranking
Clementine	.134
Enterprise Miner	.127
Oracle	.163
Microsoft SQL	.110
IBM DB2	.115
Salford CART	.125
Megaputer	.108
Insightful Miner	.117

Special Case 3: This is a small start-up landscaping and construction company with less than 50 employees. The employees have limited knowledge of computers and software. The goal of using data mining software is to find the best way to attract new customers.

- **Our goal is:** To find the best data mining software.
- **Our: criteria are:** Listed seven factors. The three most important criteria we are looking for are Human engineering, Training and Support, and Understandability.
- **Our alternatives are:** Clementine, Enterprise Miner, Oracle, Microsoft SQL, IBM DB2, Salford CART, Megaputer, and Insightful Miner.

Criteria	Weights
Training and Support	.226
Human Engineering/Testability	.201
Understandability	.166
Reliability	.145
Efficiency	.130
Modifiability	.076
Portability	.055

Software	Ranking
Clementine	.132
Enterprise Miner	.124
Oracle	.150
Microsoft SQL	.124
IBM DB2	.117
Salford CART	.122
Megaputer	.114
Insightful Miner	.118

Other Cases

Online Company/E-commerce

An online/e-commerce company in the recently growing industry: First of all, because an Internet company has both actual and potential customers, it needs a tool that can hold and analyze large amounts of data. Secondly, it might have engineers or a technical department, so it may not put weight on human engineering and training and support. Consequently, we put more weight on modifiability and less weight on engineering and support. As a result, Oracle would be the best tool for an online/e-commerce company because it scores the highest among eight tools. If Oracle is not available, IBM would be the second choice.

Educational Institutions

Data mining software is used worldwide in the educational industry. One of Megaputer's data mining software called PolyAnalyst gets a majority of its business from educational industry. Microsoft SQL came in first place with Ghost-Miner as the runner up.

Even though these scenarios can be used as references, they did not apply to every type of institution. Thus, it will be interesting to see what other choices are available in term of the best data mining software. What would be the best data mining software for a medical institution?

CONCLUSION

With the use of *Expert Choice* we were able to analytically evaluate eight products within a complex yet controlled environment. The detailed analysis included prioritizing our constraints, evaluating the contributing criteria, entering comparative data and performing relevant sensitivity analysis. The software, *Expert Choice,* performed the analysis, based on our definition, priorities and data.

Data mining technology is changing very rapidly. Our article focused only on the major suppliers typically available in the market place. There is no definite and explicit answer as to which tool is better suited to potential clients, mainly due to their unique priorities. As there are so many variables to quantify, the problem needs to be defined. Based on what approach the problem requires, then and only then can tools start being quantified. Certainly, the method and the process that we have used can be easily applied to analyze and compare other data mining software for each potential user. Although there is no pattern for pairing the correct software with the proper institution, with the use of this process, every institution should be able to determine which data mining software is the best for their operations.

ACKNOWLEDGMENT

We would like to thank Dr. Jennex, the *Edit-in-Chief of IJKM*, for his tremendous help and guidance during the period of revising our manuscript for more than one year. Dr. Jennex has suggested *Scenario Analysis,* a practical and wonderful idea, and offered many other specific suggestions.

REFERENCES

Alavi, M., & Leidner, D.E. (2001). Knowledge management systems: Emerging views and practices from the field. In *Proceedings of the 32nd Hawaii International Conference on Systems Sciences. IEEE Computer Society.*

Angus, J. (2006). Clementine 8.1 melds BA with BI. *InfoWorld, 26*(19), 28-29.

Bednarz, A. (2005). Microsoft beefs up SQL Server database. *Network World, 22*(13), 12.

Berger, C., & Haberstroh, B. (2005). *Oracle data mining 10g release 2: Know more, do more, spend less.* Oracle White Papers. Retrieved November 8, 2007 from http://www.oracle.com/technology/products/bi/odm/pdf/bwp_db_odm_10gr2_0905.pdf

Corral, K., Griffin, J., & Jennex, M.E. (2005). Expert's perspective: The potential of knowledge management in Data Warehousing. *Business Intelligence Journal, 10*(1), 36-40.

Davenport, T., & Harris, J. G. (2007). *Competing on analytics: The new science of winning.* Harvard Business School Press.

Deal, K. (2004). The quest for prediction. *Marketing Research, 16*(4), 45-47.

Elder IV, J.F., & Abbott, D.W. (1998, August 28). A comparison of leading data mining tools. In *Proceedings of the Fourth International Conference on Knowledge Discovery & Data* 5*Mining*, New York.

Expert Choice Inc. (2007). *Expert Choice 11*. Retrieved November 8, 2007, from http://www.expertchoice.com/software/

Fontana, J. (2005). Microsoft's future in BI market unclear. *Network World*, *22*(43), 9-14.

Giraud-Carrier, C., & Povel, O. (2003). Characterizing data mining software. *Intelligent Data Analysis*, *7*(3), 181-192.

Haughton, D., Deichmann, J., Eshghi, A., Sayek, S., Teebagy, N., & Topi, A. (2003). A review of software packages for data mining. *The American Statistician*, *57*(4), 290-309.

Hemaida, R., & Schmits, J. (2006). An analytical approach to vendor selection. *Industrial Management*, *48*(3), 18-24.

James, G., Hakim, J., Chandras, R., King, N., & Variar, G. (2004). Reviewers' choice: Only the best survive. *Intelligent Enterprise*, *7*(1), 34-38.

Jennex, M.E. (2006, April). Technologies in support of knowledge management systems. In *Proceedings of the 6th International Forum on Knowledge Management*, Tunis.

Lampe, J. C., & Garcia, A. (2004). Data mining: An in-depth look. *Internal Auditing*, *19*(2), 4-20.

Lurie, I. (2004). Product Review: Insightful Miner. *DM Review*, *14*(6), 88.

Martin, W. E. (2005). *Managing information technology* (5th ed.). Saddle River, NJ: Prentice Hall.

Mena, J. (1998). Data mining FAQ's. *DM Review*.

O Chan, J. (2000). Enterprise information system strategy and planning. *Journal of American Business, Cambridge*, *6*(2), 148-154.

Porter, M. E., & Miller, V. (2001). Strategy and the Internet. *Harvard Business Review*, *72*(3), 62-68.

Roper-Lowe, G. C., & Sharp, J. A. (1990). The analytic hierarchy process and its application to an information technology decision. *The Journal of the Operational Research Society*, *41*(1), 49-59.

Saaty, T.L. (1980). *Multicriteria decision making: The analytic hierarchy process*. RWS Publications.

Saaty, T.L. (1996). *Decision making with dependence and feedback: The analytic network process*. Pittsburgh, PA: RWS Publications.

Saaty, T.L. (2001). *The analytic network process* (2nd version). Pittsburgh, PA: RWS Publications.

Saaty, T.L. (2005). *Theory and applications of the analytic network process*. Pittsburgh. PA: RWS Publications.

Saaty, T.L., & Vargas, L.G. (2006). *Decision making with the analytic network process: Economic, political, social and technological applications with benefits, opportunities, costs and risks*. New York: Springer-Verlag.

This work was previously published in the International Journal of Knowledge Management, edited by M. Jennex, Volume 4, Issue 2, pp. 17-34, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 1.33
Intelligent User Preference Mining

Sheng-Uei Guan
Xian Jiatong-Liverpool University, China

Ping Cheng Tan
National University of Singapore, Singapore

INTRODUCTION

A business-to-consumer environment can be developed through software agents (Guan, Zhu, & Maung, 2004; Maes, 1994; Nwana & Ndumu, 1996; Wang, Guan, & Chan, 2002) to satisfy the needs of consumers patronizing online e-commerce or m-commerce stores. This includes intelligent filtering services (Chanan & Yadav, 2000) and product brokering services to understand user's needs better before alerting users of suitable products according to their preference.

We present an approach to capture individual user response towards product attributes including nonquantifiable responses. The proposed solution can capture the user's specific preference and recommend a list of products from the product database. With the proposed approach, the system can handle any unaccounted attribute that is undefined in the system. The system is able to cater to any unaccounted attribute through a general descriptions field found in most product databases. In addition, the system can adapt to changes in user's preference.

BACKGROUND

In e-commerce activities, consumers are confused by the large number of options and varieties of goods available. There is a need to provide on top of the existing filtering and search services (Bierwirth, 2000) an effective piece of software in the form of a product brokering agent to understand their needs and help them in selecting products.

Definitions

A user's choice in selecting a preferred product is often influenced by the product attributes ranging from price to brand name. This research will

classify attributes as accounted, unaccounted, and detected. The same attributes may also be classified as quantifiable or nonquantifiable. *Accounted attributes* are attributes that the system is specially customized to handle. A system is designed to capture the user's choice in terms of price and brand name, making them accounted attributes. *Unaccounted attributes* are not predefined in the system ontology. The system does not know whether an unaccounted attribute represents a product feature. Such attributes merely appear in the product descriptions field of the database. The system will attempt to identify unaccounted attributes that affect the user's preference and consider them as *detected attributes*. Thus, detected attributes are unaccounted attributes that are detected to be crucial in affecting the user's preference.

Quantifiable attributes contain specific numeric values (e.g., memory size) and their values are well defined. Nonquantifiable attributes on the other hand do not have any logical or numeric values, and their valuation could differ from user to user (e.g., brand name). The proposed system defines price and quality of a product in the ontology and considers them to be quantifiable, accounted attributes.

Related Work

One of the research goals among related work is to understand a user's needs before recommending products through the use of product brokering services. Due to the difference in complexity, different approaches were proposed to handle quantifiable and nonquantifiable attributes. One approach to handling quantifiable attributes is to compile these attributes and assign weights representing their relative importance to the user (Guan, Ngoo, & Zhu, 2002; Sheth & Maes, 1993; Zhu & Guan, 2001). The weights are adjusted to reflect the user's preference.

Much research aimed at creating an interface to understand user preference in the context of nonquantifiable attributes. This represents a more complex problem as attributes are highly subjective with no discrete metric to measure their values. Different users give different values to a particular attribute. MARI (Multi-Attribute Resource Intermediary) (MARI, 2007) proposed a "word-of-mouth" approach to solving this problem. The project split up users into general groups and estimated their preference to a specific set of attributes through the group each user belongs to. Another approach to handling nonquantifiable attributes involves requesting the user for preferred attributes. Shearin and Liberman (2001) provided a learning tool for users to explore their preferences before requesting them to suggest desirable attributes.

The problems in related work lie in the handling of nonquantifiable attributes, as the approaches are too general. Most work so far only attempted to catch user preference through generalization and stereotyping instead of understanding specific user needs. Another problem is that most works are only able to handle a specific set of attributes. The attributes to be handled are hard-coded into the system design, and the consequence is that it is not able to handle attributes that are unaccounted. However, the list of product attributes is often large. The approach used in related research may not be able to cover all the attributes, as they need to classify them into the ontology.

DESCRIPTION OF INTELLIGENT USER PREFERENCE DETECTION

The proposed approach attempts to capture user preference based on two quantifiable accounted attributes, price and quality. It learns incrementally any unaccounted attribute that affects a user's preference. If any unaccounted attribute is suspected, the system comes up with a list of candidate attributes and verifies their importance through a genetic algorithm (Haupt & Haupt, 1998). Thus, attributes that were unaccounted for

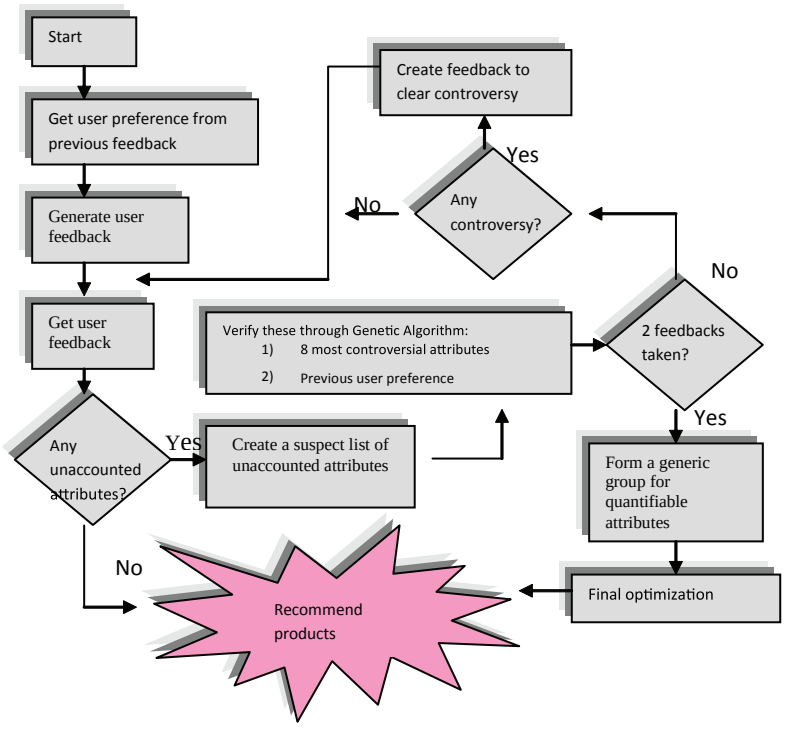
previously will be considered. The unaccounted attributes are derived from the general descriptions field of a product. The approach is therefore adaptive in nature, as the system is not restricted by the attributes it is designed to cater to.

The overall procedure is shown in Figure 1. The system first retrieves any captured information regarding the user from previous feedback and generates a list of products from the product database for the user to rank and investigates the presence of any unaccounted attribute affecting the user's preference. The system then compiles a list of possible attributes that are unaccounted for by analyzing the user feedback and rank these attributes according to their suspicion levels. The most suspicious attributes and any information captured from previous feedback are then verified using a genetic algorithm. If two cycles of feedback are completed, the system attempts to generalize any quantifiable attributes to form a generic group of attributes. The system finally optimizes the information collected by the genetic algorithm and recommends a list of products from the product database to the user according to the preference captured.

Tangible Score

In our application, we shall consider two quantifiable attributes, price and quality, as the basis in deriving the tangible score. The effect of these two attributes is always accounted for. The equation to derive this score is as shown in Equations 1-3.

Figure 1. System flowchart diagram



$$Score_{Price\ Competitive} = PrefWeight*(MaxPrice - Price)/MaxPrice \quad (1)$$

$$ScoreQuality = (1.0 - PrefWeight)*Quality \quad (2)$$

Equation 1 measures the price competitiveness of the product. PrefWeight is the weight or importance the user places on price competitiveness as compared to quality with values ranging from 0 to 1.0. A value of 1.0 indicates that 100% of the user's preference is based on price competitiveness. A product with a price close to the most expensive product will have a low score in terms of price competitiveness and vice versa.

Equation 2 measures the score given to quality. The quality attribute measures the quality of the product and takes a value ranging from 0.0 to 1.0. The value of "1.0 – PrefWeight" measures the importance of quality to the user. The final score given to tangible attributes are computed by adding Equations 1 and 2 as shown in Equation 3 below.

$$TangibleScore = Score_{PriceCompetitive} + Score_{Quality} \quad (3)$$

Modification Score for Detected Attributes

The modification score is the score assigned to all detected attributes by the system. These detected attributes were previously unaccounted for but had been detected by the system to be an attribute in the user's preference. These include all other attributes

$$Modificati\ on\ Score = \sum_{i=1}^{NoOfAttrib\ utes} (K_i - 1) * TangibleSc\ ore \quad (4)$$

besides price and quality. As these attributes may not have a quantifiable value, the score is taken as a factor of the TangibleScore derived earlier. The modification score is shown in Equation 4 whereby the modification factor K is introduced.

The values of each modification factor K ranges between 0.0 and 2.0. A value of K is assigned for each newly detected attribute. The modification factor K takes a value of 1.0 that gives a modification score of 0 when the detected attribute does not affect the user's choice. When K<1.0, there is a negative or penalty score for the particular attribute when the user has negative interest. When K >1.0, we have a bonus score to the attributes when the user has positive preference towards certain attributes. The final score for the product is as shown in Equation 5 as the sum of tangible and modification scores.

$$Final\ Score = Tangible\ Score + Modification\ Score \quad (5)$$

A Ranking System for User Feedback

As shown in Equations 1 and 2 earlier, there is a need to capture user preference in terms of the PrefWeight in Equation 1 and the various modification factor K in Equation 4. The system will request the user to rank a list of products. The system makes use of this ranked list to assess a best value for PrefWeight in Equations 1 and 2. In case when no unaccounted attributes affect the user's feedback, the agents will be evolved and optimized along the PrefWeight gradient.

Fitness of Agents

The fitness of each agent depends on the similarity between the agent's ranking of the product and the ranking made by the user. It reflects the fitness of agents in capturing the user's preference.

Unaccounted Attribute Detection

To demonstrate the system's ability to detect unaccounted attribute, the ontology will contain

only price and quality while all other attributes are unaccounted and remain to be detected, if they are relevant to the user. These unaccounted attributes include nonquantifiable attributes that are subjective in nature. The unaccounted attributes can be retrieved by analyzing the descriptions field of a product database thus allowing new attributes to be included without the need of change in system design.

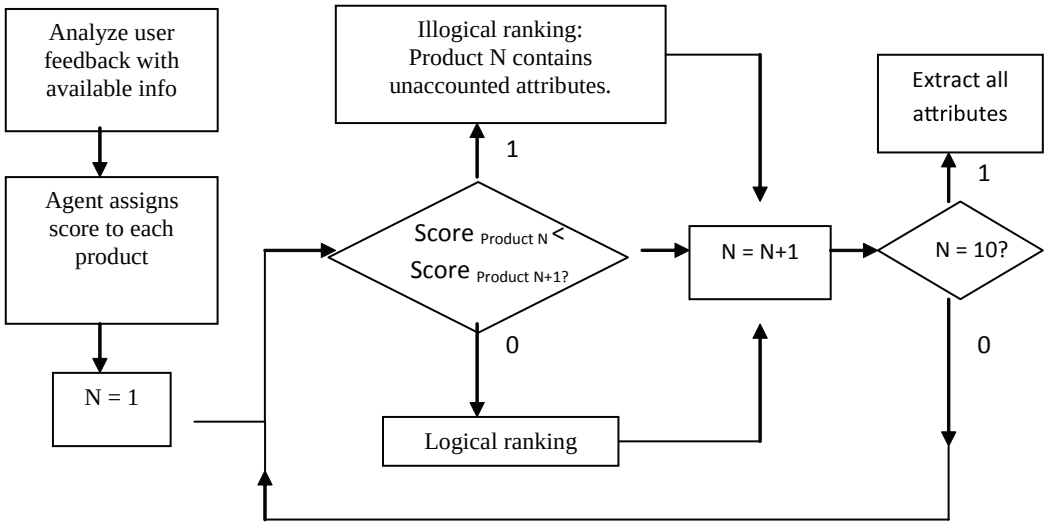
The system first goes through a detection stage where it comes up with a list of attributes that affect the user's preference. These attributes are considered as unaccounted attributes as the system has not accounted for them during this stage. A *confidence score* is assigned to each attribute according to the possibility of it being the dominant attribute influencing the user's preference.

The system will request the user to rank a list of products after that it analyzes the feedback according to the process as shown in Figure 2. The agents then attempt to explain the ranking by optimizing the PriceWeight and various K values. The fittest agent will give each product a score.

The system loops through the 10 products that are ranked by the user and compares the score given to products. If the user ranks a product higher than another, this product should have a higher score than a lower ranked product. However, if the agent awards a higher score to a product ranked lower than another, the product is deemed to contain an unaccounted attribute causing an illogical ranking. This process is able to identify all products containing positive unaccounted attributes that the user has preference for.

The next step is to identify the unaccounted attributes inside these products that cause such illogical rankings. The product descriptions for these products with illogical rankings are analyzed and tokenized. Each word in the product descriptions field is considered as a possible unaccounted attribute affecting the user's preference. Each token is considered as a possible attribute affecting the user's taste. The system will analyze the situation and modify the confidence score according to the cases as shown.

Figure 2. Chromosome encoding



1. The token appears in other products and shows no illogical ranking: deduction of points.
2. The token appears in other products and shows illogical ranking: addition of points.

Confirmation of Attributes

The attributes captured in previous feedbacks may be relevant in the current feedback as the user may choose to provide more than one set of feedback. The system thus makes a conjecture that the user's preference is influenced by certain attributes affecting him in previous feedbacks if available and eight other new attributes with the highest confidence score. The effect of these attributes on the user's preference is verified next.

Each agent in the system will estimate the user's preference by randomly assigning a modification factor ("K" value) for each of the eight attributes with high confidence score. Attributes identified to be positive are given K values greater than 1.0 while negative attributes would have K values less than 1.0. The PrefWeight and K values are optimized by a genetic algorithm to improve the fitness level of the agents.

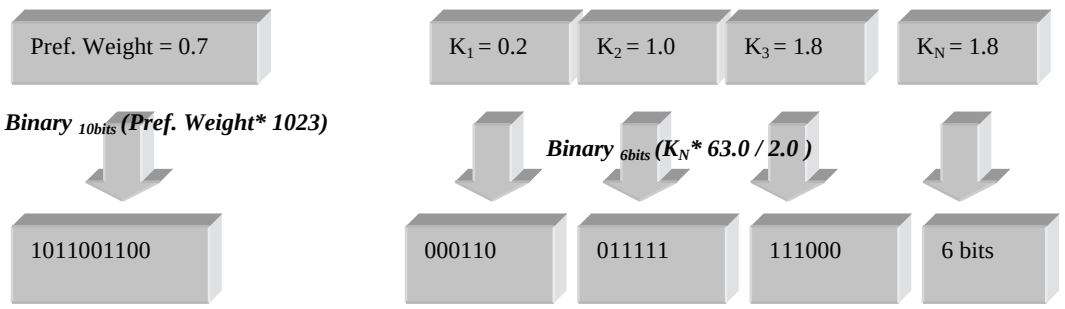
Optimization Using Genetic Algorithm

The status of detected attributes perceived by the agents and the most suspicious attributes will be verified here. The PrefWeight and various K values will be optimized to produce maximum agent fitness. As this is a multidimensional problem (Osyczka, 2001) with each new K value introducing a new dimension, we use a genetic algorithm to convert the attributes into binary strings. The agents are evolved under the genetic algorithm to optimize the fitness of each agent. The attributes of each agent are converted into a binary string as shown in Figure 3, and each bit represents a chromosome. In the design, 10 bits are used to represent the PrefWeight while 5 bits are used to represent the various K values.

Incremental Detection System

The system takes an incremental detection approach in understanding user preference, and the results show success in analyzing complex user preference. The system acknowledges that not all vital attributes may be captured within one set of feedback and thus considers refinement of

Figure 3. Process identifying products with illogical rankings



the results of previous sets. The attributes that affect a user's preference in one feedback become the prime candidates in the next set of feedback. In this way, the attributes that are detected are preserved and verified while new unaccounted attributes are being detected allowing the agents to learn incrementally about the attributes that affect the user's preference. However, some of the information captured by the system may be no longer valid as the number of feedback cycles increase. This creates a problem in the incremental detection system, as the information may not be relevant. To solve this problem, the system checks the validity of past attributes affecting the user's preference and delete attributes that are no longer relevant in the current feedback. Each set of feedback contains two feedback cycles.

Both feedback cycles attempt to detect the presence of any unaccounted attributes. In addition, the first cycle will delete any attributes that are passed from previous feedbacks and no longer relevant. These attributes should have a K value of 1.0 after we apply the genetic algorithm. Any controversial attributes detected by the first cycle will be clarified under the second feedback cycle.

IMPLEMENTATION OF INTELLIGENT USER PREFERENCE SYSTEM

A prototype was created to simulate the product broker. An independent program is written and run in the background to simulate the user. This program is used to provide feedback to the system and ranks the list of products on behalf of a simulated user who is affected by price and quality as well as a list of unaccounted attributes. The system is also affected by some generic groups of quantifiable attributes. It was observed that the performance of the system is closely related to the complexity of the problem. More complex problems tend to have lower overall performance. However, this is alleviated by providing multiple sets of feedback. The system was able to detect those attributes affecting the user's preference, and in the cases tested, the gap in performance was negligible. The system also demonstrated its ability to adapt to changes in user preference. This is important when multiple sets of feedback are involved as the user's preference may vary between feedback cycles.

FUTURE TRENDS

The current system generated user feedbacks to clarify any doubts on suspicious attributes. However, more than half of the feedbacks were generated in random to increase the chances of detecting new attributes. These random feedbacks were generated with products of different brand names having an equal chance of being selected to add to the variety of the products used for feedbacks. This could be improved by generating feedbacks to test certain popular attributes to increase the detection capabilities.

CONCLUSION

We have presented a solution in handling unaccounted attributes without the need of change in the ontology or system design. The results showed that the system is indeed capable of capturing the user's preferences even when unknown attributes were present. The system is also able to handle the presence of multiple unaccounted attributes and classify quantifiable attributes into a generic group. Given the possibility that unaccounted attributes could remain undetected during the process of mining, it could happen that a list of products recommended to the user has none matched against her preference or worse, that the user who makes a purchase based on available information and subsequently he discovers that

there were other products better meeting his needs. How to ameliorate such a situation is a challenging task, and it is on our future research agenda.

REFERENCES

Bierwirth, C. (2000). *Adaptive search and the management of logistic systems: Base models for learning agents.* Kluwer Academic Publishers.

Chanan, G., & Yadav, S. B. (2001). A conceptual model of an intelligent catalog search system. *Journal of Organizational and Electronic Commerce, 11*(1), 31-46.

Guan, S.-U., Ngoo, C. S., & Zhu, F. (2002). HandyBroker: An intelligent product-brokering agent for m-commerce applications with user preference tracking. *Electronic Commerce and Research Applications, 1*(3-4), 314-330.

Guan, S.-U., Zhu, F., & Maung, M. T. (2004). A factory-based approach to support e-commerce agent fabrication. *Electronic Commerce and Research Applications, 3*(1), 39-53.

Haupt, R. L., & Haupt, S. E. (1998). *Practical genetic algorithm.* John Wiley & Sons.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM, 37*(7), 30-40.

MARI. (2007). Retrieved March 4, 2008, from http://agents.media.mit.edu/projects_previous.html

Nwana, H.S., and Ndumu, D.T. (1996). An introduction to agent technology. BT Technology Journal, 14(4), 55-67.

Osyczka, A. (2001). Evolutionary Algorithms for Single and Multicriteria Design Optimization. Physica-Verlag.

Shearin, S., & Lieberman, H. (2001). Intelligent profiling by example. In *Proceedings of the International Conference on Intelligent User Interfaces*, Santa De, New Mexico (pp. 145-151).

Sheth, B., & Maes, P. (1993). Evolving agents for personalized information filtering. In Proceedings of the 9th Conference on Artificial Intelligence for Applications (pp. 345-352). IEEE Press.

Wang, T., Guan, S.-U., & Chan, T.K. (2002). Integrity protection for code-on-demand mobile agents in e-commerce. *Journal of Systems and Software, 60*(3), 211-221.

Zhu, F.M., & Guan, S.-U. (2001). Evolving software agents in e-commerce with gp operators and knowledge exchange. In *Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference.*

KEY TERMS

Accounted Attribute: A quality, feature, or characteristic that is listed in product specifications of a specific product.

Attribute: A quality, feature, or characteristic that some product has.

E-Commerce: Consists primarily of the distributing, buying, selling, marketing, and servicing of products or services over electronic systems such as the Internet and other computer networks.

Genetic Algorithms: Search technique used in computer science to find approximate solutions to optimization and search problems. Genetic algorithms are a particular class of evolutionary algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, natural selection, and recombination (or crossover).

M-Commerce: M-commerce, or mobile commerce, stands for electronic commerce made through mobile devices.

Ontology: Studies being or existence and their basic categories and relationships, to determine what entities and what types of entities exist.

Product Brokering: A broker is a party that mediates between a buyer and a seller.

Software Agent: An abstraction, a program that describes software that acts for a user or other program in a relationship of agency.

Tokenize: The process of converting a sequence of characters into a sequence of tokens or symbols.

This work was previously published in Encyclopedia of Information Communication Technology, edited by A. Cartelli & M. Palma, pp. 470-476, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 1.34
Mining Software Specifications

David Lo
National University of Singapore, Singapore

Siau-Cheng Khoo
National University of Singapore, Singapore

INTRODUCTION

Software is a ubiquitous component in our daily life. It ranges from large software systems like operating systems to small embedded systems like vending machines, both of which we frequently interact with. Reducing software related costs and ensuring correctness and dependability of software are certainly worthwhile goals to pursue.

Due to the short-time-to-market requirement imposed on many software projects, documented software specifications are often lacking, incomplete and outdated (Deelstra, Sinnema & Bosch 2004). Lack of documented software specifications contributes to difficulties in understanding existing systems. The latter is termed program comprehension and is estimated to contribute up to 45% of total software cost which goes to billions of dollars (Erlikh 2000, Standish 1984; Canfora & Cimitile 2002; BEA 2007). Lack of specifications also hampers automated effort of program verification and testing (Ammons, Bodik & Larus 2002).

One solution to address the above problems is mining (or automatic extraction of) software specification from program execution traces. Given a set of program traces, candidate partial specifications pertaining to the behavior a piece of software obeys can be mined.

In this chapter, we will describe recent studies on mining software specifications. Software specification mining has been one of the new directions in data mining (Lo, Khoo & Liu 2007a, Lo & Khoo 2007). Existing specification mining techniques can be categorized based on the form of specifications they mine. We will categorize and describe specification mining algorithms for mining five different target formalisms: Boolean expressions, automata (Hopcroft, Motwani & Ullman 2001), Linear Temporal Logic (Huth & Ryan 2003), frequent patterns (Han & Kamber 2006) and Live Sequence Charts (Harel & Marelly 2003).

BACKGROUND

Different from many other engineering products, software changes often during its lifespan (Lehman & Belady 1985). The process of making changes to a piece of software e.g., to fix bugs, to add features, etc., is known as software maintenance. During maintenance, there is a need to understand the current version of the software to be changed. This process is termed as program comprehension. Program comprehension is estimated to take up to 50% of software maintenance efforts which in turn is estimated to contribute up to 90% of total software costs (Erlikh 2000, Standish 1984; Canfora & Cimitile 2002). Considering the $216.0 billion of software component contribution to the US GDP at second quarter 2007, the cost associated with program comprehension potentially goes up to billions of dollars (BEA 2007). One of the root causes of this problem is the fact that documented software specification is often missing, incomplete or outdated (Deelstra, Sinnema & Bosch 2004). Mining software specifications is a promising solution to reduce software costs by reducing program comprehension efforts.

On another angle, software dependability is a well sought after goal. Ensuring software runs correctly at all times and identifying bugs are two major activities pertaining to dependability. Dependability is certainly an important issue as incorrect software has caused the loss of billions of dollars and even the loss of lives (NIST 2002; ESA & CNES 1996; GAO 1992). There are existing tools for performing program verification**.** These tools take formal specifications and automatically check them against programs to discover inconsistencies, identify bugs or ensure that all possible paths in the program satisfy the specification (Clarke, Grumberg & Peled 1999). However, programmers' reluctance and difficulty in writing formal specifications have been some of the barriers to the widespread adoption of such tools in the industry (Ammons, Bodik & Larus 2002, Holtzmann 2002). Mining software specifications can help to improve software dependability by providing these formal specifications automatically to these tools.

MAIN FOCUS

There are a number of specification mining algorithms available. These algorithms can be categorized into families based on the target specification formalisms they mine. These include specification miners that mine Boolean expressions (Ernst, Cockrell, Griswold and Notkin 2001), automata (Cook & Wolf 1998; Reiss & Reinieris, 2001; Ammons, Bodik & Larus 2002; Lo & Khoo 2006a; Lo & Khoo 2006b; Mariani, Papagiannakis and Pezzè 2007; Archaya, Xie, Pei & Xu, 2007; etc.), Linear Temporal Logic expressions (Yang, et al. 2006; Lo, Khoo & Liu 2007b; Lo, Khoo & Liu 2008, etc.), frequent patterns (Li & Zhou, 2005; El-Ramly, Stroulia & Sorenson, 2002; Lo, Khoo & Liu 2007a; etc.) and Live Sequence Charts (Lo, Maoz & Khoo 2007a, Lo, Maoz & Khoo 2007b).

These mined specifications can aid programmers in understanding existing software systems. Also, a mined specification can be converted to run-time tests (Mariani, Papagiannakis & Pezzè 2007; Lo, Maoz & Khoo 2007a; Lo, Maoz & Khoo 2007b) or input as properties-to-verify to standard program verification tools (Yang, Evans, Bhardwaj, Bhat and Das, 2006; Lo, Khoo & Liu 2007b).

Preliminaries

Before proceeding further, let us describe some preliminaries. Specifications can be mined from either traces or code. A program trace is a sequence of events. Each event in a trace can correspond to a statement being executed, or a method being called, etc. In many work, an event is simply the signature of a method that is being called.

Traces can be collected in various ways. A common method is to instrument a code by inserting `print' statement to various locations in the code. Running the instrumented code will produce a trace file which can then be analyzed.

Mining Boolean Expressions

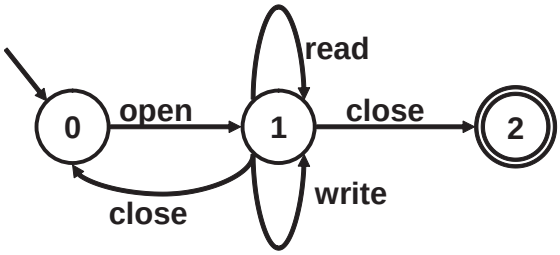
Ernst, Cockrell, Griswold and Notkin (2001) propose an algorithm that mines Boolean expressions from program execution traces at specific program points. Sample Boolean expressions mined are x=y+z, x>5, etc. The algorithm is based on a set of templates which is then matched against the program execution traces. Template instances that are satisfied by the traces above a certain threshold are outputted to the user.

Mining Automata

Simply put, an automaton is a labeled transition system with start and end nodes. Traversing an automaton from start to end nodes will produce a sentence, which will correspond to a program behavior (e.g., file protocol: open-read-write-close). An automaton represents a set of valid sentences that a program can behave. An example of an automaton representing a file protocol is drawn in Figure 1.

One of the pioneering work on mining automata is the work by Ammons, Bodik and Larus (2002). In their work, a set of pre-processed traces are input to an automata learner (Raman & Patrick, 1997). The output of the learner is a specification in the form of an automaton learned from the trace file. This automaton is then presented to end users for fine tuning and modifications.

Figure 1. File protocol specification



Lo and Khoo (2006a) define several metrics for assessing the quality of specification mining algorithms that mine automata. Among these metrics, precision and recall are introduced as measures of accuracy to existing specification miners producing automata. Precision refers to the proportion of sentences accepted by the language described by the mined automaton that are also accepted by the true specification. Recall refers to the proportion of sentences accepted by the language described by the true specification that are also accepted by the mined automaton. In the same work, a simulation framework is proposed to evaluate existing work on mining automaton-based specifications, which also help identify room for improvement.

Lo and Khoo (2006b) next propose an architecture using trace filtering and clustering to improve the quality of existing specification miners. The architecture pre-processes the traces by first filtering anomalies in the traces and then clustering them into similar groups. Each group is then fed separately to an existing specification mining algorithm to produce an automaton describing a partial specification of a system. These partial specifications are later merged into a unified automaton. It has been shown that the quality of the resultant specification mining algorithm after filtering and clustering are employed is better than before. In particular, in a case study on a Concurrent Versions System (CVS) application, the precision is doubled with a small reduction in recall.

Mariani, Papagiannakis and Pezzè (2007) use an improved automata learner (Mariani and Pezzè, 2005) and a Boolean expressions miner (Ernst, Cockrell, Griswold and Notkin, 2001) to generate regression tests of third party or Com-

mercial off-the-shelf (COTS) components. Their algorithm first learns an automaton and a set of Boolean expressions and then converts them to regression tests to ensure the compatibility of third party components when used together.

Other work on mining automata includes: (Archaya, Xie, Pei & Xu, 2007; Reiss & Reinieris, 2001; Cook & Wolf 1998; etc.)

Mining Linear Temporal Logic Expressions

Linear Temporal Logic (LTL) is a formalism for specifying precise temporal requirements. It models time as a sequence of states where each state is an event from a fixed set. The full description of LTL can be found in (Huth & Ryan, 2004). Here, we focus on 3 temporal operators of LTL namely G, X and F. The operator X refers to the next state, *F* refers to the current or a future state, and *G* captures all future states (globally). In software there are many requirements expressible in LTL, for example:

1. Whenever (globally when) *resource.lock* is called, (from the next state onwards) finally *resource.unlock* is eventually called
   Or
   G(*resource.lock*->XF (*resource.unlock*))
2. Whenever (globally when) a correct pin is entered and user requested money and the balance is sufficient, (from the next state onwards) finally an ATM eventually dispenses money
   Or
   G(*correct_pin*->XG(*request_money*->XG(sufficient->XF(dispense))))

Yang, Evans, Bhardwaj, Bhat and Das (2006) mine significant two-event temporal logic expressions stating "whenever an event E_1 occurs eventually an event E_2 occurs" from program execution traces. An expression is significant if it satisfies a minimum threshold of ``satisfaction rate''. The algorithm is limited to mining two-event temporal logic expressions due to the exponential complexity involved in mining expressions of arbitrary sizes. To capture behaviors involving more than two events, they proposed a partial solution where previously mined two-event expressions are concatenated to form longer expressions. However, not all more-than-two event expressions can be mined this way, and superfluous rules that are not significant might be introduced in this process.

Lo, Khoo and Liu (2007b, 2008) extend this work by devising an algorithm to mine significant LTL expressions of arbitrary sizes. An expression is significant if it obeys minimum thresholds of support and confidence. A novel search space pruning strategy is employed to enable efficient mining of rules of arbitrary size.

Mining Frequent Patterns

Li and Zhou (2005) use a closed itemset mining algorithm by (Grahne & Zhu 2003) in their proposed algorithm called PR-Miner to recover elements of a program (function, variable, etc) that are often used together in a code. These associations among program elements can be composed as rules that reflect implicit programming rules in code.

El-Ramly, Stroulia and Sorenson (2002) propose a new pattern mining algorithm to mine frequent user-usage scenarios of a GUI based program composed of screens – these scenarios are termed as interaction patterns. Given a set of series of screen ids, frequent interaction patterns capturing common user interactions with the GUI are obtained.

Lo, Khoo and Liu (2007a) propose a new pattern mining algorithm to mine frequent patterns of program behavior from program execution traces —these patterns are termed as iterative patterns. Patterns can occur repeatedly within a program trace and across multiple traces. These patterns follow some constraints of Live Sequence Charts (LSC), which is one of the standards in

Figure 2. A sample iterative pattern (of 32 events) mined from JBoss application server (note: diagram is read top to bottom, left to right)

Connection Set Up	TransactionManagerLocator.getInstance TransactionManagerLocator.locate TransactionManagerLocator.tryJNDI TransactionManagerLocator.usePrivateAPI	**Transaction Commit**	TxManager.commit TransactionImpl.commit TransactionImpl.beforePrepare TransactionImpl.checkIntegrity TransactionImpl.checkBeforeStatus TransactionImpl.endResources TransactionImpl.completeTransaction TransactionImpl.cancelTimeout TransactionImpl.doAfterCompletion TransactionImpl.instanceDone
TxManager Set Up	TxManager.begin XidFactory.newXid XidFactory.getNextId XidImpl.getTrulyGlobalId		
Transaction Set Up	TransactionImpl.associateCurrentThread TransactionImpl.getLocalId XidImpl.getLocalId LocalId.hashCode TransactionImpl.equals TransactionImpl.getLocalIdValue XidImpl.getLocalIdValue TransactionImpl.getLocalIdValue XidImpl.getLocalIdValue	**Transaction Disposal**	TxManager.releaseTransactionImpl TransactionImpl.getLocalId XidImpl.getLocalId LocalId.hashCode LocalId.equals

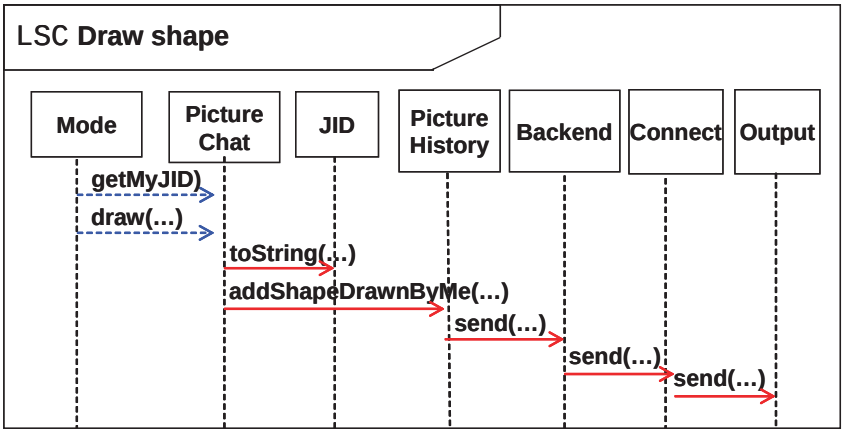
software modeling community. An example of a specification mined from the transaction component of JBoss Application Server is shown in Figure 2. It specifies that a series of connection setup events is followed by transaction manager setup events, transaction setup events, transaction commit events and eventually transaction disposal events.

Mining Live Sequence Charts

Lo, Maoz and Khoo (2007a, 2007b) extend the work on mining iterative pattern (Lo, Khoo & Liu, 2007a) and propose an algorithm to mine significant Live Sequence Charts (LSC) from program execution traces. A chart is significant if it obeys minimum thresholds of support and confidence. While iterative pattern follows some of the semantics of LSC they are not LSC. LSC is a formal version of UML sequence diagram with pre- and post- chart. An LSC specifies that when the behavior specified by the pre-chart occurs, the behavior specified by the post-chart will also occurs. Mined LSCs are also different from Linear Temporal Logic expressions mined in (Lo, Khoo & Liu, 2007b, Lo, Khoo & Liu, 2008) as the satisfactions of the pre- and post- charts conform to the semantics of LSC (Harel & Marelly 2003). Also for mining LSCs, an event in a trace is a triple (caller, callee, method signature). An example of a mined chart from Jeti instant messaging application (Jeti 2006) is shown in Figure 3. It describes a series of methods that are being called when a user of Jeti draws a line in the shared canvas with another communication party.

We have reviewed various approaches to mine different forms of software specifications. The specifications mined can aid user in understanding existing software and serve as input to other software engineering tasks, e.g., software verification, testing, etc.

Figure 3. A sample LSC mined from Jeti messaginge application (boxes, arrows, dotted arrows and solid arrows correspond to classes, method calls, pre-chart and post-chart respectively)



FUTURE TRENDS

As program traces can be huge, there is a need to improve the efficiency of existing techniques further. Further industrial case studies are also needed to adapt existing techniques to the industry. A comparison of existing techniques will do well to help users to better understand which technique works best in a particular situation. There is much room for further theoretical and practical contributions to the domain of mining software specifications.

CONCLUSION

Software is a ubiquitous component of our daily life. Documented software specifications are often missing, incomplete and outdated. This causes difficulties in understanding software systems. Program comprehension accounts for a significant proportion of total software cost. On another angle, programmers' reluctance and difficulty in writing formal specifications have been some barriers to the wide spread adoption of automatic program verification tools. Specification mining is one of the promising solutions to the above problems. It can provide candidate specifications to aid programmers in understanding existing programs. The specification can also be input to program verification tools and converted to run-time tests to aid program verification and bug detection. In the future, we look forward to more theoretical and technical contribution to this field, as well as more case studies and the availability of more open source tools.

REFERENCES

Ammons, G., Bodik, R., and Larus, J. (2002). Mining specifications. *Proceedings of the 29th Symposium on Principles of Programming Languages*, 4-16.

Archaya, M., Xie, T., J. Pei, and J. Xu (2007). Mining API patterns as partial orders from source code: from usage scenarios to specifications. *Proceedings of the 6th Joint Meeting of European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 25-34.

Canfora, G. and Cimitile, A. (2002). Software maintenance. *Handbook of Software Engineering and Knowledge Engineering (Volume 1*), 91-120, World Scientific.

Clarke, E.M., Grumberg, O. and Peled, D.A. (1999). *Model checking.* MIT Press.

Cook, J.E. and Wolf, A.L. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215-249.

Deelstra, S., Sinnema, M. and Bosch, J. (2004). Experiences in software product families: Problems and issues during product derivation. *Proceedings of the 3rd Software Product Line Conference*, 165-182.

El-Ramly,.M, Stroulia, E., and Sorenson, P. (2002) From run-time behavior to usage scenarios: An interaction-pattern mining approach. *Proceedings of International Conference on Knowledge Discovery and Data Mining*, 315-324.

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IEEE IT Pro*, 2(3), 17-23.

Ernst, M.D., Cockrell, J., Griswold, W.G., and Notkin, D. Dynamically discovering likely program invariants to support program evolution, *IEEE Transactions on Software Engineering*, 27(2), 99-123.

European Space Agency (ESA) and Centre National d'Etudes Spatiales (CNES) Independent Enquiry Board. (1996). ARIANE 5 – flight 501 failure: Report by the inquiry board. A copy at: http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html.

Grahne, G. and Zhu, J. (2003). Efficiently using prefix-trees in mining frequent itemsets. *Proceedings of the 1st Workshop on Frequent Itemset Mining Implementation*.

Han, J. and Kamber, M. (2003). *Data mining: concepts and techniques.* Morgan Kaufmann.

Harel, D. and Marelly, R. (2003). *Come, let's play: Scenario-based programming using LSCs and the play-engine.* Springer-Verlag.

Holtzmann, G.J. (2002) The logic of bugs. (2002). *Proceedings of the 10th Symposium on Foundations of Software Engineering*, 81-87.

Hopcroft, J. E., Motwani, R. and Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation.* Addison Wesley.

Huth, M. and Ryan, M. (2003). *Logic in computer science: Modeling and reasoning about systems.* Cambridge Press.

Jeti. Version 0.7.6 (Oct. 2006). http://jeti.sourceforge.net/.

Lehman, M. and Belady, L. (1985). *Program Evolution – Processes of Software Change*. Academic Press.

Lo, D., Khoo, S-C. (2006a). QUARK: Empirical assessment of automaton-based specification miners. *Proceedings of the 13th IEEE Working Conference on Reverse Engineering*, 51-60.

Lo, D., Khoo, S-C. (2006b). SMArTIC: Toward building an accurate, robust and scalable specification miner. *Proceedings of the 14th ACM SIGSOFT Symposium on Foundations of Software Engineering*, 265-275.

Lo, D., Khoo, S-C, Liu, C. (2007a). Efficient mining of iterative patterns for software specification discovery. *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, 460-469.

Lo, D., Khoo, S-C, Liu, C. (2007b). Mining temporal rules from program execution traces. *Proceedings of the 3rd International Workshop on Program Comprehension through Dynamic Analysis*, 24-28.

Lo, D., Khoo, S-C, Liu, C. (2008). Efficient mining of recurrent rules from a sequence database. *Proceedings of the 13th International Conference on Database Systems for Advance Applications.*

Lo, D., Maoz, S. and Khoo, S-C. (2007a). Mining modal scenarios from execution traces. *Companion to the 22nd Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 777-778.

Lo, D., Maoz, S. and Khoo, S-C. (2007b). Mining modal scenario based specifications from execution traces of reactive systems. *Proceedings of the 22nd International Conference on Automated Software Engineering*, 465-468.

Lo, D., Khoo, S-C. (2007). Software specification discovery: A new data mining approach. *NSF Symposium on Next Generation Data Mining.*

Mariani, L., Papagiannakis and S., Pezzè, (2007). Compatibility and regression testing of COTS-component-based software. *Proceedings of the 29th International Conference on Sofware Engineering*, 85-95.

Mariani, L. and S., Pezzè, (2005). Behavior capture and test: Automated analysis of component integration. *Proceedings of the 10th International Conference on Engineering of Complex Computer Systems*, 292-301.

Raman, A.V. and Patrick, J.D. (1997). The sk-strings method for inferring PFSA. *Proceedings of the Workshop on Automata Induction, Grammatical Inference and Language Acquisition*.

Reiss, S.P. and Renieris, M. (2001). Encoding program executions. *Proceedings of the International Conference on Software Engineering*, 221-230.

Standish, T. (1984). An essay on software reuse. IEEE *Transactions on Software Engineering*, 5(10): 494-497.

US Bureau of Economic Analysis (BEA). (Sept 2007) BEA: News release: Gross domestic product. Online at: http://www.bea.gov/newsreleases/national/gdp/gdpnewsrelease.htm.

US National Institute of Standards and Technology (NIST). (2002). Software errors cost U.S. economy $59.5 billion annually. Online at: http://www.nist.gov/public affairs/releases/n02-10.htm

US Government Accountability Office (GAO). (1992).GAO report: Patriot missile defense – software problem led to system failure at Dhahran, Saudi. A copy at: http://www.fas.org/spp/starwars/gao/im92026.htm.

Yang, J., Evans, D., Bhardwaj, D., Bhat, T. and Das, M. (2006). Perracotta: Mining temporal API rules from imperfect traces. *Proceedings of the 28th International Conference on Software Engineering*, 282-291.

KEY TERMS

Automaton: A labeled transition system with start and end nodes describing a language. A path from the start to an end node corresponds to a sentence in the language.

Linear Temporal Logic: Formalism commonly used to describe temporal requirements precisely. There are a few basic operations given with symbols G, X, F, U, W, R corresponding to English language terms 'Globally', 'neXt', 'Finally', 'Until', 'Weak-until' and 'Release'.

Live Sequence Charts: A formal version of UML sequence diagram. It is composed of a pre- and post- chart. The pre-chart describes a condition which if satisfied entails that the behavior described in the post-chart will occur.

Program Comprehension: A process of understanding a piece of software.

Program Instrumentation: Simply put, it is a process of inserting `print' statements to an existing program such that by running the instrumented program, it will produce a trace file reflecting the behavior of the program when the program is run.

Program Testing: A process find defects in a piece of software by running a set of test cases.

Program Trace: A series of events where each event can correspond to a statement that is being executed, a function that is being called etc., depending on the abstraction level considered in producing the trace.

Program Verification: A process to ensure that software is always correct no matter what input is given with respect to some properties, e.g., whenever a resource is locked for usage, it is eventually released.

Software Maintenance: A process of incorporating changes to existing software, e.g., bug fixes, feature additions, etc., while ensuring the resultant software works well.

Specification Mining or Specification Discovery: A process for automated extraction of software specification from program artifacts. We use artifacts in a liberal sense to include program traces, code, repository, data, etc.

Software Specification: A description on how a piece of software is supposed to behave.

This work was previously published in Encyclopedia of Data Warehousing and Mining, Second Edition, edited by J. Wang, pp. 1303-1309, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Section II
Development and Design Methodologies

This section provides in-depth coverage of conceptual architectures, frameworks and methodologies related to the design and implementation of software systems and applications. Throughout these contributions, research fundamentals in the discipline are presented and discussed. From broad examinations to specific discussions on particular frameworks and infrastructures, the research found within this section spans the discipline while also offering detailed, specific discussions. Basic designs, as well as abstract developments, are explained within these chapters, and frameworks for designing successful software applications.

Chapter 2.1
Ontology Based Object-Oriented Domain Modeling:
Representing Behavior

Joerg Evermann
Memorial University of Newfoundland, Canada

Yair Wand
The University of British Columbia, Canada

ABSTRACT

An important step in developing the requirements for an information system is analyzing the application domain. In this step, conceptual models are used for representing an application domain. However, while languages for software design are available and widely used, no generally accepted language exists for conceptual modeling. This work suggests the use of object-oriented software modeling languages also for conceptual modeling. Such use can support a more accurate transition from domain models to software models. As software-modeling languages were not intended for modeling application domains, their constructs lack the required semantics. While previous papers addressed the representation of structural elements of domains using object concepts, this paper addresses behavioral aspects, related to change and interaction. The proposed semantics are based on a mapping between ontological concepts that describe behavior and object-oriented constructs related to dynamics. Based on these mappings, modeling rules are proposed to guide the modeler in creating ontologically well-formed models. The mappings and rules are exemplified using UML and are demonstrated using a case study.

INTRODUCTION

A good understanding of the application domain is necessary to develop the requirements for information systems (IS). Such understanding can be facilitated with the use of conceptual models. Conceptual modeling is the "activity of formally describing some aspects of the physical and social world for the purpose of understanding" (Mylopoulos, 1992).

Despite possible benefits to IS development of using conceptual models, no widely used formal or semi-formal language for conceptual modeling exists. In contrast, formal and semi-formal languages, notably object-oriented languages, are commonly used in software design. As reported in (Dobing & Parsons, 2006, 2008) and also found in our case study (Sec. 8), practitioners, for lack of a language specific to conceptual modeling, have been using software design languages for this purpose. However, this often occurs in an unguided way, possibly leading to confusion and difficulties in understanding. Without guidance, the support of UML for describing domains other than software is poor and this can lead to miscommunication (Smolander & Rossi, 2008).

Adopting widely used and well-accepted object-oriented languages, usually employed for software design, in a guided way and with clearly specified semantics for conceptual modeling, has several potential benefits: (1) It can provide a shared language to support better communication between analysts and software designers. (2) It can help mitigate translation problems between the conceptual and the software models, (also called "impedance mismatch" (Cilia, Haupt, Mezini, & Buchmann, 2003; Kolp, Giorgini, & Mylopoulos, 2002; Roe, 2003; Rozen & Shasha, 1989). More specifically, because the domain model is specified in the same language as software, the domain model can also serve as an initial model of the software system (Coad & Yourdon, 1991), which can subsequently be adapted to particular technologies. Such technology-driven refactoring is beyond the scope of this paper. The discussion in Section 9 will revisit this point in more detail. (3) A clear representation of application aspects can reduce possible confusion of business and implementation aspects in conceptual models (Parsons & Wand, 1997). (4) Assigning semantics to language constructs for domain representation purposes can provide modeling rules (Evermann & Wand, 2005a).

Because object-oriented languages were not developed for conceptual modeling, they lack application domain semantics. For example, while language constructs such as "Method" or "Operation" have clear meaning for software design, it less clear what they represent in the application domain. However, assigning application domain semantics to language constructs, while necessary for their use in application domain modeling, is insufficient. It is also desirable to identify *modeling rules* to ensure that the created models represent only really possible situations in the application domain. Modeling rules can improve the ability to communicate and reason about the domain by restricting the possible interpretations of a model (Hadar & Soffer, 2006), and hence can support convergence of the domain understanding among different stakeholders, a pre-requisite for development and implementation success. Therefore, such rules can improve the effectiveness of the created models as ways to communicate and reason about the domain (Reinhartz-Berger & Sturm, 2008).

Previous research (Evermann & Wand, 2005b) proposed the use of object-oriented design languages for modeling the structural aspects of application domains. That research proposed specific application domain semantics for the static structure constructs found in UML class diagrams, and suggested modeling rules to develop well-formed and meaningful (with respect to perceptions of the real world application domain) models. The present work addresses the behavioral aspects of conceptual modeling, focusing on constructs to describe change and interaction. We exclude use case related constructs as they describe external interactions with a system, whereas the remaining UML constructs describe the system itself.

Our approach is based on the use of ontology, a specification of concepts that exist in a domain. Previously, ontologies have been used mostly to *evaluate* modeling languages (Green & Rosemann, 2000; Opdahl & Henderson-Sellers, 2002; Opdahl, Henderson-Sellers, & Barbier, 1999).

This work extends the ontological approach to *prescribe* modeling rules for using constructs related to dynamic to model behavioral aspects of a domain. We use the Unified Modeling Language UML (OMG, 2005) as a specific instantiation of object-oriented software design languages, due to its wide acceptance (Dobing & Parsons, 2006, 2008).

The paper is organized as follows. Section 2 presents the research methodology, followed by an introduction to ontology and the specific ontology used for this research (Sec. 3). These introductions are limited to the analysis of behavior and interactions. The main sections (Sec. 4-7) develop the proposed semantics and modeling rules. The paper closes with a presentation of case study research that provides empirical support for the proposed semantics and rules (Sec. 8).

RESEARCH METHODOLOGY

To assign application domain semantics to object-oriented language constructs, we create mappings between language constructs and domain concepts. Domain concepts are specified in terms of an ontology. Ontology is an area of study that is "that branch of philosophy which deals with the order and structure of reality in the broadest sense possible" (Angeles, 1981). A particular ontology specifies what exists and how things behave in a domain. Ontological mappings are established by the following two steps (Wand & Weber, 1993):

1. Representation mapping: Assign each ontological concept a language construct with which to *represent* it.
2. Interpretation mapping: Assign each language construct an ontological *interpretation*.

Once ontological mappings have been established, the transfer of ontological assumptions and constraints to the language generates modeling rules. This transfer is based on the principle that relationships between ontological concepts should also hold between the language constructs that are used to represent them.

Our analysis is guided by the principle that the mappings should retain the existing relationships between language constructs, so as not to affect its software design semantics. For example, as associations can relate UML classes, there should be something that relates the ontological concepts to which UML classes are mapped. If this condition cannot be satisfied, then the assignment of semantics to object constructs cannot be accomplished without violating some fundamental design language aspects. Given this requirement, the mapping of one language construct has implications for the possible mapping of other, related ones. We address these interdependencies by using an iterative method, changing our perspective repeatedly between the representation mapping and the interpretation mapping. We begin by proposing an initial representation mapping for core ontological concepts (Section 4). Based on this mapping, we propose an interpretation mapping for related, as yet unmapped, language constructs (Section 5). We repeat this for a second time to cover all remaining ontological concepts and language constructs (Sections 6, 7). In this way, we ensure a mapping that is internally consistent and respects the existing dependencies between language constructs as much as possible.

This iterative methodology also addresses the problem of "correct" interpretation of the ontology and the language specifications. As both specifications are texts that must be read and interpreted, the iterative methodology follows the hermeneutic cycle (Gadamer, 1976; Ricoeur, 1976), recognized as essential in interpretive IS research (Boland, 1985; Chalmers, 2004; Myers, 1995; Prasad, 2002). This prevents premature assignment of meaning and allows the entire meaning of the language and ontology specifications to emerge.

Our work has the following limitations. First, it is not our purpose to suggest modeling rules for software design. The derived rules are appropriate for conceptual modeling but may not be so for software design. However, models that conform to our rules are valid software models, as we do not violate the existing software semantics of the language. Second, our rules do not guide us in how to perceive the world. Thus, we might suggest rules on how to use modeling constructs such as objects and classes, but not on how to identify them in the domain. For example, we do not offer rules that tell the modeler how to identify things in the domain. Instead, our rules are of the kind that, given the modeler has already identified a thing, the rules tell her how to model it in object-oriented terms. Third, it is beyond the scope of a single paper to examine all UML constructs, as several hundreds UML constructs exist. We restrict ourselves to the basic constructs commonly seen in UML diagrams. Furthermore, some language constructs serve a purely syntactic purpose or only have meanings in the software context and cannot be assigned any domain semantics (e.g. Pseudo State, Stub State). We consider these constructs not relevant for conceptual modelling. We also exclude use case related constructs, as they describe external interactions with a system, whereas the remaining UML constructs describe the system itself.

ONTOLOGY

The term *ontology* has seen increasing use in such areas as information systems, artificial intelligence, and knowledge engineering (Gruninger & Lee, 2002; Noy & Hafner, 1997; Uschold & Gruninger, 1996). The present research takes up the call for a return to philosophical ontology (Guarino & Welty, 2002; Smith & Welty, 2001). An ontology is taken to be a commitment to the belief in the existence of certain entities in external reality, for example the business and organizational world. A specific ontology is a set of assumptions about what exists or is perceived to exist in a domain. Adopting an ontology is a fundamental philosophical choice that cannot be justified *a-priori*. As any philosophy, it is the framework that enables one to carry out research (Kuhn, 1996), and the merits of its adoption can only be assessed based on the results of that research. Some reasons for our choice of ontology, based on previous research results, are given below.

The Bunge-Ontology

The specific ontology chosen for our purposes is based on the work of Mario Bunge (1977, 1979). We use this particular ontology for pragmatic reasons:

- It is based on ontological work done over a long period (Bunge, 1977, pg. xiii).
- It is an axiomatic system and formally represented in set theory notation.
- It has not been developed specifically for use in software development, but is instead based on "the ontological presuppositions of contemporary scientific research" (Bunge, 1977, pg. xiii). Thus, its use can help distinguish between domain and software concepts.
- It has been shown to provide a good benchmark for the evaluation of modeling languages and methods (Dussart, Aubert, & Patry, 2004; Evermann & Wand, 2005b; Opdahl & Henderson-Sellers, 2001; Parsons & Wand, 1997; Rosemann & Green, 2000; Soffer, Golany, Dori, & Wand, 2001; Wand, Storey, & Weber, 1999; Wand & Weber, 1989, 1993; Weber & Zhang, 1996).
- It has been used to assign ontological meaning to object concepts (Wand, 1989).
- It has been empirically shown to lead to useful outcomes (Bodart, Patel, Sim, & Weber, 2001; Cockroft & Rowles, 2003; Evermann & Wand, 2006; Gemino, 1999; Weber & Zhang, 1996).

The following paragraphs introduce the ontological concepts of Bunge's work (Bunge, 1977, 1979). This is necessarily an incomplete exposition of the entire ontology, including only the concepts relevant to this paper. A brief synopsis is presented in Table 1.

The world is made up of substantial *things* that exist physically in the world. Things can combine to form a *composite* thing. Things possess (substantial) *properties.* Properties in general are those possessed by a set of things, for example "color," "speed,, "salary," and so on. Individual properties, that is, properties of an individual thing, can be considered values of properties in general, (e.g., a specific thing is blue in color, and is traveling at a speed of 100 mph; a specific thing is a person earning a salary of $50,000). Intrinsic properties are those that a thing possesses by itself, whereas mutual properties exist between two or more things. Composites possess *emergent* properties, that is, properties not possessed by any component. A *law* is any restriction on the properties of a thing.

Any thing can be described by a set of *attributes,* which (Bunge, 1977) also calls *state functions.* A set of state functions is called a *functional schema* or *model.* A thing can be described by many different schemata. For example, a person may be described by height and weight, or described by location and organizational unit. The *state* of a thing is defined as the set of values of *all* state functions comprising a particular model. The *lawful state space* of a thing is defined by constraining the values of the state functions to those values consistent with the laws the thing adheres to.

Table 1. Concepts of the Bunge-ontology

Ontological Concept	**Explanation**
Thing	Fundamental concept. The world consists of things and only of things.
Property	Things have properties.
Intrinsic Property	Property of one thing.
Mutual Property	Property of two or more things.
Law	Restriction of or relation on properties.
Composition	Things can be composed to form composite things.
Emergent Property	Property of a composite thing that is not a property of one of its components.
State Function	Function describing a property of a thing. Synonymous with attribute.
Functional Schema (Model)	Set of state functions describing things. A model of things that are similar in some ways.
State	Value vector of state functions of a functional schema.
Natural Kind	Set of things adhering to same laws (common behavior).
Quantitative Change	Change of state of a thing.
Qualitative Change	Change of natural kind of a thing by loss or acquisition of properties.
Event	Change of state.
Process	Ordered set of events of one thing.
Lawful Transformation	Path in state space between an initial and a final state, where all points on the path are lawful states.
History	The states a thing traverses over time.
Action	A thing A acts on a thing B iff the state history of B depends on the existence of A.
Interaction	Two or more things acting on each other.

All things change, and every change is a change of things. A change may be *quantitative*, in which case the values of one or more state functions (and individual properties) are changed, or it may be *qualitative*, in which case state functions (and general properties) are acquired or lost. Concurrent with loss or acquisition of properties is the loss or acquisition of behavior as things lose or gain possible ways in which they can change. Change always involves the change of state of some thing. Since all things are changeable, every lawful state space contains at least two distinct states.

A discrete change is termed an *event*. An event can be described as an ordered pair of states. An event is defined for the state space of a *single* thing. A change in one thing may also be a change in another (e.g., changes of mutual properties). This is an interaction consisting of two distinct events in the two things. If the state space is non-denumerable, that is for continuous change, a change is represented by a triple (s_i, s_f, g) representing the initial state, the final state and a function g, the *lawful transformation*, which represents the path in the state space that the thing traverses. All possible changes must be in accordance with laws.

Interaction is defined through the state history of a thing: If the state history of one thing depends on the existence of another thing then the second is said to *act on* the first. Things interact, if each acts upon the other. Every thing acts on, and is acted on, by other things. Changes occur as a consequence of laws: If two properties are lawfully related and one changes, then the other may also change, depending on the laws that relate them. Since laws relate properties of one thing only, for a thing A to act on a thing B as the result of laws, there must exist a mutual property of A and B which is lawfully related to (intrinsic) properties of A and B. Furthermore, interaction may give rise to mutual properties. The interaction of a person enrolling at a university to become a student would give rise to the mutual property "tuition fee balance." Hence, some properties necessarily exist prior to interaction and some others may exist post interaction (Evermann, 2005).

REPRESENTATION MAPPING

We base our mapping process on the mapping of some fundamental concepts in existing research (Evermann & Wand, 2005b). These fundamental mappings are in agreement also with other work on UML and Bunge's ontology (Dussart et al., 2004; Evermann, 2005; Opdahl & Henderson-Sellers, 2001, 2002). Table 2 shows these mappings. With these fundamental mapping of Table 2 in mind, we begin our mapping of concepts related to change and behavior.

States and State Transitions

The central concepts of behavior in Bunge's ontology are states and state transitions. Object-oriented languages such as UML also provide states and state transition. We propose to represent ontological states by object-states and ontological state transitions by object-state transitions. Some consequences follow from this mapping.

In Bunge's ontology, a state is the *complete* assignment of values to the state functions, that is a vector of the values of *all* state functions. While the object-oriented literature has noted the connection between attributes and states (Booch, 1994; Coad & Yourdon, 1990; Jacobson, 1992; Rumbaugh, 1991), UML does not support modeling this connection. We therefore propose the following rule to preserve the ontological relationship between states and properties also in object-oriented languages:

Rule 1: *Every object-state must be defined by a specific assignment of values to a set of attributes of the object for which the state is defined.*

This rule ensures that appropriate attributes are modeled to be able to fully represent the dynamics

Table 2. Prior mappings of fundamental concepts

Ontological Concept	Object Construct in UML	Remarks
Thing	Object	
Property	Attribute	
Intrinsic Property	Attribute of 'ordinary' class	
Mutual Property	Attribute of association class	
Emergent Property	Class attribute	Attribute of an aggregate made of class instances.
Functional Schema	Class	
Natural Kind	Set of objects (extension of class)	Described by class.
Composition	Aggregation	
	Composition	Does not have an ontological equivalence. In ontology things exist independent of whether they are components or not.
	Association	Does not have an ontological equivalence. Implies interaction (Evermann, 2005).
A collection of mutual properties that are generated together	Association class	

allowed by the state space and the transitions within this space. For example, if a student in a university domain has two states, "passed" or "failed," this must be modeled using some attributes (for example "Course Credits" with different number of course credits defining each state, "passed" or "failed").

Ontologically, every change of state changes the value of at least one state function. For example, when a student passes a course (transitions from "failed" to "passed" standing), the changed value would be that of the attribute "Course Credits". Hence, from Rule 1 follows:

Corollary 1: *An object's state transition must change the value of at least one of its attributes.*

Interpretation Mapping (1st Iteration)

Sub-States

Object-oriented languages like UML allow the definition of a state hierarchy in which a state may contain sub-states. An object in a sub-state is also in the super-state containing that sub-state. In Bunge's ontology, states do not have sub-states. Hence, we must find an ontological interpretation for this object-oriented "shorthand" notation.

Ontologically, sub-states can be interpreted using two different functional schemata of the represented thing, spanning two different state spaces. For example, let a telephone answering machine have a state "Passive" and a composite state "Active" with sub-states "Greeting Recording," "Message Recording," and "Message Playback." States "Active" and "Passive" can be characterized by an attribute "Tape Speed" (zero for "Passive" and greater than zero "Active"). This is one model of the answering machine. Distinguishing among the sub-states of "Active" requires additional state functions, for example "Recording Source" (with values "Microphone" or "Telephone Line"), and "Mode" (with values "Recording" or "Playback"). Thus, the second model is defined by three attributes. In this model it is possible to distinguish between states such as "Message Recording" ("Tape Speed > zero," "Recording Source=Telephone Line," and "Mode=Recording"), and "Greeting

Recording" ("Tape Speed > zero," "Recording Source=Microphone," and "Mode-recording").

Formally, let state x (defined by a set of state variables $a=a_1$) be a point in state space sp_x, spanned by attribute a. Let sub-state y be a point in state space sp_y, spanned by attribute set a and a set of additional attributes b. Multiple points in sp_y, for example $<a=a_1, b=b_1>$, $<a=a_1, b=b_2>$, and so on can be mapped to a single point x in sp_x. State space sp_x is a *projection* of sp_y onto attribute set a. Based on the above example and this formal discussion we propose the following rule and corollary:

Rule 2: *Object-oriented sub-states are defined using attributes in addition to those defining the super-state.*

Corollary 2: *For an object transitioning among sub-states, the values of only those attributes change, that are not used to define the super-state.*

In object-oriented languages, two or more sub-states of the same super-state may be concurrent. In addition to the conditions on the super- and sub-states expressed by Rule 2 and Corollary 2 we require that, for two sub-states of a common super-state to be concurrent, the two higher-dimensional state spaces are independent:

Corollary 3: *Concurrent sub-states require mutually disjoint sets of additional attributes in the class description.*

The rules proposed in this section help to ensure that the attribute definitions for object classes are sufficient to support the desired behavioral characteristics of the object as expressed in state charts. Specifically, in UML, the proposed language rules relate elements of state charts with those of class diagrams.

Active States

Some object-oriented languages allow modeling of object states during which the object undergoes a change. In UML 1.x, these were called action states. In UML 2.0, actions and activities are separated from states. However, the notion of a state during which changes occur remains, as behavior can be associated with states in the form of a "DoActivity." This is incompatible with the ontological notion that a state is an assignment of values to attributes at a specific instance in time. Again, we must identify the proper ontological interpretation of this object-oriented "shorthand notation".

We build on our interpretation of object-oriented sub-states: If a change occurs while a thing is in a certain state, there must exist a state variable not employed for the state definition and that state variable may change. Hence, action states can be represented by means of sub-states:

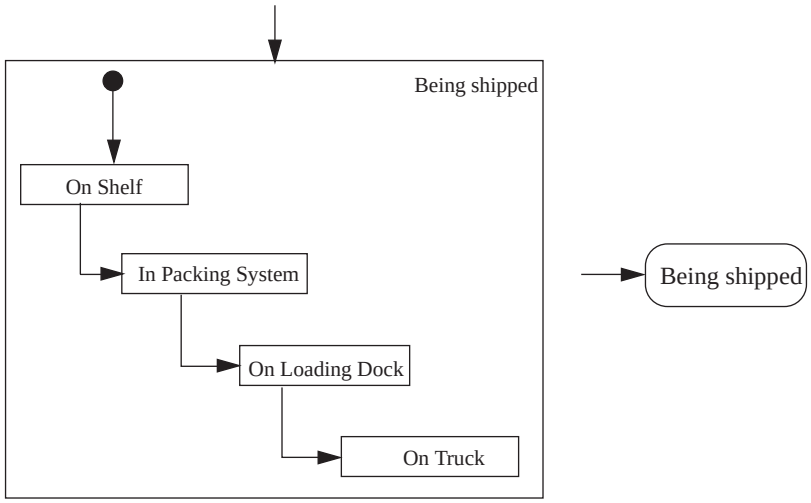
Rule 3: *Action states are composite states comprising a set of sub-states. The object transitions among the sub-states while in the action state. State charts and class diagrams must reflect this.*

For example, the situation in Figure 1A is equivalent to that in Figure 1B. An item that is in the action state of "being shipped" undergoes state transitions while being in that state, for example from state "On Shelf" to a new state "In Packing System" and so on.

Operations

Bunge's ontology provides no construct that is equivalent to methods or operations in object-oriented languages. It has been suggested that these are software related constructs that are without relevance for real-world models (Parsons & Wand, 1991, 1997) and thus need not be interpreted ontologically. However, as operations feature prominently in object-oriented modelling,

Figure 1. Action-states



this section shows how operations are related to state transitions, thereby gaining an ontological interpretation, and derives rules for their use in conceptual modeling.

Whereas operations specify behavior only in abstract terms, a method provides a specific implementation of an operation[2]. As Bunge's ontology expresses dynamics through states and transitions, we must use these concepts to assign domain semantics to methods and operations.

We note that in the object approach, the *entire* range of behavior of an object (except creation and reclassification, i.e. qualitative change) is determined by its set of operations. Ontologically, *every* change of a thing is describable in terms of state transitions. We therefore propose:

Rule 4: *The quantitative behavior of objects of a particular class (for each model) is entirely describable by a top-level state chart SC_0 associated with the class.*

Since the behavior of objects is limited to the operations defined on the object:

Rule 5: *Every object-oriented state-transition in SC_0 must correspond to an operation modeled for the class of objects that SC_0 is associated with and vice versa.*

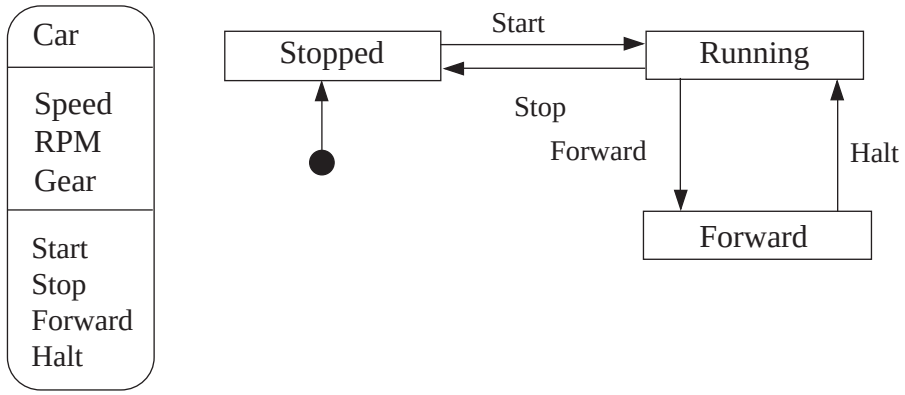
Since in our ontology every thing is able to change, we propose the following:

Corollary 4: *Every object must have at least one operation.*

Bunge's ontology, as adapted in (Wand & Weber, 1993, 1995), distinguishes among *stable* and *unstable* states. A thing can only leave a stable state due to interaction. In object-oriented descriptions, an object remains in a particular state until an operation is invoked on it. An operation reflects the complete response to this invocation; the object then awaits another operation to be invoked. Since we have proposed that operations are realizations of top-level state transitions, we add now the rule that top-level state transitions are transitions from and to stable states:

Corollary 5: *States in SC_0 represent stable states.*

Figure 2. Class definition and state chart



(A) (B)

The above rules and corollaries allow the modeler to identify operations that may have been missed in constructing the class diagram, or identify operations for which there are no corresponding state transitions. Such operations would be redundant. Figure 2 shows an example. Consider a car that can be in three states, engine "Stopped," engine "Running," and going "Forward." Assume the states of a car are defined as in Table 3. Then the class definition in Figure 2 (A) is consistent with the top-level state chart SC_0 depicted in Figure 2 (B).

Explicitly modeling the link between operations and state transitions provides the operation and method designer a way of ensuring that the full state description of a system is realized. It also helps ensure that operations do not implement change beyond that in the state description.

Behavior

In object-oriented languages such as UML, state charts may be used to specify the behavior that implements an operation. These state charts must follow some rules. In general, behavior represents lawful transformations that describe in greater detail the state transition represented by an operation. Hence, the beginning and end of the path in state space represented by a particular state chart must match the initial and final states of the operation whose behavior this state chart describes:

Table 3. Example state definitions

Stopped	(Speed = 0, RPM = 0)
Running	(Speed = 0, RPM = 1000, Gear = 0)
Forward	(Speed = 50, RPM = 3000, Gear = 1)

Corollary 6: *A state chart describing an operation must begin and end with the initial and final state of the transition in SC_0 that corresponds to the operation.*

Corollary 7: *The behavior represented by a state chart must modify the attribute values of the object in a way that corresponds to the values defined for the initial and the final state of the state chart.*

These rules and corollaries allow the modeler to identify incomplete state charts. If a state chart describing a particular behavior does not begin and end with the proper states, it may be incomplete. Corollary 7 can help design behavior specifications that conform to the state description by providing pre- and post-conditions of a method. For example, let the answering machine switch from state "Message Recording" to state "Passive."

The initial state is characterized by attributes and values "Tape Speed > 0," "Recording Source = Telephone Line," and "Mode = Recording" while the final state is defined by "Tape Speed = 0." A method implementing an operation corresponding to this state transition should assume the object representing the answering machine is in a state possessing attributes according to the initial state and must ensure the assignment of object attributes according to the final state.

REPRESENTATION MAPPING

In Bunge's ontology, interaction is represented by state transitions, rather than message-passing, as in object-oriented modeling. Ontologically, interaction arises because things adhere to laws that must be satisfied at all times: When an event in thing A changes a property that is lawfully related to a property of thing B (this relation must be via mutual properties), the property of thing B may need to adjust so that the law remains satisfied. Hence, interactions can, in theory, be derived from the knowledge of the laws, for example by means of constraint solving (Mackworth, 1977) or constraint programming (Van Hentenryck, 1989). We say "in theory" because these methods are not generally applicable and, except for special cases, not computationally efficient.

It is the object-oriented system designer's task to ensure that the modeled message-passing pattern satisfies all application domain laws (often termed business rules). It is often the case that such laws are only implicitly understood by analysts, rather than being explicitly stated and modeled. A rigorous ontological analysis of the domain can help explicate the existence of laws or constraints and suggest possible ways how laws (business rules) can be implemented via interactions (message passing).

The discussion of Bunge's ontology in Section 3 suggests that since laws constrain the state variables of only a single thing, changes between two things must happen by virtue of mutual properties. Thus, a change of a mutual property in one thing does not lead to a change in another thing but *is* a change in a property of another thing. Since mutual properties are mapped to association class attributes (see Table 2), we derive the following rule:

Rule 6: *For every two classes of objects between which message passing is declared, there exists an association class with at least one attribute.*

This rule can help the modeler to identify which messages need to be shown in a model and between which two objects an association class needs to be identified. This in turn can provide for a richer analysis of the application domain. The case study in Section 8 demonstrates that this is feasible as the number of relevant messages, and therefore the number of association classes, typically remains manageable.

The scope of the model and level of abstraction determines whether a particular thing and its mutual properties should be modeled. Clearly, one does not wish to model individual atoms and molecules even though they are ultimately the connected things. Scope and level of abstraction are determined based on the model purpose prior to modeling. For example, intermediate communication mechanisms such as communication systems are typically outside the scope of a model of a business (but not a technical) domain: A telephone line between two people possesses mutual properties with both the speaker and the listener. It is these mutual properties and the properties of the phone line (voltage, waveform, etc.) that undergo change. However, as the telephone line is typically outside of the model scope, it and its properties will not be included in the model. Instead, the interaction between the two individuals will be considered using properties shared directly between the individuals, such as shared beliefs, goals, and so on that may be affected by the interaction.

In Bunge's ontology every thing acts on and is acted upon by other things. Hence, there should be interactions originating from and terminating on the thing:

Rule 7: *Every object must be the receiver and sender of, or responder to, some message.*

This rule shows the need to critically examine the interactions or information flows. One could argue that things in the environment of a system may only send but not receive messages, for example a customer ordering a product, a student registering for a course, and so on. Nevertheless, even for these examples, the actions of ordering and registering are pointless unless the customer and the student also receive messages. This could be order acknowledgements or shipping details for the customer. For the student, the messages may involve tuition fee billing or textbook requirement messages. Neither customers nor students will only send messages without expecting to receive any. This rule also illustrates differences between software and domain models. Consider a `LineItem` object in a transaction application that might in a software model be shown as only receiving messages. However, as we wish to describe the domain, rather than the software, we require an interpretation of what a `LineItem` object represents. For example, it might represent a particular product being ordered by a particular customer. In that case, it is not a thing, but an event or activity, and should not be modeled as an object.

Often, a thing is made up of parts. In such cases, it is sufficient for any of its parts to receive and send some message to satisfy rule 7. The receiving and sending must not necessarily be done by the same part. For example, assume an object A with parts P_1 and P_2. If P_1 receives messages (is acted upon) but does not send any messages (acts on), and P_2 sends messages but does not receive any, we consider rule 7 to be satisfied, as the object A both receives and sends messages. In this case, some interaction necessarily exists between the parts P_1 and P_2 (which is not shown on the level of the composite object).

A specific instance of such a composite object is the domain environment. The necessarily limited scope of any model leads to some things interacting with things beyond the model scope, that is, with the environment. Rather than showing the individual objects comprising the environment (which are outside the scope), the modeler may instead choose to designate one thing as the environment, with the understanding that it is a composite object and subject to message passing amongst its parts.

In summary, while we suggest that message passing is not ontologically real, we advocate its use in conceptual modeling to represent interaction, with the understanding that the ontologically real interaction occurs by mutual properties and therefore requires association classes.

Interpretation Mapping (2nd Iteration)

Message passing is one of the core concepts of the object-oriented approach. It is the mechanism by which object interaction is realized. Messages can be interpreted in two ways, depending on the ontological status we ascribe to them:

- Messages are not things in the world. They are abstract concepts that serve as descriptions, illustrations, abstractions or representations of interaction (Wand, 1989).
- Messages are substantive things in the world. They are ontologically real.

There are two arguments against the second interpretation (Evermann, 2005). First, the message passing mechanism for interaction has previously been examined with respect to Bunge's ontology and found to be an unsuitable construct or mechanism for describing real world business domains (Parsons & Wand, 1991, 1997; Wand &

Weber, 1993). Consider the following examples:

– The machine sends a message to the machined item to move itself to a new location.
– The general ledger sends a message to an office desk to depreciate its value.
– A truck sends a message to the crate to load itself onto the loading dock.

While acceptable in software specifications, such messages are not likely to be observed between these things in the real world. Clearly, a machine does not send messages to machined items to move them. Instead, an operator moves items to and from the machine.

Second, if messages did represent real things in the world, they should be represented as objects (see Table 2). Instead of two objects interacting directly, one object would have to interact with a message object, which in turn would need to interact with the second object. This would lead to an infinite regress. Based on these two reasons, we interpret messages as abstract expressions of interaction, not as things. Consequently, they are mapped to the ontological concept of interaction, not to things.

In UML, messages are associated with value specifications (their arguments). Specifying message arguments implies that interaction occurs through changes to the value of properties represented by the message parameters. However, message arguments do not refer to attributes of either the sending or receiving object, and are therefore not interpreted as (mutual) properties.

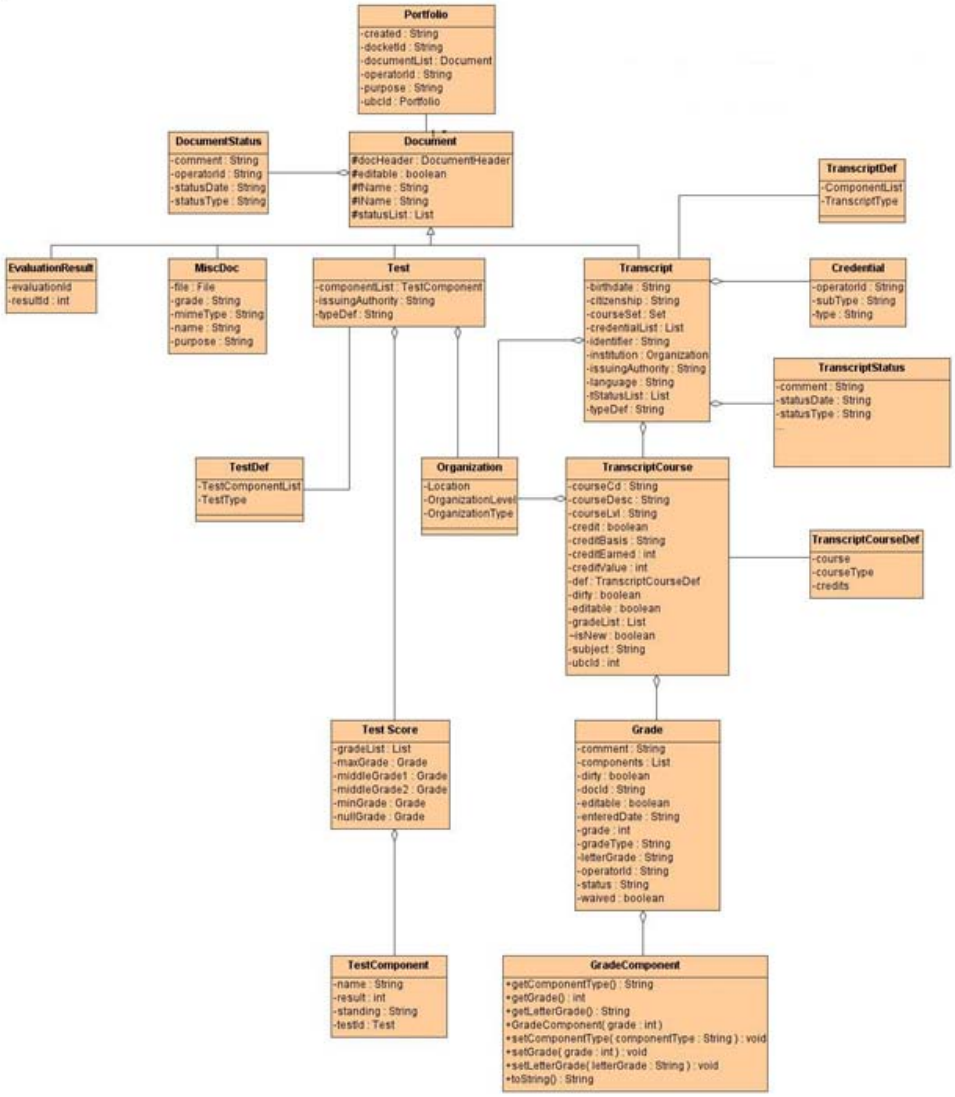
To summarize, messages and their arguments do not have a direct ontological interpretation, but express interaction. In turn, Sect. 6 proposed that interpretation should be expressed using association class attributes.

CASE STUDY

A case study was conducted to examine the feasibility of the proposed modeling rules in practical situations. The organization under study was a large North American university. The project under study was the provision of an opportunity for prospective students to assess their likelihood of admittance[3]. This project was chosen as the basis for the case study for two reasons. (1) The main stakeholders and informants of the analysis were still available for interviews. Their involvement with the project and the analysis was recent so that rich data could still be gathered. (2) The project team had used UML for an initial model of the organizational domain. UML class diagrams were used specifically for understanding the organizational domain. The project leader confirmed there was no intention to use them for later software design and all coding would be done independent of any generated UML model. It was hoped that additional insight could be gained by comparing the project team's UML model with that developed as part of this case study. When this case study was begun, the project was almost at the beginning of the implementation stage. Figure 3 shows the class diagram developed by the project team.

As part of the case study, an independent business analysis was undertaken by the first author in order to model the domain of student admissions and assessment. This served two purposes: (1) it tested the applicability of the proposed modeling rules and (2) produced an alternative model with ontological semantics, which could be compared to the original one created by the project team. Interviews were conducted with stakeholders and informants who contributed to the project. All interviews were conducted on-site using open-ended questions. The following interviews were conducted for purposes of domain analysis and model creation:

Figure 3. UML class diagram developed by project team



- Approx 1-hour interview with director, student systems, IT services
- Approx 1.5-hour interview with a high school student recruiter
- Approx 2-hour interview with an admissions officer
- Three approx 1-hour interviews with three first year university students
- Three approx 1 hour interviews with lead analyst of the project

In addition, access was provided to all project documentation, mainly requirements documents generated from a prior business process reengineering project. Moreover, frequent e-mail contact was maintained with the lead analyst of the project for clarification and discussion of details. For purposes of evaluating and comparing the independently generated model, more interviews were conducted:

- Approx. 2-hour interview with the project leader
- Approx. 2-hour interview with the lead developer

For preparation for these interviews, the entire set of independently created diagrams was provided two weeks prior to the interview with a request to carefully examine these for correctness, usefulness, usability and to compare these to the project's own UML model. The interviewees were made aware of these aims before the interviews commenced.

The process of analyzing the domain and modeling it according to the proposed ontological semantics and rules showed that all produced diagrams were valid UML models, and adhered to all UML syntactic rules. The models were neither trivial nor overly complex. Hence, use of the rules was found to be at least feasible in this practical setting. The independently generated model consisted of the following diagrams:

- 20 class diagrams showing 35 classes and 14 association classes,
- Eight sequence diagrams showing 37 messages, representing interactions that occur through changes to the attributes of the 14 association classes,
- Two state charts showing seven states and sub-states.

An excerpt of the class model is shown in Figure 5. In the model, application is not a thing (it is not represented as a class, but as an association class). Its attributes reflect mutual properties, the result of an interaction between the university and the applicant. Either the applicant or the university can modify these mutual properties, which would be further interaction between university and the applicant. Similarly, acknowledgement is not a thing (it is not represented as a class, but as an association class). Its attributes reflect mutual properties, the results of the application event/activity and can be changed by either the university or the applicant. Such changes are interactions. Note further that while the domain describes student admissions, there is no admission object. Instead, in the state chart in Figure 4, "accepted" is a state that the university can have with respect to a student. One may model acceptance as a state transition to this state, that is, a change in the university object.

The process of business analysis and modeling showed that at various stages in the process the rules and proposed semantics led the modeler to include information in the model that was not necessary for building software, but that was important for domain understanding.

With respect to action and interaction, the general absence of detailed state charts in the independent business re-analysis indicated mostly *qualitative* interactions (e.g. a student enrolling, a student being admitted, etc.). This might not be surprising, as non-technical, human organizational systems appear to generally involve more qualitative than quantitative change.

When quantitative change was modeled (e.g. as a transition between the states "Not Admissible," "Conditionally Admitted," and "Unconditionally Admitted" based on revisions to the required grades), application of the rules led to a fundamental insight into the domain. Instead of assigning states such as "on hold," "pending," "admitted," and so on to the student, the rules led us to assign such states to the university (Figure 4). The student's state cannot change without interaction, and changes to admission criteria by the university do not affect the student *per se*, but only the student's admissibility with respect to the university. Hence, it was concluded that these states were states of the university, defined using mutual properties of the university and the student. Consequently, the university possesses an independent sub-state machine for each student. This observation confirms the notion that in the real world it is the university that can change the state of the students (i.e., those aspects of state

Figure 4. Example state chart diagram from the case study

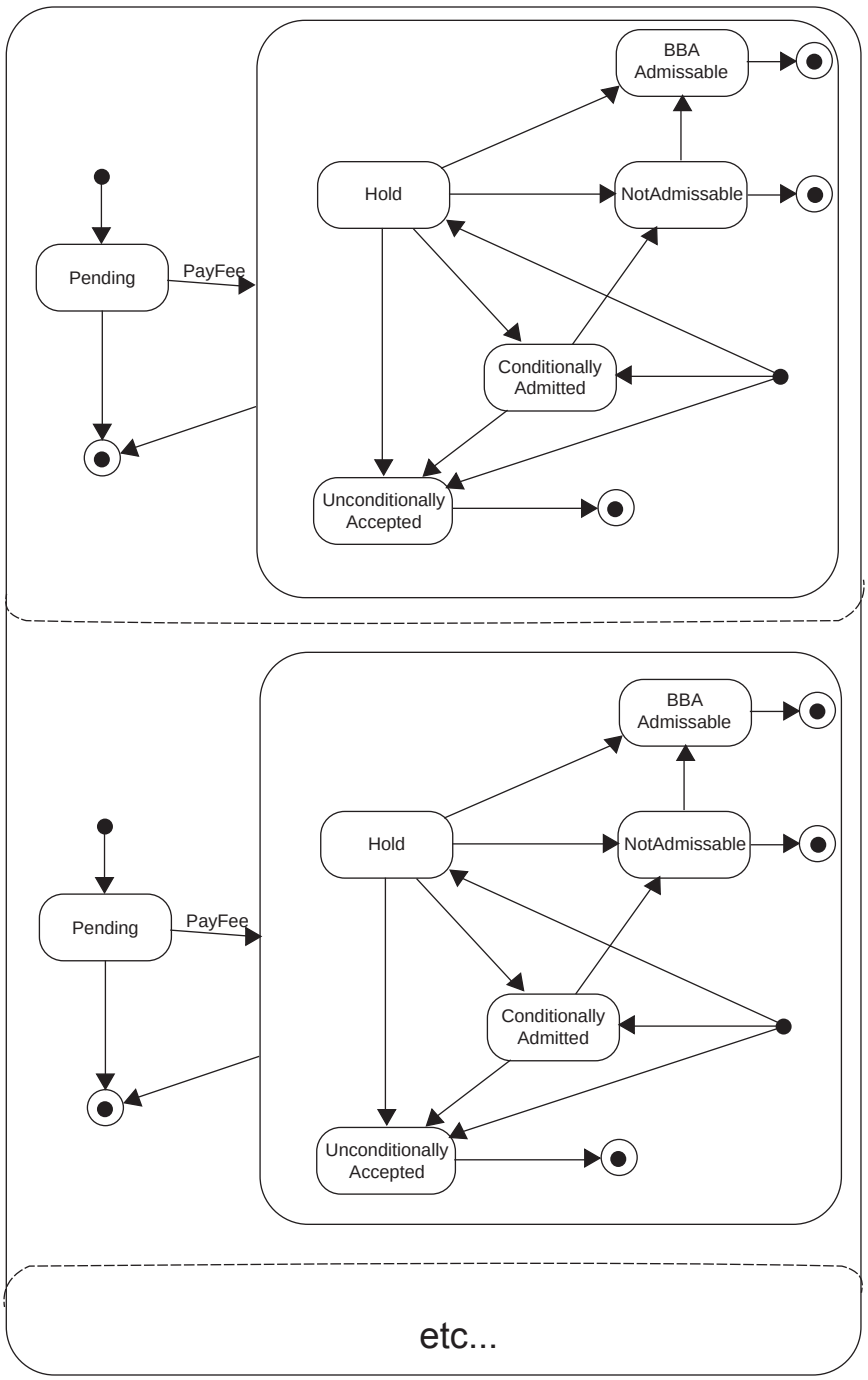
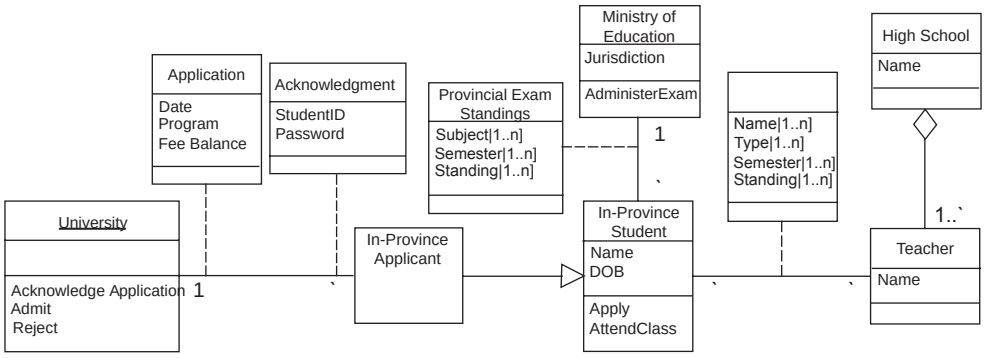


Figure 5. Excerpt of the case study class diagram

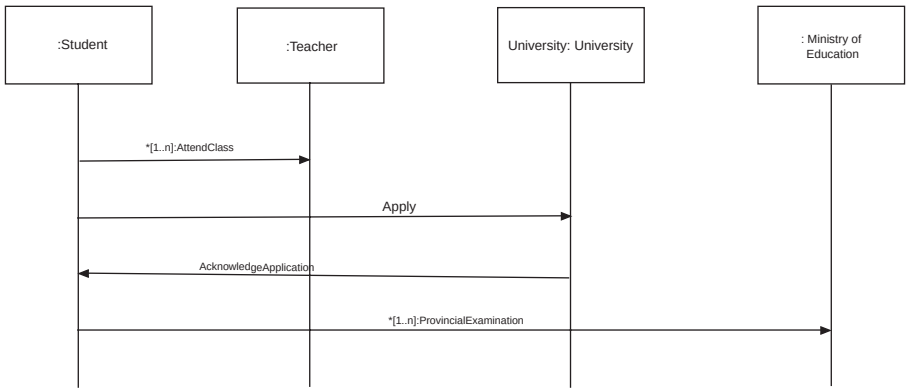


related to the university), and it is the university that holds the state information. Thus, modeling the admission domain with the view that admission information is part of the university state leads to a more realistic model. However, this may not be intuitively obvious to software designers or programmers.

The initial models created by the project team did not provide any operations. Attributes were assumed to be provided with accessor operations. Either approach hides the semantics of the changes inside any behavior implementations. Hence, this information might not be "visible" for the domain understanding and conceptual modeling process, and might be implicitly assumed by the software designer.

The newly developed models included operations for all object classes, developed based on the requirements to express qualitative and quantitative change. The former was derived from the different sub-classes that students can be in, the latter as expressed in state charts. In this way, the capabilities of an object of a given class become explicit and clear. Moreover, as these operations are related to external stimuli, modeled as messages in the sequence diagrams (e.g., Figure 6), the ordering of the changes expressed by methods can be deduced. For example (Figures 5, 6), it is clear that an In-Province Student (students refer to high-school, rather than university students in this domain) can become an applicant. The university can then process the application of the applicant and the Ministry of Education can administer exams for the In-Province Student. This information was lacking in the model developed by the project team. Explicating it in the concep-

Figure 6. Excerpt of the case study sequence diagram



tual model improved understanding and allowed critique and assessment of the correctness of the information about the application domain.

Subsequent discussion with two project team members, the project lead (LF) and the lead developer (CH) of the project, were undertaken to corroborate and confirm these observations. Both suggested that the newly developed models indeed included more information about the domain under study. Both team members commented on the fact that the original model contained many hidden and implicit assumptions:

We relied a lot on assumptions that were never written down in the model ... yours is more comprehensive. (CH - lead developer)

Ours [models] have all sorts of stuff around that is assumed but not modeled. (LF - project lead)

This may be in part due to the additional information included in the model. Note that this is contextual information, that is, information about aspects of the domain that would not be reflected in the design of the software. As such, this information would be useful in allowing team members and stakeholders to discover *why* the design for the information system is defined one way or another, even though the information does not itself constitute or directly affect the software design.

The second important purpose for a conceptual model besides the representation of a real world domain is the starting point for software design. The alternative model was seen as appropriate for this purpose:

I don't see any reason why you couldn't just take these [the models] and run with them. (CH)

During the discussion with the project lead, it also became clear that the use of UML without any guidelines was a substantial challenge in the project. Some analysts were aware of the subtle differences between possible interpretation and the difficulties associated with them:

It's normally difficult to model a course object, because it is a relationship ... What do you mean by a course? The curriculum, the interaction, the grade? (LF)

Another interesting finding is that the extensive use of sequence diagrams and modeling of interactions, which is a central aspect of the proposed ontological semantics and rules, was viewed as something that should be included more often in projects but is not done:

They don't get done as official project documents. When developers meet, two thirds of them will use in-official sequence diagrams ... That makes things much clearer. (CH)

This observation suggests that the interaction-centered perspective enforced by the proposed guidelines can make a positive contribution to understanding. Interactions are not officially documented, yet interaction diagrams appear as unofficial tools. This seems to indicate that explicit modeling of interactions, as prescribed by the proposed rules, can help with understanding of the model and the domain. This demonstrates the importance of interactions in analyzing a real-world domain.

When the modeling rules were revealed at the end of the case study, the project lead and lead developer agreed that modeling rules were necessary and helpful, both for guiding the modeling process and for ensuring model quality and consistency:

Such rules would have helped in our group. The rules would tell whether a model is good and can help answer some questions. They seemed like a lot of valid questions to ask. (CH)

Rules can force the modelers to think deeper about what they're modeling. (LF)

In summary, the case study supported the use of the proposed rules and confirmed their potential usefulness in understanding the domain and communicating that understanding through conceptual models.

Discussion

The objective of this work was to extend the use of object-oriented languages from software design to conceptual modeling of an application domain. This required that domain semantics be assigned to language constructs, which was done by defining mappings between language constructs and ontological concepts (Table 4). The article began by examining the central ontological concepts related to change and interaction and mapped them to object-oriented constructs. The ontological elements used include states, state transitions and lawful transformations. With this representation mapping in mind, object-oriented constructs such as sub-states, activity states and partitions were examined. We then re-examined ontological interaction and discussed the object-oriented concept of message passing.

Table 4 shows that most of the basic ontological concepts can be assigned an object-oriented equivalent. This mapping is either a one-to-one relation by definition or has been made so by our proposed modeling rules. This indicates that object-oriented languages can be used to model application domains.

Based on the proposed mappings, the article explored the consequences of transferring ontological assumptions to the language, resulting in a set of rules. These rules are applicable to object-oriented languages when they are used for conceptual modeling. However, they may not be applicable to, nor are they intended for, using object-oriented languages for software design. Yet, the rules do not contradict existing rules for using object constructs for software design.

These results extend well-known and widely used object-oriented languages into the domain of business analysis and conceptual modeling. The use of the same language for both domain modeling and software design can bridge the gap between system analysis and software design, often called "impedance mismatch" (Cilia et al., 2003; Kolp et al., 2002; Roe, 2003; Rozen & Shasha, 1989). Traditionally, the use of different languages required either a translation of the conceptual model to the software design model, or a relatively independent development of the software model based on the designer's understanding of the problem. Using the same language eliminates the translation step (Evermann & Wand, 2005a). Thus, it can eliminate a possible source of errors, inadvertent alterations, omissions, and so on. Figure 7 shows the traditional

Table 4. Summary of interpretations related to behavior and interaction

Ontological Concept	**Object-Oriented Construct**	**Remarks**
State	State	Connection with attributes of objects made.
State	Sub-state	
State	Action State	Action states are composite states.
Transition	Transition	
Transition	Operation	
Lawful Transformation	Method	
Interaction	Message	Not a substantial thing.

process. The domain is represented in a domain model, using a domain modeling language (1). This is then translated, a potentially error-prone step, to a software model in different language (2), which is then transformed to software (3).

Figure 8 shows software development without translating between modeling languages. The modeling language elements are interpreted with respect to the domain (1) and used to represent the domain using a common modeling language (2). Then, the semantics of the modeling language are assumed with respect to software (3) and the same model is interpreted as a software design model (4).

Domain analysts as well as software developers can use the same familiar language to communicate not only amongst themselves but also between the two groups, reducing the potential for misunderstandings and misinterpretation of diagrams.

Additionally, the use of the modeling rules leads to further benefits. Application of ontological modeling rules have been shown to reduce the variability in conceptual models (Hadar & Soffer, 2006). A reduction in the range of possible models of a domain allows easier and more effective stakeholder communication and can improve the consensus of stakeholders on how to describe the domain. It can also promote convergence of model interpretations, thus helping to ensure that stakeholders share the same problem and domain understanding.

CONCLUSION

The research results were tested on a case study that demonstrated the feasibility and applicability of the proposed semantics and modeling rules for medium size projects. The resulting models were found to be helpful and the modeling rules useful in the construction of the domain model.

Finally, we note that the results depend on the ontological model assumed. Thus, their validity can only be demonstrated by empirical methods. Such methods would include in particular, controlled experiments and realistic case studies. While experiments can test the usability of the rules and clarity of the resulting models (Evermann & Wand, 2006), case studies are required to examine the applicability and usefulness of the method.

Figure 7. Traditional process using model translations, from (Evermann & Wand, 2005a)

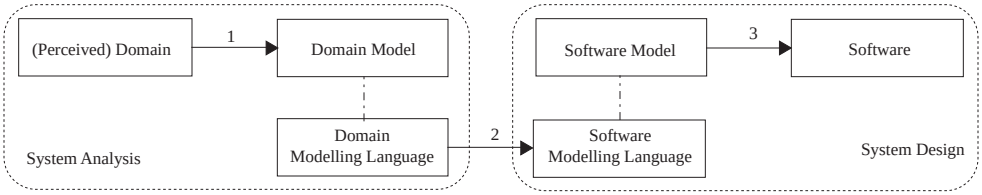
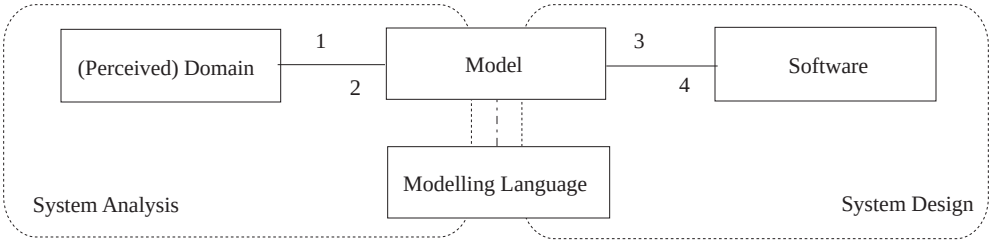


Figure 8. Process without model translation, using common modeling language, from (Evermann & Wand, 2005a)



REFERENCES

Angeles, P. (1981). *Dictionary of philosophy.* New York: Harper Perennial.

Bodart, F., Patel, A., Sim, M., & Weber, R. (2001). Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Research, 12*(4).

Boland, R. (1985). Phenomenology: A preferred approach to research in information systems. In E. Mumford, R. Hirschheim, G. Fitzgerald & T. Wood-Harper (Eds.), *Research methods in information systems.* Elsevier.

Booch, G. (1994). *Object oriented analysis and design with applications.* Redwood City, CA: Benjamin/Cummings.

Bunge, M. A. (1977). *Ontology I: The furniture of the world* (Vol. 3). Dordrecht, Holland: D. Reidel Publishing Company.

Bunge, M. A. (1979). *Ontology Ii: A world of systems* (Vol. 4). Dordrecht, Hollond: D. Reidel Publishing Company.

Chalmers, M. (2004). Hermeneutics, information, and representation. *European Journal of Information Systems, 13*, 210-220.

Cilia, M., Haupt, M., Mezini, M., & Buchmann, A. (2003). *The convergence of AOP and active databases: Towards reactive middleware*. Paper presented at the International Conference on Generative Programming and Component Engineering GPCE.

Coad, P., & Yourdon, E. (1990). *Object-oriented analysis.* Englewood Cliffs, NJ: Yourdon Press.

Coad, P., & Yourdon, E. (1991). *Object-oriented design*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Cockroft, S., & Rowles, S. (2003). *Ontological evaluation of health models: Some early findings.* Paper presented at the 7th Pacific Asia Conference on Information Systems PACIS, Adelaide, Australia.

Dobing, B., & Parsons, J. (2006). How the UML is used. *Communications of the ACM, 49*(5).

Dobing, B., & Parsons, J. (2008). Dimensions of UML diagram use: A survey of practitioners. *Journal of Database Management, 19*(1).

Dussart, A., Aubert, B. A., & Patry, M. (2004). An evaluation of inter-organizational workflow modeling formalisms. *Journal of Database Management, 15*(2), 74-104.

Evermann, J. (2005). *The association construct in conceptual modeling - An analysis using the bunge ontological model.* Paper presented at the 17th International Conference on Advanced Information Systems Engineering, Porto, Portugal.

Evermann, J., & Wand, Y. (2005a). Ontological semantics and formal syntax. *IEEE Transactions on Software Engineering, 31*(1), 21-37.

Evermann, J., & Wand, Y. (2005b). Ontology-based object-oriented business modelling: Fundamental concepts. *Requirements Engineering, 10*(2), 146-160.

Evermann, J., & Wand, Y. (2006). Ontological modelling rules for UML: An empirical assessment. *Journal of Computer Information Systems, 47*(1).

Gadamer, H.-G. (1976). *Philosophical hermeneutics*: University of California Press.

Gemino, A. (1999). *Empirical comparisons of systems analysis modeling techniques.* Unpublished PhD Thesis, The University of British Columbia, Vancouver, BC.

Green, P., & Rosemann, M. (2000). Ontological analysis of integrated process modelling. *Information Systems, 25*(2).

Gruninger, M., & Lee, J. (2002). Ontology applications and design. *Communications of the ACM, 45*(2).

Guarino, N., & Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM, 45*(2), 61-65.

Hadar, I., & Soffer, P. (2006). Variations in conceptual modeling: Classification and ontological analysis. *Journal of the AIS, 7*(8).

Jacobson, I. (1992). *Object-oriented software engineering: A use case driven approach*. Wokingham, MA: Addison-Wesley.

Kolp, M., Giorgini, P., & Mylopoulos, J. (2002). *Information systems development through social structures.* Paper presented at the International Conference on Software Engineering and Knowledge Engineering SEKE.

Kuhn, T. (1996). *The structure of scientific revolutions.* Chicago, IL: The University of Chicago Press.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 99-118.

Myers, M. (1995). Dialectical hermeneutics: A theoretical framework for the implementation of information systems. *Information Systems Journal, 5*, 51-70.

Mylopoulos, J. (1992). Conceptual modeling and telos. In P. Locoupoulos & R. Zicari (Eds.), *Conceptual modeling, databases, and cases.* New York, NY: John Wiley & Sons, Inc.

Noy, N. F., & Hafner, C. D. (1997). The state of the art in ontology design: A survey and comparative review. *AI Magazine, 18*(3), 53-74.

OMG. (2005). *Unified modeling language: Superstructure, version 2.0* (No. formal/05-07-04): The Object Management Group.

Opdahl, A., & Henderson-Sellers, B. (2001). Grounding the OML meta-model in ontology. *The Journal of Systems and Software, 57*(2), 119-143.

Opdahl, A., & Henderson-Sellers, B. (2002). Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modeling, 1*(1), 43-67.

Opdahl, A., Henderson-Sellers, B., & Barbier, F. (1999). An ontological evaluation of the OML metamodel. In E. Falkenberg & K. Lyytinen (Eds.), *Information system concepts: An integrated discipline emerging*: IFIP/Kluwer.

Parsons, J., & Wand, Y. (1991). *The object paradigm - two for the price of one?* Paper presented at the Workshop on Information Technology and Systems WITS.

Parsons, J., & Wand, Y. (1997). Using objects for systems analysis. *Communications of the ACM, 40*(12), 104-110.

Prasad, A. (2002). The contest over meaning: Hermeneutics as an interpretive methodology for understanding texts. *Organizational Research Methods, 5*, 12-33.

Reinhartz-Berger, I., & Sturm, A. (2008). Enhancing UML models: A domain analysis approach. *Journal of Database Management, 19*(1).

Ricoeur, P. (1976). *Interpretation theory: Discourse and the surplus of meaning.* Fort Worth, TX: The Texas Christian University Press.

Roe, P. (2003). *Distributed XML objects.* Paper presented at the Joint Modular Languages Conference JMLC.

Rosemann, M., & Green, P. (2000). Integrated process modelling: An ontological analysis. *Information Systems, 25*(2), 73-87.

Rozen, S., & Shasha, D. (1989). Using a relational system on Wall Street: The good, the bad, the ugly, and the ideal. *Communications of the ACM, 32*(8), 988-993.

Rumbaugh, J. (1991). *Object oriented modeling and design.* Englewood Cliffs, NJ: Prentice Hall.

Smith, B., & Welty, C. (2001). *Ontology: Towards a new synthesis.* Paper presented at the Second International conference on Formal Ontology and Information Systems FOIS, Qgunquit, Maine.

Smolander, K., & Rossi, M. (2008). Conflicts, compromises, and political decisions: Methodological challenges of enterprise-wide e-business architecture creation. *Journal of Database Management, 19*(1).

Soffer, P., Golany, B., Dori, D., & Wand, Y. (2001). Modelling off-the-shelf information systems requirements: An ontological approach. *Requirements Engineering, 6*(3), 183-199.

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods, and applications. *Knowledge Engineering Review, 11*(2).

Van Hentenryck, P. (1989). *Consistency techniques in logic programming.* Cambridge MA: MIT Press.

Wand, Y. (1989). A proposal for a formal model of objects. In W. Kim & F. Lochovsky (Eds.), *Object-oriented concepts, languages, applications and databases* (pp. 537-559): ACM Press/ Addison-Wesley.

Wand, Y., Storey, V., & Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems, 24*(4), 494-528.

Wand, Y., & Weber, R. (1989). An ontological evaluation of systems analysis and design methods. In E. Falkenberg & P. Lindgreen (Eds.), *Information system concepts: An in-depth analysis*: Elsevier Science Publishers, B.V.

Wand, Y., & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems, 3*, 217-237.

Wand, Y., & Weber, R. (1995). On the deep structure of information systems. *Information Systems Journal,* (5), 203-223.

Weber, R., & Zhang, Y. (1996). An analytical evaluation of Niam's grammar for conceptual schema diagrams. *Information Systems Journal, 6*(2), 147-170.

ENDNOTES

1. The first author was with the School of Information Management, Victoria University of Wellington, Wellington, New Zealand for the duration of this research.
2. The terminology changed in UML 2.0. Here, operations are associated with behavior. Opaque behavior is a special class of behavior and represents what was called a method in UML 1.5.
3. Bunge's ontology is applicable to this domain comprising a university, high schools and other composite objects. According to Bunge (1977), certain things are composed of other things (composite things, pg. 114, Definition 3.4). Furthermore, if the parts of things interact, the composite thing is known as a system (Bunge, 1979, pg. 6, definition 1.1). Moreover, Bunge (1979) expounds a hierarchy of systems (ontological levels) that includes psychological and social/technical systems. For example, a university department is neither a simple (basic) thing, nor a mere composite, but a system with interacting ("connected") parts. The BWW ontology has been applied to a diverse range of domains, such as enterprise systems, workflows and business processes, etc., showing that it is applicable to organizational and social phenomena.

This work was previously published in Journal of Database Management, Vol. 20, Issue 1, edited by K. Siau, pp. 48-77, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 2.2
An Ontology Based Representation of Software Design Patterns

Jens Dietrich
Massey University, New Zealand

Chris Elgar
SolNet Solutions Limited, New Zealand

ABSTRACT

This chapter introduces an approach to define Design patterns using semantic Web technologies. For this purpose, a vocabulary based on the Web ontology language OWL is developed. Design patterns can be defined as RDF documents instantiating this vocabulary, and can be published as resources on standard Web servers. This facilitates the use of patterns as knowledge artefacts shared by the software engineering community. The instantiation of patterns in programs is discussed, and the design of a tool is presented that can x-ray programs for pattern instances based on their formal definitions.

INTRODUCTION

Design patterns are artefacts used to share knowledge across particular communities. Software Design patterns (Gamma, Helm, Johnson, & Vlissides, 1995) describe structural and behavioural properties of software, and are used by the software engineering community in order to exchange knowledge about software design. Currently, this is done the old-fashioned way: patterns are published in books or on Web sites, and consumed by people reading and applying them. This includes recognising pattern instances in programs in order to comprehend the structure of complex programs, and using patterns as design blueprints in order to address a specific design problem.

The main limitation with this modus operandi is the lack of tool support—patterns have to be found, and good and appropriate patterns have to be selected manually. A more effective, tool supported way of processing patterns (in particular instantiation and recognition) requires a formal representation of the patterns that supports reasoning about patterns. There are some obvious con-

tenders of modeling frameworks and languages that could be used for this purpose. This includes first order predicate logic, higher order logic, and UML based modeling languages (either profiles or separate, MOF based modeling languages). In recent years, several approaches to Design pattern formalisation have been proposed based on these choices. While these representations allow us to reason about the internal structure of a pattern, there is limited support for reasoning about the patterns themselves, their metadata, and relationships to other patterns. Moreover, logic based and UML based approaches usually lead to monolithic, static models. While this has advantages (for instance, to ensure the consistency of the models) it does not support the sharing of patterns in an open environment like the Internet. Here, knowledge is distributed and inherently inconsistent, and additional means are needed to deal with inconsistency. This is one of the key issues addressed by the Semantic Web initiative endorsed by the W3C (Berners-Lee, Hendler, & Lassila, 2001). The Semantic Web is based on the idea of a distributed, open knowledge base where everybody can publish knowledge in the form of simple subject-predicate-object assertions. This leads necessarily to inconsistencies that must be resolved. The key to solving this problem is the prioritisation of knowledge based on metadata annotations—knowledge is selected based on the author, the time of creation, and other explicit and harvested metadata.

Another issue is whether it is realistic to hope that the software engineering community will adopt one particular model or vocabulary to represent patterns. It seems more likely that there will be multiple models, each supported by a particular community, standard body, or vendor group. Mappings defining transformations between instances of the respective models will have to be developed. In the case of formal logic, this is only possible with higher order constructs describing the relationship between different predicates, functions, and types. For UML based models, standards to transform instances of different meta models are only emerging now in the realm of model driven architecture (MDA), in particular QVT ("MOF QVT Final Adopted Specification," 2005). On the other hand, modern ontology languages like OWL have built-in support to express the relationship between different ontologies.

We claim that a modeling language suitable to publish Design patterns on the Web should meet the following requirements:

1. A formal semantics is needed in order to safeguard reasoning about the patterns.
2. Pattern definitions must be easy to process by tools. In particular, this is the case if an XML based serialization format is supported.
3. There must be facilities to describe the relationships with alternative models, languages, or pattern vocabularies.
4. The physical distribution of pattern definitions and the separation of schema and instances must be supported. The pattern definition language itself should be deployed as a network resource and single pattern definitions should reference this resource. In particular, it should be possible to validate the pattern definitions against such a central schema. This is similar to the validation of XML documents against their XML schema or DTD. Furthermore, patterns refining other patterns can also refer to them as network resources, and clients can easily resolve these references using standard network clients.

It appears that a pattern deifintion language based on the Web ontology language OWL (McGuinness & Harmelen, 2004) meets these requirements. Firstly, OWL has a formal semantics (Patel-Schneider, Hayes, & Horrocks, 2004) and built-in derivation rules which support safe reasoning about patterns. This can be used to infer additional knowledge from models or to check them for consistency. Secondly, OWL ontologies

(both schemas and instances) can be serialised as XML using the RDF/XML syntax, and a growing number of tools are available to facilitate the work with OWL knowledge bases, including editors like Protégé (Knublauch, Musen, & Rector, 2004), APIs like Jena ("Jena – A Semantic Web Framework for Java," 2006), documentation tools like OWLDoc ("OWLDoc," 2006) and reasoners such as Fact (Horrocks, 1999), Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2006) and Racer (Haarslev & Moeller, 2001). Thirdly, OWL has built-in constructs that can be used to describe the relationship between different vocabularies. Fourthly, OWL has a simple yet powerful import facility (owl:imports[1]) that can be used to reference the assertions made in external knowledge bases. In particular, this feature can be used to link ontology instances with a central ontology schema.

For this reason we believe that OWL is a suitable language to formalise Design patterns. In this chapter, we will present an ontology that can be used to define patterns. We discuss how metadata can be used to reason about patterns, and sketch how tools can process these definitions. This leads to a discussion on Design pattern instances. The results presented here are mainly based on Dietrich and Elgar (2005a, 2005b).

BACKGROUND

The Semantic Web initiative aims at turning the Web currently focusing on human end users into a Web that also contains content that can be processed by machines (Berners-Lee et al., 2001). The basic infrastructure is provided by the resource description framework RDF (Klyne & Carroll, 2004)—a simple language that uses subject-predicate-object constructs to describe resources. This strongly resembles the structure of the Web for humans consisting of node-link-node triples with Web pages being the nodes and the hyper links connecting them being the edges.

Meaning is added to RDF by using an ontology language such as RDFS, DAML-OIL, or OWL. We focus on OWL as it is the most expressive of these languages, and has been standardised by the W3C. Using OWL, resources can be associated with resource types (classes), and predicates[2] can be defined for associating instances of certain classes. Furthermore, several constraints can be expressed restricting properties, and rules can be added to allow the assertion of new, additional properties. The formal semantics of OWL defines the precise meaning of these language constructs, and forms the sound base for OWL reasoners (Patel-Schneider et al., 2004). By describing how concepts are used in context, ontologies are defined—shared terminologies or "specifications of a conceptualization" (Gruber, 1993).

In Dietrich and Elgar (2005a), we have developed a terminology expressive enough to define basic Design patterns, and open enough to be extended whenever more expressiveness is required. This openness is important to define patterns that have a limited scope. Traditional pattern catalogues like the GangOf4 catalogue aim at programming language independent patterns, and their scope is object-oriented programming in general. However, it does make sense to publish patterns that refer to features not present in all languages, such as inner classes and annotations (Java) or class instance variables (Smalltalk). The scope of these patterns is a certain programming language, and they are shared within their respective communities. While the core ontology presented here does not directly support such PL specific patterns, it can be easily extended in various ways to do so.

Using OWL (or RDF for that matter) has another advantage: patterns are identified by uniform resource identifiers (URIs). These URIs are usually composed from the pattern name and a name space. This allows distinguishing between several flavours of one pattern all using the same name. A good example is Iterator. The definitions presented in Gamma et al. (1995) and

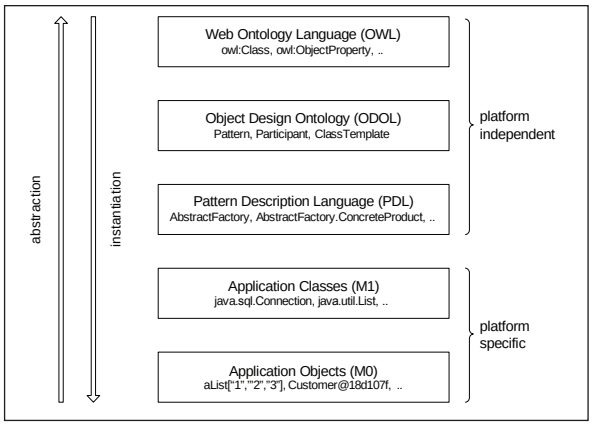
Grand (1998) differ—the GangOf4 definition has an additional first() method. The Iterator pattern used in Java is again different—there is an additional, optional remove() method. Adding a name space and referencing the pattern with the full URI immediately removes the ambiguity from the pattern definitions.

AN ONTOLOGY FOR SOFTWARE DESIGN PATTERNS

Metamodeling Architecture

The formal definition of Design patterns, their participants, and the properties of and relationships between the participants proposed here is based on an OWL ontology, that is, on a system of OWL classes, their properties, and relationships. These classes are instantiated in Design pattern definitions. Design patterns themselves do not contain classes, methods, and similar programming artefacts, but only placeholders or role names for such artefacts. In other words, Design pattern definitions contain typed variables that are instantiated by a particular pattern instance found in a concrete program. It has been pointed out by Eden (2002a) that for this reason UML class models are not suitable to represent Design patterns. The same applies to other M1 models—they contain concrete classes, not variables. In Dietrich and Elgar (2005a) we have proposed a meta model architecture to tackle this problem (Figure 1). The lower layers correspond to MOF M0 (application objects) and M1 (application classes, members, and associations). M1 artefacts instantiate pattern participants defined using the pattern description language (PDL). These are the variables found in pattern definitions like the AbstractFactory and the AbstractProduct in the AbstractFactory (Gamma et al., 1995) pattern. These variables are typed. The types are constraints restricting the kind of (M1) artefact that can instantiate these variables. These types are modeled in the object design ontology layer

Figure 1. Metamodeling architecture



(ODOL). In ODOL, the (OWL) ontology defined is the base for the pattern definitions. It contains classes such as ClassTemplate and MethodTemplate and their relationships, as well as the Pattern class representing patterns themselves. The meta model of ODOL is OWL– ODOL contains instances of OWL classes (for instance, owl:Class and owl:ObjectProperty).

The Ontology Schema

The ODOL ontology presented here is a revised version of the ontology presented in Dietrich and Elgar (2005a), adding pattern refinement and aligning it with the refactoring meta model proposed in Tichelaar, Ducasse, Demeyer, and Nierstrasz (2000). This adds significant expressiveness to the ontology, as it makes it possible to attribute relationships between participants such as Access, Association, and Invocation. Figure 2 does not show the full ontology, and in particular the data type properties (owl:DatatypeProperty) are omitted for space reasons. This includes the abstractness of class templates, access modifiers for member templates, cardinalities for associations, and similar properties. For the complete ontology, the reader is referred to "The ODOL OWL Ontology" (2006). The graph syntax used in Figure 2 and Figure 3 should not be confused with the RDF instantiation syntax. While the nodes do

Figure 2. The ODOL ontology: Class hierarchy

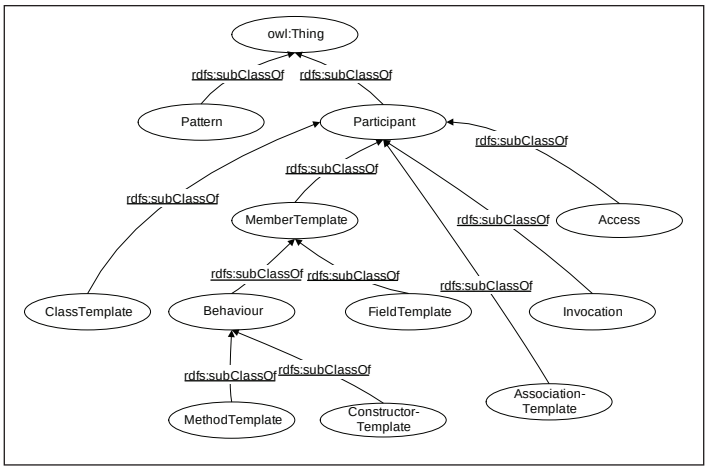
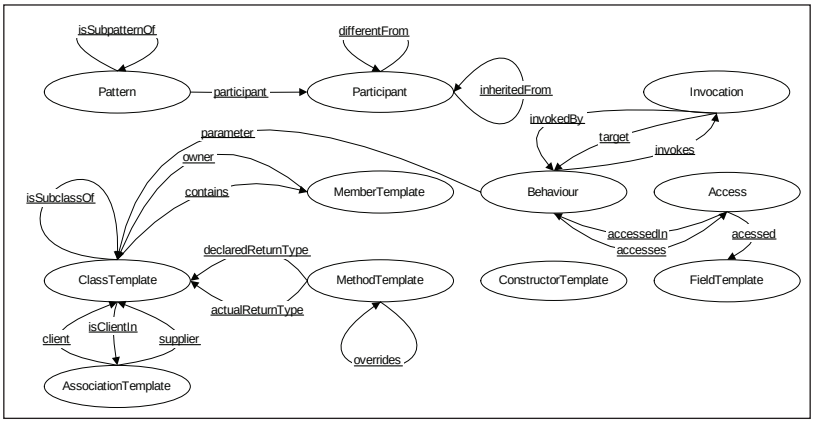


Figure 3. The ODOL ontology: Object properties



represent RDF resources (the OWL classes that are part of ODOL), the edges are object properties (owl:ObjectProperty) having the start node as the domain (rdfs:domain) and the end node as the range (rdfs:range). This is different from standard RDF graphs. We emphasise this difference by underlining the property names.

Built-In Rules

The model has a number of constraints describing its intended meaning. In particular, object properties are associated with OWL property types. For these types, the OWL semantics defines rules that can be used by reasoners to check the consistency of the model, and to infer additional assertions. This implies that pattern definitions do not have to include assertions that are redundant according to those rules. However, it is useful to have these redundant statements when processing pattern definitions, as this results in a better connected graph that is easier to process by applications. Table 1 shows some of the built-in rules.

In addition to these rules, the ontology contains references to *informal* definitions of some of the concepts defined. For instance, the (XML version of the) ontology contains the following references to Web sites intended for end users:

Table 1. ODOL rules (selection)

Property	Domain	Range	Rule(s)
isSubPatternOf	Pattern	Pattern	transitive
contains	ClassTemplate	MemberTemplate	inverseOf “owner”
owner	MemberTemplate	ClassTemplate	inverseOf “contains”, functional
associationClient	AssociationTemplate	ClassTemplate	functional, inverseOf “isClientIn”
associationSupplier	AssociationTemplate	ClassTemplate	functional
isClientIn	ClassTemplate	AssociationTemplate	inverseOf “associationClient”
isSubclassOf	ClassTemplate	ClassTemplate	transitive
overrides	MethodTemplate	MethodTemplate	transitive

```
<owl:Class rdf:ID=”Pattern”>
      <owl:sameAs
        rdf:resource=”http://en.wikipedia.org/wiki/Design_pattern_
        %28computer_science%29”/>
      <owl:sameAs
        rdf:resource=”http://hillside.net/patterns/definition.html”/>
</owl:Class>
```

Code Snippet 1: Informal definition of pattern in ODOL.

While the formal rules listed in Table 1 can be used directly by tools to process pattern definitions, these statements are intended for human end users like application programmers in order to help them to write correct applications consuming the patterns defined using this ontology. Using the terminology proposed by Uschold (2003), ODOL contains *formal* semantics intended for machine processing as well as *informal* semantics intended for human processing.

Defining Patterns

While the Design pattern ontology resembles the classical UML meta model, it contains some features that add additional expressiveness needed to represent Design patterns. For instance, the AbstractFactory pattern (Gamma et al., 1995) contains a ConcreteFactory method that overrides an AbstractFactory method and returns an instance of a ConcreteProduct class. The declared return type of the ConcreteFactory method instantiation equals the declared return type of the abstract method instantiation. Therefore, we must be able to distinguish between declared return type and the actual return type. In Code Snippet 2, the declared return type of the create method is the Connection class, while the actual return type is the ConnectionImpl class.

```
1. public Connection create () {
2.      return new ConnentionImpl () ;
3. }
```

Code Snippet 2: A concrete factory method.

To address this problem, the ontology has two object properties associating method templates with return types (class templates): declaredReturnType and actualReturnType. There is an implicit constraint relating these two properties that can be expressed as a rule:

$$\text{actualReturnType}(m,t_1) \;\&\; \text{declaredReturnType}(m,t_2) \Rightarrow \text{isSubclassOf}(t_1,t_2)$$

Currently, this kind of rule is not part of the

ontology. Languages like OWL do not support custom derivation rules, although this area is an active area of research with emerging standards such as RuleML (Boley, Tabet, & Wagner, 2001) and SWRL (Patel-Schneider et al., 2004). Therefore, the isSubclassOf property cannot be inferred and has to be added explicitly to the pattern definition, and it remains the responsibility of the client to check the constraint. On the instance level, this would usually be done by a compiler.

The definition of AbstractFactory (without data type properties) is depicted in Figure 4 and Figure 5. The types are defined in the central ontology deployed at a certain URL. This resource is explicitly imported using an owl:imports directive. This allows maintaining a central ontology that can be referenced by different patterns.

Figure 4. The AbstractFactory pattern (participants and their types)

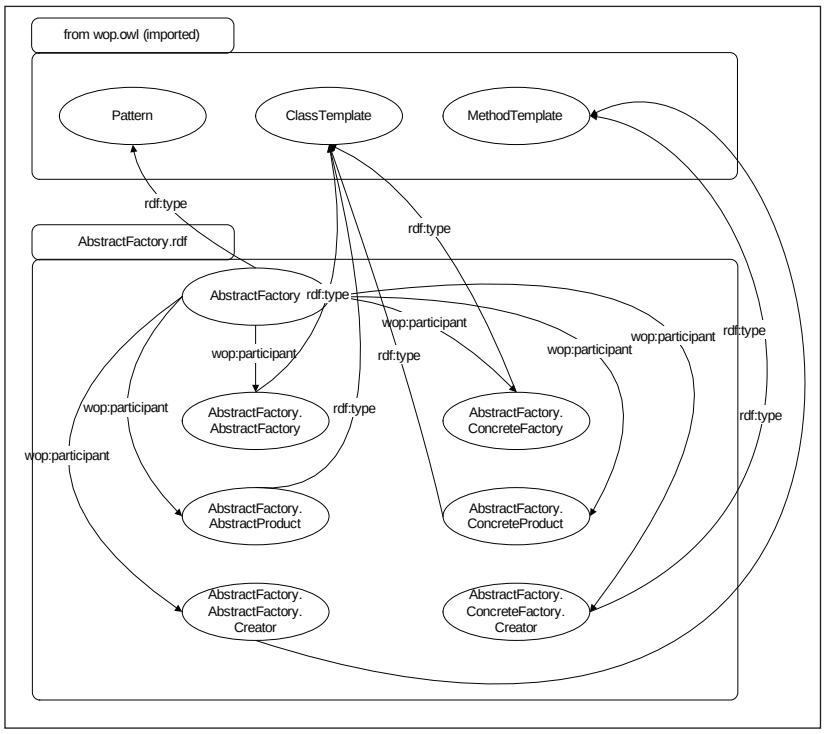
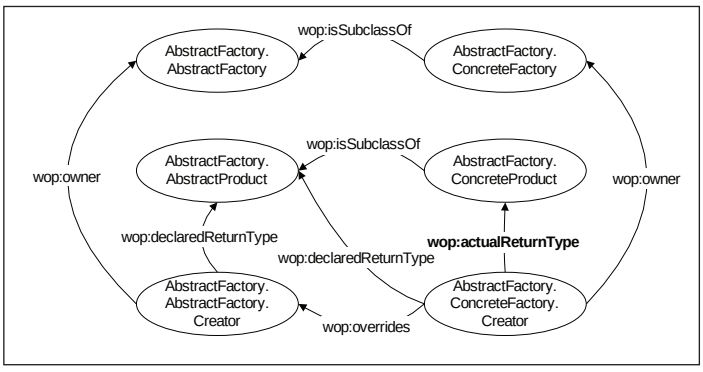


Figure 5. The AbstractFactory pattern (other object properties)



Pattern Refinement

Pattern refinement defines an inheritance relationship between patterns. Using refinement, patterns inherit participants from parent patterns, and add new participants or new relationships. The simplest form of refinement is a pattern definition that imports another pattern, references all of the participants using the wop:participant property, and adds new participants and properties.

In addition to this, ODOL contains additional constructs to support multiple inheritance and joins between participants inherited from different parents. The isSubPatternOf property associates a pattern with a parent pattern. This property is not functional, that is, multiple inheritance is explicitly permitted. The semantics of this property is that all participants and their properties of the parent pattern are appended to the definition of the importing pattern. Using a derivation rule, this can be expressed as follows:

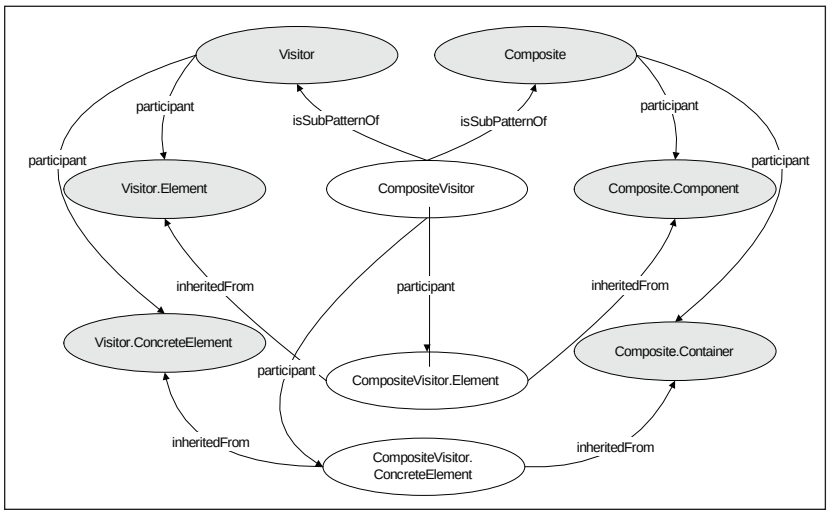
$$isSubPatternOf(pat_1,pat_2) \ \& \ participant(pat_2,part) \Rightarrow participant(pat_1,part)$$

The inheritedFrom relationship identifies participants with participants inherited from parents. The semantics is simple, the bindings for these participants must be the same. This property is also not functional, that is, a "local" participant can be associated with more than one imported/ inherited participant. This allows joining inherited participants, as depicted in Figure 6. In this example, the CompositeVisitor is a Visitor (Gamma et al., 1995) visiting a Composite (Gamma et al., 1995) structure. It refines both the Visitor and the CompositeVisitor pattern.

Multiple inheritance can lead to conflicts, for instance if participants with contradicting properties are defined. It is again part of the informal semantics of ODOL to require that inheritedFrom can only associate properties with no conflicting ODOL properties. That means that:

1. If two joined participants have an ODOL data type property, then the values of these participants must be the same.
2. The graph containing participants and ODOL object properties obtained by replacing the participants referenced using the inheritedFrom predicate by the referencing participants must be consistent.

Figure 6. Definition of CompositeVisitor using pattern refinement (imported participants are represented by the shapes with a darker colour)



An example for how condition 2 could be violated is the following situation: let R be a functional property in ODOL, and let R(a,b) and R(c,d) be statements in two pattern definitions P1 and P2 (written in relational syntax). If P is a sub pattern of both P1 and P2 with conditions joining a and c, but no condition joining b and d, the "functional" condition for R would be violated.

Note that both integrity conditions are explicitly referring to the ODOL vocabulary. Conflicts in other vocabularies might not be as critical. In particular, conflicts concerning metadata (such as dc:Description or dc:Creator) are acceptable.

Attaching Meta Information

Reasoning about patterns becomes important if patterns are published in an open, essentially uncontrolled and unmoderated environment like the Internet. In particular, this reasoning will facilitate the selection of trustworthy knowledge. There are no special constructs in ODOL to support metadata, instead standard vocabularies are used. In particular, the Dublin Core ("The Dublin Core Metadata Initiative," 2006) properties can be used to annotate the pattern instances in pattern definitions. Using these annotations, derivation rules can be used to select all GangOf4 patterns (using an assertion like "dc:Creator=GangOf4"). Often, additional predicates have to be used in these rules, either based on explicit meta information, or on harvested meta information such as:

1. **isSecure:** Has an HTTPS connection been used to download patterns?
2. **isKnownBy:** Is the pattern resource referenced by the Web sites of pattern communities such as hillside.net or on vendor Web sites like ibm.com.
3. **isWellknown:** Is the pattern definition known as well referenced by Google? That is, does the Google query link:<aPatternURL> return more than 42 results?

INSTANTIATING DESIGN PATTERNS

Pattern Instances

Defining the patterns in OWL/RDF enables us to deploy the pattern definitions on standard Web servers, where they can be found and accessed by applications and standard search engines. There are several use cases describing how to process Design patterns, the most important one being Design pattern recognition in programs. This enables users to X-Ray complex software packages in order to improve program comprehension. It may also be used to measure Design pattern density in software package in order to quantify design quality: the use of best practise Design patterns can be considered as desirable as it makes it easier for teams to communicate about design.

The Design pattern definitions can be easily translated into derivation rules in typed first order logic with the pattern itself as the predicate in the head of the respective rule, and the pattern participants as free variables. The precise definition is presented in Dietrich and Elgar (2005b), and we use the AbstractFactory pattern to explain the transformation. The pattern has the following participants: AbstractFactory (af), AbstractProduct (ap), ConcreteFactory (cf), ConcreteProduct (cp), AbstractFactory.Creator (afc), ConcreteFactory.Creator (cfc). These participants are associated with typed variables, and the types are the respective ODOL classes. The head of the rule is Abstract Factory(af,ap,cf,cp,afc,cfc). The prerequisites of the rule are:

1. isSubclassOf(cf, af)
2. isSubclassOf(cp, ap)
3. contains(af, afc)
4. contains(cf.cfc)
5. declaredReturnType(afc,ap)
6. declaredReturnType(cfc,ap)
7. actualReturnType(cfc, cp)
8. overrides(cfc, afc)
9. isAbstract(af,"abstract")

10. isAbstract(afc, ”abstract”)
11. isAbstract(ap,”abstract”)
12. ..

That is, object properties are translated into binary constraints associating participants with other participants (or more precisely, the respective variables), while data type properties are transformed into binary predicates associating participants with string literals. The typing imposes additional constraints that can be seen as unary predicates. In the AbstractFactory example, this would lead to additional prerequisites such as isClass(af), isMethod(afc), and so forth. Similar to the mechanisms proposed by other authors (Beyer, Noack, & Lewerentz, 2003; Eden, 2002a; Kraemer & Prechelt, 1996), Design pattern detection becomes a matter of instantiating the variables in these rules. The values binding the variables are artefacts from the program to be analysed. Artefacts include classes and their members, but also associations, method invocations, and field accesses. If Var(P) denotes the set of participants in a pattern P and A(Progr) the set of artefacts in a program Progr, then a pattern instance is a function instance:Var(P)→A(Progr).

Instantiation requires a universe of artefacts to be extracted from the analysed program, and a set of ground facts for the predicates used in the rules instantiated using these artefacts. Because the rules are structurally fairly simple (datalog without negation), the only challenge is to find the appropriate fact base. The universe itself must contain representations of the artefacts found in the program. These could either be names, instances of classes of the programming language representing artefacts (such as reflection frameworks), or instances of classes from separate object models describing the structure of programs. As far as the fact base is concerned, there are some fundamentally different approaches that can be used to instantiate predicates. This includes:

1. Using the reflective features of the programming language itself. Most modern PLs offer such a feature, and relationships such as membership, return type, and inheritance can be easily instantiated.
2. Source code analysis. Reflection based analysis is usually not suitable to instantiate associations and invocations. This often requires source code analysis. The respective concepts have been studied comprehensively in the realm of compiler technology. Source code is usually represented by abstract syntax trees (AST), and analysis is done using AST visitors. Using visitors, symbol tables are compiled to resolve the references between artefacts.
3. Naming pattern analysis. Often naming patterns do have meaning, and they can be used for program analysis as well. A good example for meaningful naming patterns is the java bean specification (“The JavaBeans Spec,” 2006).The use of certain prefixes for method names and parameter signatures indicates that there are certain properties of and relationships between objects. An example is the one to many relationship between event sources and event listeners. Tools such as GUI builders use these semantics.
4. Behavioural analysis. Based on the run and observe approach, behavioural analysis can detect relationships not accessible by any of the above methods. This includes the analysis of code that uses reflection, for instance the interpretation of strings as methods. Behavioural analysis requires that the program can be executed “completely,” that is, that each code branch is visited during code execution. In some cases, complete execution is provided by test suites, particularly if test-case driven methodologies like extreme programming (Beck, 1999) are used in conjunction with tools measuring test coverage. Technically, behavioural analysis can take advantage of

debugging APIs or newer approaches used to instrument code (for instance, aspect-oriented programming).

Client Architecture

The WebOf Patterns (WOP) project ("The WebOf-Patterns Project," 2006) is a client that consumes and produces pattern definitions based on ODOL. Technically, the client is an Eclipse plugin. It interprets the pattern definitions in Java, and features the following functionality: a *pattern scanner* with a built-in HTTP client that downloads patterns from a given URL, detects pattern instances in Eclipse Java projects and compiles JavaDoc style (HTML and XML) reports from the findings. The client also supports several pattern aggregations (see page 15). A *pattern browser* generates a graph from pattern definitions that can be navigated in a browser-like manner. A *pattern publishing wizard* allows users to select artefacts, their properties and relationships from projects, give those artefacts abstract role names, and generate the pattern definition as a RDF file. This file can then be deployed on a standard Web server.

The WOP client serves as a reference application for the ontology. It contains guidelines on how to consume the ontology based pattern definitions. This makes it part of the "formal semantics for human processing" (Uschold, 2003). This is similar to software specifications—many specs consist mainly of interfaces (UML, IDL, Java interfaces). The semantics of these interfaces is defined by documents, formal constraints (like OCL), and test suites. Often, a reference implementation is provided that can be considered as part of the semantics—software engineers are referred to the reference implementation if the other parts of the specification leave questions open.

The pattern scanner is the most important part of the WOP client, as it has to map the structures defined in the ontology to the structures found in the Java programming language. This is rather simple for some elements, as there is an obvious homomorphism (structure preserving mapping) between the ontology and the grammar of the programming language—classes have methods, methods have return types, and so forth. It is less obvious for participants that have no direct counterparts in Java, such as Association.

The WOP Client has the following structure:

- Participants are represented by Java classes in the nz.ac.massey.cs.wop package. Example: AssociationTemplate, ClassTemplate, Participant, Pattern.
- Artefacts instantiating the participants are represented by Java classes in the nz.ac.massey.cs.wop.reflect package. Example: Association, JClass, JMethod, and so forth.
- Object properties are represented by classes implementing the interface nz.ac.massey.cs.wop.BinaryConstraint<SOURCE,TARGET> in the nz.ac.massey.cs.wop.constraints package. Example: IS_SUBCLASS_OF, OVERRIDES.
- Data type properties are represented by subclasses of nz.ac.massey.cs.wop.UnaryConstraint<TYPE> in the nz.ac.massey.cs.wop.constraints package. Examples: IS_ABSTRACT_CLASS, IS_PUBLIC_MEMBER.

The type parameters of the generic constraint classes represent the types associated by the respective constraints. This must be consistent with the domain and the range types of the associated object property. The constraints contain methods defining the extension of the respective (unary or binary) predicate in the context of the program investigated. The Eclipse project analysed is represented by an instance of Workspace. This contains a reference to a class loader, the byte code location(s), the source code location(s), and entry points for behavioural analysis. In particular, the BinaryConstraint interface defines the following three methods (SOURCE and TARGET are the generic type parameters):

1. boolean check(SOURCE,TARGET,Workspace)
2. Iterator<SOURCE> getPossibleSourceInstances(Workspace,TARGET)
3. Iterator<TARGET> getPossibleTargetInstances(Workspace,SOURCE)

Code Snippet 3: Methods in BinaryConstraint defining the semantics of an object property in Java

These methods precisely define the semantics of the associated object property. The first method allows verifying whether two given instances instantiate the predicate. The other two methods return an iterator over possible instances if one instance is known. This supports the "navigation" of the predicates. There is no method that computes the entire extension of the predicate (i.e., an iterator over SOURCE-TARGET pairs). This is highly inefficient and can generally be avoided by choosing appropriate algorithms to resolve the constraints.

For instance, the IS_SUBCLASS_OF class implements BinaryConstraint as follows:

1. Both SOURCE and TARGET are JClass, JClass.
2. If instances of both source and target are given, reflection (isAssignableFrom in java.lang.Class) is used to check whether class 1 is a subclass of class 2.
3. If only the source is known, the reflection methods getSuperclass() and getInterfaces() are recursively called, the results are collected in a collection and an iterator over this collection is returned.
4. If only the target is known, all classes in the workspace are scanned to find all subclasses of the target, and the respective iterator is returned.

In general, classes implementing the interface nz.ac.massey.cs.wop.reflect.Analyser provide the structural information needed by the constraint classes. Analysers are organised as a chain, and if an analyser cannot provide the information requested the next analyser is called (ChainOfResponsibility pattern (Gamma et al., 1995)). The analysers currently implemented are based on reflection, (Eclipse) AST analysis, and naming pattern analysis.

The pattern scanner itself uses cost-based scheduling to arrange the constraint in an optimal order. It always attempts to solve "cheap" constraints (like those that can be verified using reflection) first, and arranges constraints so that at least one of the participants is already known. This is facilitated by having redundant constraints (the deductive closure according to the rules encoded in the ontology in Table 1), as this results in a better connected graph consisting of participants as nodes and binary constraints as edges. The ontology itself is accessed using the Jena ("Jena – A Semantic Web Framework for Java," 2006) framework that includes an OWL reasoner.

The constraint resolution process uses a simplified version of SLD (Robinson, 1965). The unification can be simplified as there are no complex terms in the pattern rule.

Validation

The question arises whether the formal pattern definitions and the tool processing them (the pattern scanner) capture the meaning of the respective pattern. This includes two aspects: firstly, correctness, that is, are all pattern instances detected *intended* instances? Secondly, completeness, that is, are all intended instances found by the scanner? In other terms, are there false positives (correctness) and are there false negatives (completeness)?

The WebOfPatterns client has been validated using test cases based on code that is part of two text books on patterns in Java (Grand, 1998; Stelting & Maassen, 2001). While this validation is successful in the sense that all test cases succeed, it has some obvious weaknesses. Firstly, the code examples used in these text books are (on purpose) very simple, and more complex language features such as inner classes and reflection, are avoided. The current version of the WebOfPatterns source

code analyser does not support inner classes, and therefore certain instances of predicates like actualReturnType and invokedBy are not detected. That is, the fact base built by the client is incomplete. Therefore, certain pattern instances are not found—there are false negatives. But as this is mainly due to limitations of the current implementation of the analysers, the problem should be solved as the client is improved. If behavioural analysis is used, the completeness of the fact base still depends on whether the analyser is able to cover the entire program. This would be the case for programs with unit test cases and 100% test coverage.

An interesting approach to validation would be the comparison of scan results with the results produced by other tools such as CrocoPat (Beyer, Noack, & Lewerentz, 2003) and Fujaba (Niere, Schafer, Wadsack, Wendehals, & Welsh, 2002).

The question whether there are false positives is more difficult to answer and requires the analysis of some scan results. This leads to the introduction of some new concepts related to pattern instantiation.

Normal and Aggregated Instances

The analysis of pattern instances found in non trivial programs reveals a number of "unintended instances." Most of them feature different participants instantiated by one artefact. For instance, consider the following code snippet:

```
1. public abstract class A {
2.     public abstract A copy ( ) ;
3. }
4. public class AImpl extends A {
5.     public A copy ( ) {
6.         A clone = new AImpl ( ) ;
7.         / / configure, set instance variables
8.         return clone;
9.         }
10. }
```

This program instantiates AbstractFactory in a trivial way: A is both the AbstractFactory (role) and the AbstractProduct, while AImpl is the ConcreteFactory and the ConcreteProduct. In this program, a cloneable structure is defined, with A representing the specification of the structure and AImpl being an implementor. The question arises whether there is an implicit constraint in AbstractFactory that requires that the factory and the product must be different. The ontology does contain the respective language feature, the wop:differentFrom property[3]. For some patterns, there is clearly a need for such a constraint. For example, in the Proxy (Gamma et al., 1995) pattern, the Proxy (wrapper) and the RealSubject (the wrapped class) must be different. For other patterns the situation is less clear. In this case we opt for not having such an explicit constraint. This leaves two options open: to use the refinement mechanism to add such an explicit constraint in a sub pattern, or make it a feature of the client to add the wop:differentFrom constraints on demand. Conceptually, we call instances of patterns where different participants are mapped to different artefacts *normal*. In other terms, normal pattern instances are injective.

The number of "good" instances found is surprisingly high. Often it is useful to identify some instances by abstracting from certain details. For instance, two AbstractFactory instances might be considered equal if all abstract participants are mapped to the same artefacts. Two instances of Bridge (Gamma et al., 1995) could be considered variants of the same instance if they have the same abstraction and the same implementor. In general, pattern aggregation is used to identify certain patterns. Instead of considering pattern instances as defined above, classes of instances modulo an equivalence relation ~. This relation meets the following three conditions:

1. **Reflexivity:** $\forall$ instance: instance ~ instance
2. **Symmetry:** $\forall$ $instance_1$,$instance_2$: $instance_1$ ~ $instance_2 \Rightarrow$ $instance_2$ ~ $instance_1$

Table 2. Scan results

Package	Time[d] (s)	Abstract-Factory	Adapter	Bridge	Composite	Proxy	Template-Method
Mandarax 3.4	35	166	91	0	3	2	495
MySQL ConnectorJ 3.1.12	134	56	289	28	0	0	0

3. **Transitivity:** $\forall$ $instance_1$,$instance_2$,$instance_3$: $instance_1$~ $instance_2$ & $instance_2$~ $instance_3$ $\Rightarrow$ $instance_1$~ $instance_3$

Such a relation can be defined as projection with respect to a subset of participants **S** that are part of a pattern as follows:

$instance_1 \sim_S instance_2$ iff $\forall p \in S: instance_1(p) = instance_2(p)$

There are a number of possible aggregations, each defining a separate view on the structure of the program investigated.

1. Identify instances if they map all marked participants to the same artefacts. This allows a default aggregation to be built into the ontology, making this aggregation part of the pattern itself. For this purpose, the ontology has a built-in data type property isAggregationPoint that allows tagging participants explicitly.
2. Identify instances if they map abstract participants to the same artefacts.
3. Identify instances if they map abstract classes to the same artefacts.
4. Identify instances mapping abstract class templates to the same classes, and method templates to methods sharing the same name and the same owner class. This allows the identification of instances containing overloaded methods.
5. No aggregation ("null aggregator").

Aggregation of instances is an alternative to using higher order logic constructs like Eden's higher dimension entities (Eden, 2002b) to describe patterns. By adding default aggregation support to the pattern definition itself but also allowing tools to override it by applying no or alternative aggregations we gain greater flexibility in supporting alternative views on the system design.

Limitations of our Approach and Future Trends

There are some known limitations with the approach presented here. ODOL is not expressive enough to describe constraints referencing instances (and not classes). An example of a pattern that uses this kind of constraint is Prototype (Gamma et al., 1995). The Prototype participant contains a clone() method. The semantics of this method is to provide a copy of the Prototype *object*. In Java style syntax, this can be written as follows: "prototype.clone()!=prototype". This cannot be expressed directly using the current version of ODOL. To address this problem, a data type property isClone is defined in ODOL. The semantics of this method is mainly informal, and the responsibility to interpret it correctly is delegated to applications. Obviously, there is a trade-off here between ontological reductionism (having a small number of modeling primitives with a clear semantics) on the one side, and how simple and convenient it is for software engineers to use the ontology on the other side. In the long term, the solution is to have more expressive vocabularies and to define more derived concepts by means of explicit rules.

There are entire families of patterns for which the ontology presented here is not appropriate.

Table 3. Pattern instance aggregation

Package/ Pattern	No Aggregation	Built-in Aggregation	Group by abstract participants	Group by abstract classes
Mandarax 3.4 / AbstractFactory	166	48	48	29
MySQL ConnectorJ 3.1.12 / AbstractFactory	56	46	46	16
MySQL ConnectorJ 3.1.12 / Bridge	28	1	8	1

Architectural patterns (Buschmann, Meunier, Rohnert, Sommerlad, Stal, Sommerlad, et al., 1996) use concepts such as tier and layer. On the other hand, code level patterns like lazy initialisation refer to concepts used in PL grammars like blocks and error handlers.

A very promising direction for further research is the adoption of the approach presented by Fowler (1999) and Opdyke (1992) for refactoring. A formal, ontology based refactoring language would enable software engineers to find solutions to design problems on the Web, and to perform them in the context of the programming language used. The design problems themselves can be described using an ontology similar to the one presented here as antipatterns. By means of additional derivation rules similarity between patterns can be defined, further increasing the chances of finding a solution to problems encountered.

The relationship between the platform independent ontology layer and the interpretation of these concepts in a particular programming language is an interesting area for further research. We have argued that a client is more than just an implementation. The client has to interpret the concepts found in the ontology correctly. This warrants a closer integration of ontology and client. In particular, for languages like Java where components can be deployed as Web resources,[5] RDF statements can be used to associate object properties and their correct interpretation in a context given by a programming language (Java) and a client application (WOP).

The pattern scanner presented here is far from being mature. Despite the gaps in the AST analyser mentioned, there are other issues related to scalability. The main benefit of having a scalable pattern scanner in the standard toolbox of software engineers (in the form of ANT tasks, or Eclipse plugins) is that this will enable researchers to gather vast amounts of empirical data about software design, and how this design evolves. This may lead to new research on pattern based metrics, and investigations into quantifying the correlation between patterns and software quality.

CONCLUSION

In this chapter, we have shown how techniques developed in the realm of the semantic Web can be used to formalise software Design patterns. We have elaborated an ontology based on the Web ontology language OWL and have shown how this fits into a meta modeling framework that extends the meta modeling architecture used by the OMG. Several aspects related to pattern instantiation discussed are not only relevant to our approach, but should be addressed whenever Design patterns are formalised. In particular, this concerns validation and pattern instance aggregation.

The semantic Web offers tremendous opportunities for the software engineering community. This community is most likely to adapt the semantic Web as it is already accustomed to share much of the knowledge it produces on the Web. It has pioneered the use of Wikis and Blogs, and

has found solutions for how to deal with inconsistencies in open environments. A good example is open source repositories like SourceForge using source control systems such as Subversion or CVS. Participating programmers produce inconsistencies that are resolved with merge and diff tools based on metadata (version tags, commit time, committer). Giving this community better tools to share their knowledge will significantly improve the way software engineering is done. It could eventually lead to an open eco system of patterns (and antipatterns, and refactoring, and ...) where Darwinian forces lead to the survival of the fittest.

Critical for the semantic Web to succeed in software engineering as well as in general is the availability of tools to manage the knowledge produced. Of particular importance is search engine support, for example, to be able to query Google for RDF resources containing instances of the ODOL ontology. Furthermore, modules that allow reasoning about knowledge based on customisable rules and mainly harvested metadata are needed.

Once the community accepts this approach, it is likely that domain specific ontologies will emerge serving the needs of particular communities. These communities are not necessarily organised around programming languages or vendors, but also application areas (desktop, server, embedded), industries (financial, e-commerce) or programming paradigms (Aspect-Oriented Programming). We will have to deal with a multidimensional space of ontologies. This poses new challenges for describing the relationships between these ontologies.

REFERENCES

Beck, K. (1999). *Extreme programming explained: Embrace change.* Addison-Wesley Professional.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). The Semantic Web. *Scientific American*, 34-43.

Beyer, D., Noack, A., & Lewerentz, C. (2003). Simple and efficient relational querying of software structures. In *Paper presented at the 10th Working Conference on Reverse Engineering (WCRE 2003).*

Boley, H., Tabet, S., & Wagner, G. (2001). Design rationale of RuleML: A markup language for Semantic Web rules. *Paper presented at the International Semantic Web Working Symposium (SWWS).*

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., et al. (1996). *Pattern-oriented software architecture, vol. 1: A system of patterns* (1st ed.). John Wiley & Sons.

Dietrich, J., & Elgar, C. (2005a). *A formal description of Design patterns using OWL*. Paper presented at the Australian Software Engineering Conference (ASWEC), 2005.

Dietrich, J., & Elgar, C. (2005b). *Towards a Web of patterns.* Paper presented at the Workshop on Semantic Web Enabled Software Engineering (SWESE), Galway.

The Dublin Core Metadata Initiative. (2006). Retrieved November 16, 2006, from http://dublincore.org/

Eden, A. (2002a). LePUS, A visual formalism for object-oriented architectures. *Paper presented at the 6*th *World Conference on Integrated Design and Process Technology*, Pasadena, California.

Eden, A. (2002b). A theory of object-oriented design. *Information Systems Frontiers, 4*(4), 379-391.

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Reading, MA: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*: Addison-Wesley.

Grand, M. (1998). *Patterns in Java: A catalog of reusable Design patterns illustrated with UML*. Wiley.

Gruber, T.R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition, 5*(2), 199-220.

Haarslev, V., & Moeller, R. (2001). Description of the RACER system and its applications. *Paper presented at the 2001 International Workshop on Description Logics (DL2001),* Stanford.

Horrocks, I. (1999). *FaCT and iFaCT.* Paper presented at the 1999 International Workshop on Description Logics (DL'99).

The JavaBeans Spec. (2006). Retrieved November 16, 2006, from http://java.sun.com/products/javabeans/docs/spec.html

Jena – A Semantic Web Framework for Java. (2006). Retrieved November 16, 2006, from http://jena.sourceforge.net/

Klyne, G., & Carroll, J.J. (2004). *Resource description framework (RDF): Concepts and abstract syntax*. W3C recommendation. Retrieved November 16. 2006, from http://www.w3.org/TR/rdf-concepts/

Knublauch, H., Musen, M., & Rector, A. (2004). Editing description logics ontologies with the Protégé OWL plugin. *Paper presented at the International Workshop on Description Logics,* Whistler, British Columbia.

Kraemer, C., & Prechelt, L. (1996). Design recovery by automated search for structural Design patterns in object-oriented software. *Paper presented at the 3rd Working Conference on Reverse Engineering (WCRE ,96).*

McGuinness, D.L., & Harmelen, F.V. (2004). *OWL Web ontology language overview.* W3C recommendation. Retrieved November 16,2006, from *http://www.w3.org/TR/owl-features/*

MOF QVT Final Adopted Specification. (2005). Retrieved November 16, 2006, from http://www.omg.org/docs/ptc/05-11-01.pdf

Niere, J., Schafer, W., Wadsack, J.P., Wendehals, L., & Welsh, J. (2002). Towards pattern-based design recovery. *Paper presented at the 24th International Conference on Software Engineering (ICSE 2002).*

The ODOL OWL Ontology. (2006). Retrieved November 16, 2006, from http://www-ist.massey.ac.nz/wop/20060324/wop.owl

Opdyke, W.F. (1992). *Refactoring object-oriented frameworks.* University of Illinois at Urbana-Champaign, IL.

OWLDoc. (2006). Retrieved November 16, 2006, from http://co-ode.man.ac.uk/downloads/owldoc/co-ode-index.php

Patel-Schneider, P.F., Hayes, P., & Horrocks, I. (2004). *OWL Web ontology language semantics and abstract syntax, W3C recommendation*. Retrieved November 16, 2006, from http://www.w3.org/TR/owl-semantics/

Robinson, J.A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12*(1), 23-41.

Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., & Katz, Y. (2006). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics (To Appear).*

Stelting, S., & Maassen, O. (2001). *Applied Java patterns.* Prentice Hall PTR.

Tichelaar, S., Ducasse, S., Demeyer, S., & Nierstrasz, O. (2000). *A meta-model for language-independent refactoring.* Paper presented at the International Symposium on Principles of Software Evolution.

Uschold, M. (2003). Where are the semantics in the semantic Web? *AI Magazine, 24*(3), 25-36.

The WebOfPatterns Project. (2006). Retrieved November 16, 2006, from http://www-ist.massey.ac.nz/wop/

ENDNOTES

1 Name space prefixes used in this chapter are defined as follows:
xmlns:wop="http://www.massey.ac.nz/iist/cs/pattern/ontology#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/"

2 In OWL, properties are called *data type properties* and relationships/predicates associating resources are called *object properties.*

3 This property is different from owl:differentFrom that states that two individuals are different. The semantics of wop:differentFrom is that the respective participants have to be instantiated by different artefacts.

4 These results have been obtained using the following configuration: Pentium 4 CPU 3.20 GHz, 1GB RAM, Windows XP Pro, Java 1.5.0_06, Eclipse 3.1.0

5 Jar files can be deployed on Web servers and accessed using a special URL class loader.

This work was previously published in Design Pattern Formalization Techniques, edited by T. Taibi, pp. 258-279, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.3
Class Patterns and Templates in Software Design

Julio Sanchez
Minnesota State University, Mankato, USA

Maria P. Canton
South Central College, USA

ABSTRACT

This chapter describes the use of design patterns as reusable components in program design. The discussion includes the two core elements: the class diagram and examples implemented in code. The authors believe that although precanned patterns have been popular in the literature, it is the patterns that we personally create or adapt that are most useful. Only after gaining intimate familiarity with a particular class structure will we be able to use it in an application. In addition to the conventional treatment of class patterns, the discussion includes the notion of a class template. A template describes functionality and object relations within a single class, while patterns refer to structures of communicating and interacting classes. The class template fosters reusability by providing a guide in solving a specific implementation problem. The chapter includes several class templates that could be useful to the software developer.

DESIGN PATTERNS

Engineers and architects have reused design elements for many years (Alexander, Ishikawa, Silverstein, Jacobson, Fiksdahl-King, & Angel, 1977); however, the notion of reusing elements of software design dates back only to the early 1990s. The work of Anderson (1990), Coplien (1992), and Beck and Johnson (1994) set the background for the book *Design Patterns* by Gamma, Helm, Johnson, and Vlissides (1995), which many considered the first comprehensive work on the subject.

The main justification for reusing program design components is based on the fact that the design stage is one of the most laborious and time-consuming phases of program development. Design reuse is founded in the assumption that once a programmer or programming group has found a class or object structure that solves a particular design problem, this pattern can then be reused in other projects, with considerable savings in the design effort. Anyone who has participated in the development of a substantial

software project appreciates the advantages of reusing program design components.

The present-day approach to design reuse is based on a model of class associations and relationships called a class pattern or an object model. In this sense, a pattern is a solution to a design problem. Therefore, a programming problem is at the origin of every pattern. From this assumption we deduce that a pattern must offer a viable solution; it must represent a class structure that can be readily coded in the language of choice.

The fact that a programming problem is at the root of every design pattern, and the assumption that the solution offered by a particular pattern must be readily implementable in code, are the premises on which we base our approach to this topic. In the context of this chapter we see a design pattern as consisting of two core elements: a class diagram and a coded example or template, fully implemented in code. Every working programmer knows how to take a piece of existing code and reengineer it to solve the problem at hand. However, snippets of code that may or may not compile correctly are more a tease than a real aide.

Although we consider that design patterns are a reasonable and practical methodology, we must also add that it is the patterns that we ourselves create, refine, or adapt that are the most useful. It is difficult to believe that we can design and code a program based on someone else's class diagrams. Program design and coding is a task too elaborate and complicated to be done by imitation or by proxy. A programmer must gain intimate familiarity with a particular class and object structure before committing to its adoption in a project. These thoughts lead to the conclusion that it is more important to explain how we can develop our own design patterns than to offer an extensive catalog of someone's class diagrams, which can be difficult to understand, and even more difficult to apply.

CLASS TEMPLATES

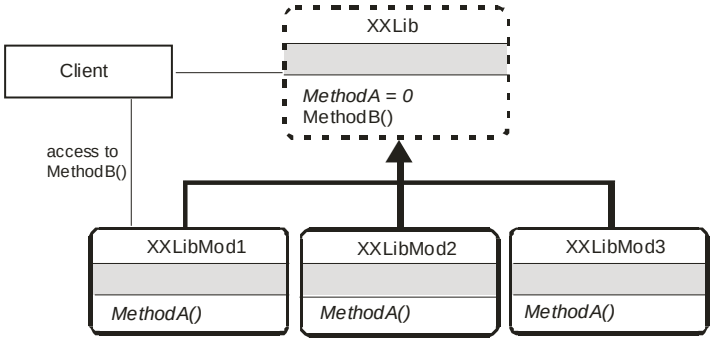
Occasionally, a programmer or program designer's need is not for a structure of communicating and interacting classes but for a description of the implementation of a specific functionality within a single class. In this case we can speak of a class template rather than of a pattern. The purpose of a class template is also to foster reusability by providing a specific guide for solving a particular implementation problem. In the following sections we include several class templates that could be useful to the practicing developer.

A Pattern is Born

We begin our discussion by following through the development of a design pattern, from the original problem, through a possible solution, to its implementation in code, and concluding in a general-purpose class diagram.

One of the most obvious and frequent uses of dynamic polymorphism is in the implementation of class libraries. The simplest usable architecture is by means of an abstract class and several modules in the form of derived classes that provide the specific implementations of the library's functionality. Client code accesses a polymorphic method in the base class and the corresponding implementation is selected according to the object referenced. But in the real world a library usually consists of more than one method. Since many languages allow mixing virtual and nonvirtual functions in an abstract class, it is possible to include nonvirtual methods along with virtual and pure virtual ones. The problem in this case is that abstract classes cannot be instantiated; therefore, client code cannot create an object through which it can access the nonvirtual methods in the base class. A possible but not very effective solution is to use one of the derived classes to access the nonvirtual methods in the base class. Figure 1 depicts this situation.

Figure 1. A class library implemented through dynamic polymorphism



The first problem of the class diagram in Figure 1 is that the client code accesses the nonvirtual MethodB() in the base class by means of a pointer to one of the derived classes. A second, and perhaps more important one, is that method selection must take place in the client's code. Both of these characteristics expose the class structure and add a substantial processing burden to the client.

There are several possible solutions to the first problem. We could make the base class a concrete class with MethodA() as a simple virtual function, with no real implementation. In this case MethodB() becomes immediately accessible. Another solution would be to create a new class to hold the nonvirtual methods originally in the base class and have this new class inherit abstractly. However, neither of these solutions addresses the most important problem, which is that client code must perform method selection. It is this characteristic of inheritance that breaks encapsulation.

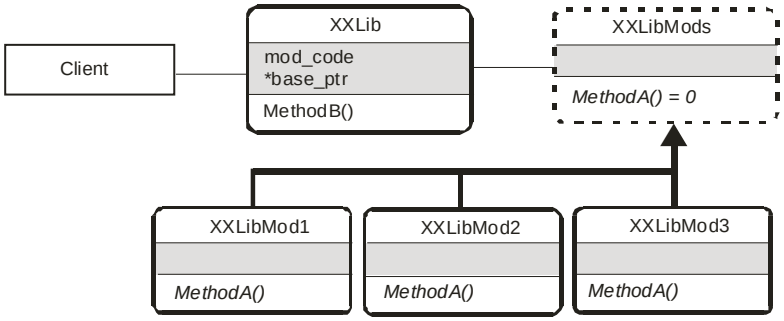
Encapsulation can be preserved by using object composition instead of inheritance. Also, combining object composition and inheritance achieves dynamic binding of polymorphic methods while preserving encapsulation. Figure 2 is a possible class diagram for implementing the library by means of object composition and inheritance.

Design of a VESA True Color Library

The design of a VESA true color graphics library can be based on combining object composition and class inheritance. The minimal functionality of this library can be stated as follows:

1. A way of setting the desired VESA true color mode.

Figure 2. An alternative implementation of the class library



2. A way of obtaining the current VESA mode as well as the vertical and horizontal resolution.
3. A way of drawing a screen pixel defined in terms of its x and y screen coordinates and its color attribute.
4. A way of reading the color attribute of a screen pixel located at given screen coordinates.

Figure 3 is a diagram of the library classes.

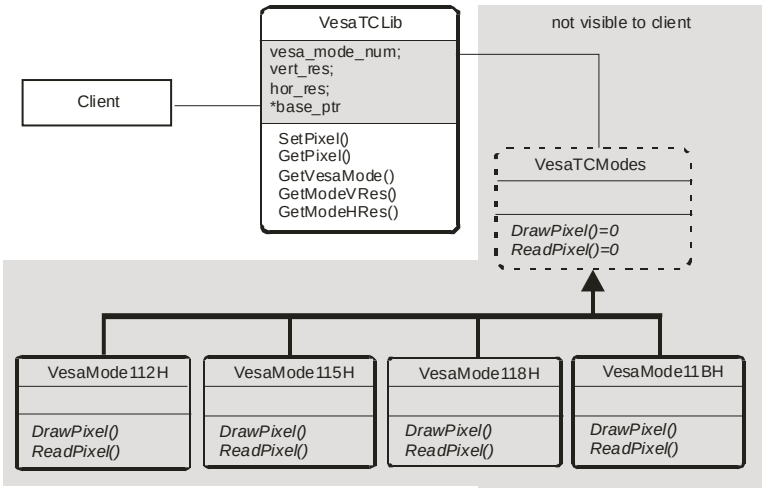
Observing the class diagram in Figure 3 we detect some features of the class structure:

1. The nonvirtual methods are located in the class named VesaTCLib.
2. VesaTCLib is a concrete class and can be instantiated by the client.
3. The mode-dependent, polymorphic methods for setting and reading pixels, named DrawPixel() and ReadPixel(), respectively, are located in an inheritance structure.
4. The methods named GetPixel() and SetPixel() in the class VesaTCLib provide access by means of a pointer to the polymorphic methods DrawPixel() and ReadPixel().

The result of this class design is that the implementation is now transparent to the client, as shown by the gray shaded background in Figure 3. The following program (Box 1) contains the schematic implementation of the class and inheritance structure in Figure 3.

The previous code sample, named SAMP-01.CPP, deserves a few additional comments. The constructor receives a coded signature that defines the VESA true color mode requested by the caller. A switch construct stores the VESA mode number, vertical and horizontal resolution, and the address of the library methods used in setting and reading a pixel in this mode. Therefore, the contingency code executes only once, when the object is created. Thereafter, each object of the class VesaTCLib is associated with a particular mode and keeps the address of the method to be used in its own pixel setting and reading operations. The methods GetPixel() and ReadPixel() of the class VesaTCLib are the client's interface to the pixel-level primitives in the library. The actual setting and reading of a screen pixel is performed as follows:

Figure 3. Class diagram for a VESA true color graphics library



Box 1.

```
//*************************************************************
// C++ program to illustrate implementation of a VESA true
// color graphics library using object composition and class
// inheritance
// Filename: SAMP-01.CPP
//*************************************************************

#include <iostream.h>

//*************************************************************
//             classes
//*************************************************************
// Abstract base class
class VesaTCModes {
public:
// Pure virtual functions
 virtual void DrawPixel(int, int, int) = 0;
 virtual unsigned long ReadPixel(int, int) = 0;
};
//*****************************
//  Polymorphic classes
//*****************************
// Note: methods have stub implementations in this demo
//   program

class VesaMode112H : public VesaTCModes {
public:
 virtual void DrawPixel(int row, int column, int color) {
 cout << "Setting pixel in Mode 112H\n";
 return;
 }
 virtual unsigned long ReadPixel(int row, int column) {
 cout << "Reading pixel in Mode 112H\n" ;
 return 0;
 }
};

class VesaMode115H : public VesaTCModes {
public:
 virtual void DrawPixel(int row, int column, int color) {
 cout << "Setting pixel in Mode 115H\n";
```

continued on following page

Box 1. continued

```
 return;
 }
 virtual unsigned long ReadPixel(int row, int column) {
 cout << "Reading pixel in Mode 115H\n" ;
 return 0;
 }
};
class VesaMode118H : public VesaTCModes {
public:
 virtual void DrawPixel(int row, int column, int color) {
 cout << "Setting pixel in Mode 118H\n";
 return;
 }
 virtual unsigned long ReadPixel(int row, int column) {
 cout << "Reading pixel in Mode 118H\n";
 return 0;
 }
};

class VesaMode11BH : public VesaTCModes {
public:
 virtual void DrawPixel(int row, int column, int color) {
 cout << "Setting pixel in Mode 11BH\n";
 return;
 }
 virtual unsigned long ReadPixel(int row, int column) {
 cout << "Reading pixel in Mode 11BH\n";
 return 0;
 }
};
//************************
// non-virtual classes
//************************
class VesaTCLib {
private:
 int vesa_mode_num;     // Object data
 int vert_res;
 int hor_res;

 VesaMode112H obj_112H;   // Objects of derived classes
 VesaMode115H obj_115H;   // required for filling pointer
 VesaMode118H obj_118H;
```

continued on following page

Box 1. continued

```
 VesaMode11BH obj_11BH;

 VesaTCModes *base_ptr;  // Base class pointer
public:
 VesaTCLib (int);      // Constructor
// Other methods
 int GetVesaMode();
 int GetModeVRes();
 int GetModeHRes();
 void SetPixel(int, int, int);
 void GetPixel(int, int);
};
//*****************************
// Methods for class VesaTCLib
//*****************************
// Constructor
VesaTCLib::VesaTCLib(int vesa_mode) {
/* The constructor is passed a mode code as follows:
  1 = VESA mode 112H
  2 = VESA mode 115H
  3 = VESA mode 118H
  4 = VESA mode 11BH
According to the mode selected, code sets the definition,
VESA mode number, and a pointer to the corresponding
object of the library module.
*/
 switch (vesa_mode) {
 case (1):
  vesa_mode_num = 0x112;
  hor_res = 640;
  vert_res = 480;
  base_ptr = &obj_112H;
  break;
 case (2):
  vesa_mode_num = 0x115;
  hor_res = 800;
  vert_res = 600;
  base_ptr = &obj_115H;
  break;
 case (3):
  vesa_mode_num = 0x118;
  hor_res = 1024;
```

continued on following page

Box 1. continued

```
    vert_res = 768;
    base_ptr = &obj_118H;
    break;
   case (4):
    vesa_mode_num = 0x11b;
    hor_res = 1280;
    vert_res = 1024;
    base_ptr = &obj_11BH;
    break;
   default:
    vesa_mode_num = 0x0;
    hor_res = 0;
    vert_res = 0;
    base_ptr = &obj_112H;
   }
}
// Methods for reading and setting a screen pixel
void VesaTCLib::SetPixel(int row, int col, int attribute) {
 base_ptr->DrawPixel(row, col, attribute);
};
void VesaTCLib::GetPixel(int row, int col) {
 base_ptr->ReadPixel(row, col);
};

// Methods that return the mode information
int VesaTCLib::GetVesaMode() {
 return vesa_mode_num;
}

int VesaTCLib::GetModeVRes() {
 return vert_res;
}

int VesaTCLib::GetModeHRes() {
 return hor_res;
}

//************************************************************
//            client code
//************************************************************

main() {
```

continued on following page

Box 1. continued

```
// Objects of class VesaTCLib
 VesaTCLib obj_1(1);    // Object and mode code
 VesaTCLib obj_2(2);
 VesaTCLib obj_3(3);
 VesaTCLib obj_4(4);

// Operations on obj_1, mode code 1
 cout << "\nVESA mode: " << hex << obj_1.GetVesaMode();
 cout << "\nHorizontal Res: " << dec << obj_1.GetModeHRes();
 cout << "\nVertical Res: " << obj_1.GetModeVRes() << "\n";
 obj_1.SetPixel(12, 18, 0xff00);
 obj_1.GetPixel(122, 133);
 cout << "\n";

// Operations on obj_2, mode code 2
 cout << "VESA mode: " << hex << obj_2.GetVesaMode();
 cout << "\nHorizontal Res: " << dec << obj_2.GetModeHRes();
 cout << "\nVertical Res: " << obj_2.GetModeVRes() << "\n";
 obj_2.SetPixel(12, 18, 0xff00);
 obj_2.GetPixel(122, 133);
 cout << "\n";

// Operations on obj_3, mode code 3
 cout << "VESA mode: " << hex << obj_3.GetVesaMode();
 cout << "\nHorizontal Res: " << dec << obj_3.GetModeHRes();
 cout << "\nVertical Res: " << obj_3.GetModeVRes() << "\n";
 obj_3.SetPixel(12, 18, 0xff00);
 obj_3.GetPixel(122, 133);
 cout << "\n";

// Operations on obj_4, mode code 4
 cout << "VESA mode: " << hex << obj_4.GetVesaMode();
 cout << "\nHorizontal Res: " << dec << obj_4.GetModeHRes();
 cout << "\nVertical Res: " << obj_4.GetModeVRes() << "\n";
 obj_4.SetPixel(12, 18, 0xff00);
 obj_4.GetPixel(122, 133);
 cout << "\n";

 return 0;
}
```

continued on following page

```
void VesaTCLib::SetPixel(int row, int col,
int atts) {
  base _ ptr->DrawPixel(row,  col,  attri-
bute);
};

void  VesaTCLib::GetPixel(int  row,  int
col) {
 base _ ptr->ReadPixel(row, col);
};
```

The processing in this case is done with a smaller processing overhead. The principal advantage of the new class design and implementation can be summarized as follows:

1. Contingency code to select the corresponding mode-dependant, pixel-level primitives is now located in the constructor; therefore, it executes only when the object is created.
2. Client code does not need to perform the mode selection operations, which have been transferred to the library classes.
3. Client code does not see or access the class inheritance structure since the pixel-level operations are handled transparently.

Developing the Pattern

In the previous sections we addressed a programming problem and found one possible solution that could be implemented in code. We also constructed a class diagram that reflects the relationships and associations of this solution in object-oriented terms. In order to make this solution easier to reuse we can eliminate all the case-specific elements from both the pattern and the coded example. Furthermore, the resulting abstraction can be given a name that provides an easy reference to the particular case.

In selecting a name for a design pattern we must carefully consider its purpose and applicability. Observe that the class structure for constructing the VESA true color library is probably applicable to many programming problems that are not related to computer graphics, or even to libraries. Its fundamental purpose is to provide an interface to an inheritance structure so that its operational details are hidden from client code. Since interface is too general a term, we could think of the word concealer in this context. For the lack of a better term we call this pattern a concealer, since its purpose is to conceal an inheritance structure so that its existence and operation are made transparent to the client. Figure 4 shows the concealer pattern in a more abstract form.

Unifying Dissimilar Interfaces

A simple but frequent design problem is to present a unified and friendly interface to a set of classes that perform different operations, for example, a set of classes that calculates the area of different geometrical figures, such as a parallelogram, a circle, a rectangle, and a square. The formula for the area of a parallelogram requires three parameters: the base, the side, and the included angle. The area of the circle and the square require a single parameter, in one case the radius and in the other one the side. The area of a rectangle requires two: the length and the width. Our task is to provide the client with a single interface to the calculation of the area of any one of these four geometrical figures. This class structure is known as a unifier pattern.

In the previous example the implementation could be based on the client passing four parameters. The first one is a signature code indicating the geometrical figure, the other three parameters hold the pertinent dimensional information. By convention, we agree that unnecessary parameters are set to NULL. The class diagram in Figure 5 represents one possible solution.

In the case of Figure 5 the method selection is based on an object of the corresponding class, therefore, it is a case of object composition. An alternative implementation could easily be based on pointers instead of object instances. Program

Figure 4. A Concealer pattern

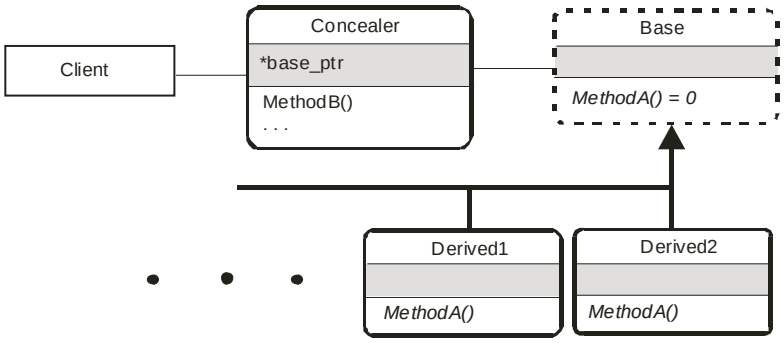
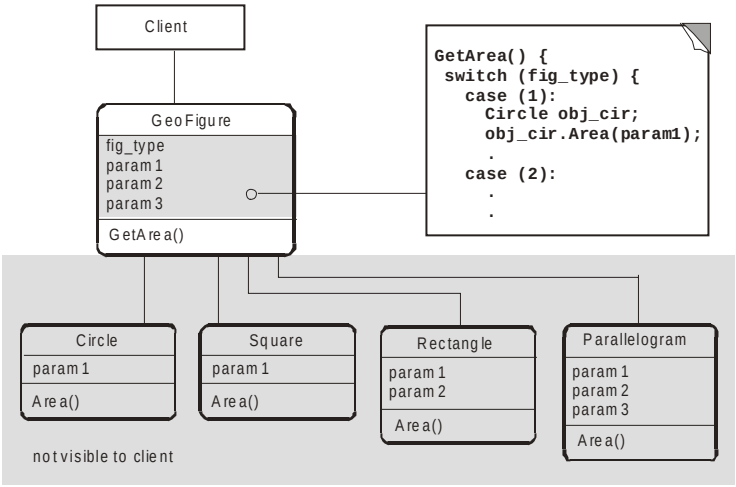


Figure 5. Implementing an interface class



SAMP-02.CPP shows the necessary processing in the first case (Box 2).

Note that in program SAMP-02.CPP the objects are instantiated inside the switch construct in the GetArea() method. This ensures that only the necessary object is created in each iteration of GetArea(). Since the objects have local scope their lifetime expires when GetArea() returns. Therefore, only the necessary memory is used.

Unifier patterns have found frequent use in modeling object-oriented frameworks. InterViews 2.6 defined a similar construct for modeling interface elements such as buttons, scrollbars, and menus (Vlissides, 1988).

An Interface Pattern

By generalizing the class diagram in Figure 5 we can develop an interface pattern in which a class provides access to other classes that implement certain functionality. Figure 6 shows this generalization.

Aggregated Class Hierarchies

On occasions we may need to implement a set of classes related hierarchically. For example, a program can draw geometrical figures, such as circles, squares, and triangles, where each of the figures is contained in an invisible, rectangular

Box 2.

```
//************************************************************
// C++ program to illustrate implementation of an interface
// class
// Filename: SAMP-02.CPP
//************************************************************

#include <iostream.h>
#include <math.h>
#define PI 3.1415

//***************************
//    classes
//***************************
//
class Circle {
public:
 float Area(float radius) {
  return (radius * radius * PI);
 }
};

class Rectangle {
public:
 float Area(float height, float width) {
  return (height * width);
 }
};

class Parallelogram {
public:
  float Area(float base, float side, float angle) {
   return (base * side * sin (angle) );
  }
};

class Square {
public:
  float Area(float side) {
   return (side * side);
  }
};
```

continued on following page

Box 2. continued

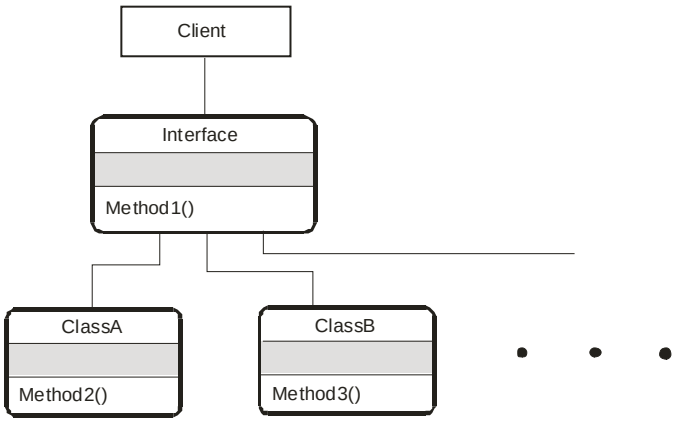
```
// Interface class
class GeoFigure {
private:
 int fig _ type;
 float param1;
 float param2;
 float param3;
public:
 GeoFigure(int, float, float, float);
 void GetArea();
};
// Constructor
GeoFigure::GeoFigure(int type, float data1, float data2,
      float data3) {
 param1 = data1;
 param2 = data2;
 param3 = data3;
 fig _ type = type;
};

// Implementation of GetArea() method
void GeoFigure::GetArea() {
float area;
 switch (fig _ type) {
  case (1):        // Circle
   Circle obj _ cir;
   area = obj _ cir.Area(param1);
   break;
  case (2):        // Rectangle
   Rectangle obj _ rec;
   area = obj _ rec.Area(param1, param2);
   break;
  case (3):        // Parallelogram
   Parallelogram obj _ par;
   area = obj _ par.Area(param1, param2, param3);
   break;
  case (4):        // Square
   Square obj _ sqr;
   area = obj _ sqr.Area(param1);
   break;
 }
```

continued on following page

Box 2. continued

```
   cout << "The area is of this object is: "
  << area << "\n";
};

//***************************
//     main()
//***************************
main() {
 GeoFigure obj1(1, 3, NULL, NULL);  // A circle object
 GeoFigure obj2(2, 12, 4, NULL);   // A rectangle object
 GeoFigure obj3(3, 12, 4, 0.7);    // A parallelogram object
 GeoFigure obj4(4, 3, NULL, NULL);  // A square object

 cout << "\nCalculating areas of objects...\n";
// Calculate areas
 obj1.GetArea();    // Area of circle object
 obj2.GetArea();    // Area of rectangle object
 obj3.GetArea();    // Area of parallelogram object
 obj4.GetArea();    // Area of square

 return 0;
}
```

Figure 6. An interface pattern



frame. Also assume that in this case the figure and its containing frame are so related that the creating of a figure (i.e., a circle, rectangle, or triangle) mandates the creation of its container frame.

A library of figure-drawing primitives could be provided by means of four classes: three to create the circle, square, and triangle figures, and one to generate the containing frame. The client would create an object of the corresponding figure and then one of the frame. Alternatively, figure-drawing classes could be linked to the frame generation class so that for each figure a corresponding frame object would be automatically created. With this approach, the frame generation operation is transparent to the client and programming is simplified.

We can add complications to the preceding example without altering its validity. For instance, we can assume that each frame implies the creation of another element called a border, and that the border must be contained in still another one called a window. Therefore, the resulting hierarchy is a geometrical figure, contained in a frame, requiring a border, which, in turn, must exist inside a window. In this example the object structure is mandated by the problem description since all the classes in the hierarchy are obligatory.

A class hierarchy can be implemented in C++ by inheritance, because the creation of an object of a derived class forces the creation of one of its parent class. If the inheritance hierarchy consists of more than one class, then the constructor of the classes higher in the hierarchy are called in order of derivation. Destructors are called in reverse order. The program SAMP-03.CPP demonstrates constructors and destructors in an inheritance hierarchy (Box 3).

When program SAM12-03.CPP executes the following messages are displayed:

```
Constructing a BaseA object
Constructing a Derived1 object
Constructing a Derived2 object
Destroying a Derived2 object
Destroying a Derived1 object
Destroying a BaseA object
```

A Class Hierarchy by Object Composition

One way of solving the problem described at the beginning of this section is to use inheritance. For example, we could implement a class hierarchy in which Circle, Square, and Rectangle were derived from a base class called Figure. The Figure class could contain polymorphic methods for calculating and drawing the geometrical object, as well as methods for creating the frame, the border, and the window. In this case the client would have the responsibility of calling all of the required methods. This is an example of how inheritance often constraints programming and exposes the underlying class structure.

Although in some cases a solution based on class inheritance may be acceptable, it often happens that the hierarchy of super classes is related to a particular method of a derived class, or to several related methods, but not to all of them (Gamma et al., 1995). Consider the case of several types of geometrical figures, all of which must be part of a window, contain a border, and be enclosed in a frame as described previously. In this case the figure-drawing methods could be made part of an inheritance structure; however, the methods that produce the window, border, and frame need not be part of the inheritance construct since these elements are required for any of the geometrical figures. One possible solution is to implement a class structure in which some methods form part of an inheritance association while others are implemented by means of object composition. Figure 7 shows a possible class diagram for this case.

In Figure 7 there are two mechanisms collaborating within the class structure. An inheritance element provides polymorphic methods that can be selected at run time in the conventional manner. Simultaneously, another class hierarchy is

Box 3.

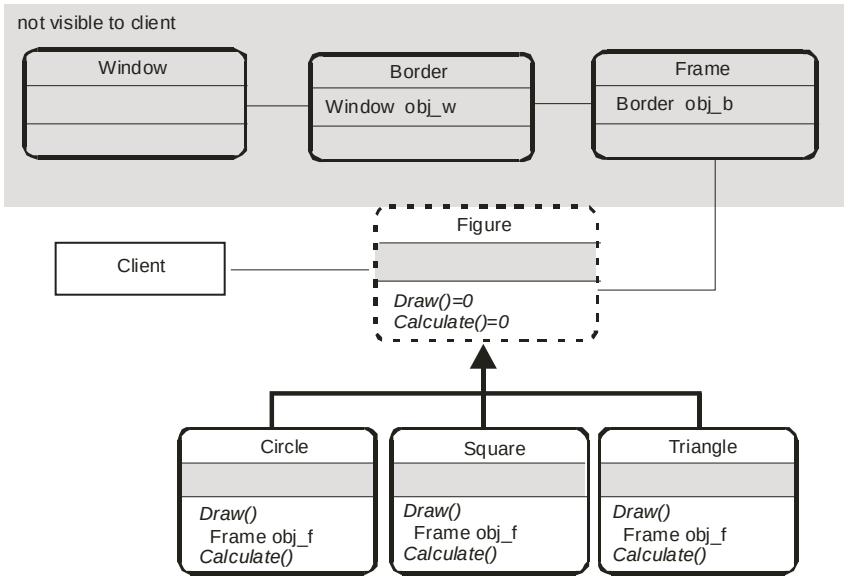
```
//************************************************************
// C++ program to illustrate constructors and destructors in
// a class inheritance hierarchy
// Filename: SAMP-03.CPP
//************************************************************

#include <iostream.h>

//***************************
//    classes
//***************************
class BaseA {
public:
 BaseA() {
  cout << “Constructing a BaseA object\n”;
 }
 ~BaseA() {
  cout << “Destroying a BaseA object\n”;
 }
};

class Derived1 : public BaseA {
public:
 Derived1() {
 cout << “Constructing a Derived1 object\n”;
 }
 ~Derived1() {
 cout << “Destroying a Derived1 object\n”;
 }

};
class Derived2 : public Derived1 {
public:
 Derived2() {
 cout << “Constructing a Derived2 object\n”;
 }
 ~Derived2() {
 cout << “Destroying a Derived2 object\n”;
 }
};
```

continued on following page

Box 3. continued

```
//*****************************
//     main()
//*****************************
main() {

// Program creates a single object of the lower class in the
// hierarchy. Constructors and destructors of the class higher
// in the hierarchy are automatically executed.

  Derived2 obj_d;
  return 0;
}
```

Figure 7. Class hierarchy by object composition



based on object composition. Note that the method Draw() of the classes Circle, Square, and Triangle, contains an object of the class Frame. Also note that the class Frame contains an object of Border, which contains an object of Window. Therefore, we can say that a circle, a square, and a triangle are a kind of figure and that all of them have a frame, a border, and a window. The program SAMP-04.CPP is an implementation of the class diagram in Figure 7 (Box 4).

Note in the program SAMP-04.CPP, as well as in Figure 7, that it is the method named Draw() in the concrete classes of the inheritance structure that instantiates the object of the class higher in the hierarchy, in this case the class named Frame. Once this object is referenced, the remainder of the hierarchy is produced automatically by means of the member object mechanism. The purpose of this construct is that the object hierarchy is generated when the method named Draw() ex-

ecutes, not when an object of the lower classes is instantiated. We can certify this operation when the program executes.

There may be cases in which we prefer that the hierarchy of super classes be instantiated at the time that the object of the lower class is created. In the example in Figure 7 this could be achieved by having a member object of the Frame class in the base class named Figure.

A Chain Reaction Pattern

In the class diagram of Figure 7 we note that when the lower classes (Circle, Square, and Triangle) instantiate an object of the class higher in the hierarchy, they start a chain reaction that produces objects of the entire hierarchy. We can abstract this operation by means of a class diagram that focuses exclusively on the chain reaction element, as is the case in Figure 8.

In Figure 8 we have implemented chaining by referencing the first chained object within a method of the Chainer class, and then by member objects of the chained classes. There are many other ways of implementing a chain reaction effect.

Box 4.

```
//*************************************************************
// C++ program to illustrate a class hierarchy by object
// composition
// Filename: SAMP-04.CPP
//*************************************************************

#include <iostream.h>

//***************************
//    classes
//***************************
class Window {
public:
Window() {
 cout << "Creating a window\n";
 }
};

class Border {
private:
 Window obj_w;     // Border class contains Window object
public:
 Border() {
  cout << "Drawing a border\n";
```

continued on following page

Box 4. continued

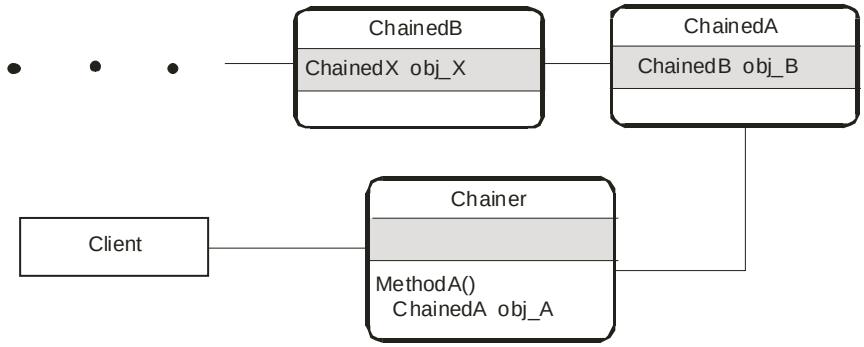
```
 }
};

class Frame {
private:
 Border obj_b;     // Frame class contains Border object
public:
 Frame() {
  cout << "Drawing a frame\n";
 }
 virtual void Draw() { return; };
};

// Abstract class
class Figure {
public:
 virtual void Draw() = 0;
 virtual void Calculate() = 0;
};

// Circle, Triangle and Square are at the bottom of the class
// hierarchy
class Circle : public Figure {
public:
 virtual void Draw() {
 Frame obj_f;
 cout << "Drawing a circle\n";
 }
 virtual void Calculate() {
 cout << "Calculating a circle\n";
 }
};

class Square : public Figure {
public:
 virtual void Draw() {
 Frame obj_f;
 cout << "Drawing a square\n";
 }
 virtual void Calculate() {
 cout << "Calculating a square\n";
 }
```

continued on following page

Box 4. continued

```
};

class Triangle : public Figure {
public:
 virtual void Draw() {
 Frame obj_f;
 cout << "Drawing a triangle\n";
 }
 virtual void Calculate() {
 cout << "Calculating a triangle\n";
 }
};

//***************************
//     main()
//***************************
main() {


  Figure *base_ptr;   // Pointer to base class
  Circle  obj_c;   // Circle, Square, and Triangle
  Square  obj_s;   // objects
  Triangle obj_t;

  cout << "\n\n";
  base_ptr = &obj_c;  // Draw a circle and its hierarchical
  base_ptr->Draw();   // super classes

  cout << "\n";
  base_ptr = &obj_s;  // Draw a square and its hierarchical
  base_ptr->Draw();   // super classes

  cout << "\n";
  base_ptr = &obj_t;  // Draw a triangle and its hierarchical
  base_ptr->Draw();   // super classes

  cout << "\n";
  base_ptr->Calculate(); // Calculate() method does not generate
            // an object hierarchy
  return 0;
}
```

Figure 8. A chain reaction pattern



Object Chaining

A programming problem often encountered consists of determining which, if any, among a possible set of actions has taken place. For example, an error handling routine posts the corresponding messages and directs execution according to the value of an error code. A common way of performing the method selection is by contingency code that examines the error code and determines the corresponding action. In C++ this type of selection is usually based on a switch construct or on a series of nested if statements. Alternatively, we can use object composition to create a chain in which each object examines a code operand passed to it. If it corresponds to the one mapped to its own action, then the object performs the corresponding operation, if not, it passes along the request to the next object in a chain. The last object in the chain returns a special value if no valid handler is found.

One of the advantages of using an object chain is that it can be expanded simply by inserting new object handlers anywhere along its members. To expand a selection mechanism based on contingency code we usually have to modify the selecting method by recoding the switch or nested if statements.

An Object Chain Example

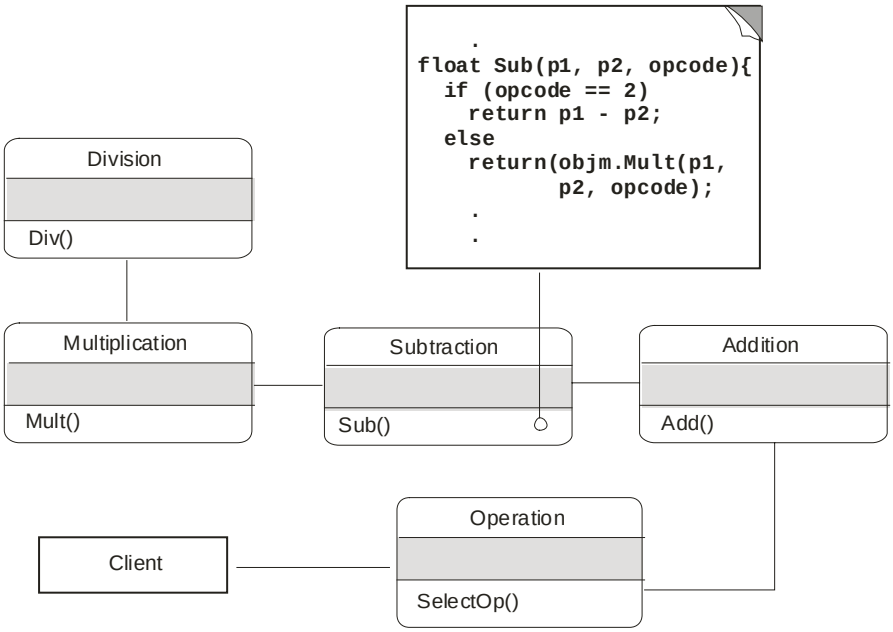
A slightly more complicated case is one in which the selected object must return a value back to the original caller. For example, we define a series of classes that perform arithmetic operations which take two operands. The classes are called Addition, Subtraction, Multiplication, and Division. An operation code is furnished as a third parameter to a class called Operation, containing a method called SelectOp() that calls the Add method in the first object in the chain. In this case the object is of the Addition class. Add() examines the opcode operand; if it corresponds to the add operation, add executes and returns the sum to the caller. If not, it passes the object to Subtract, which proceeds in a similar fashion. Figure 9 shows the class structure for this example.

The program SAMP-05.CPP shows the processing details for implementing an object chain (Box 5).

An Object Chain Pattern

We can generalize the example of an object chain in Figure 9 and abstract its basic components. In this case the fundamental characteristic of the class structure is a series of related methods, each one of which inspects a program condition that determines whether the method is to respond with a processing action or pass the request along to

Figure 9. Class diagram for an object chain



the next object in the chain. Figure 10 is a class diagram for an object chain pattern.

STRING HANDLING CLASS TEMPLATE

The solution of some program development problems is based on patterns of interacting classes and objects, while others simply require a description of the internal structure of a single class. In this case we speak of a class template. For example, programs that deal with strings or that perform substantial string manipulations could profit from a particular class design that is optimized for string handling.

String Operations

A string that is defined at run time is sometimes difficult to store as an array since its length may not be known beforehand. The C++ new and delete operators can be used to allocate memory in the free store area, but it is easier to implement a class that performs all string-handling operations consistently and efficiently, rather than to create and delete each string individually. Implementing a string-handling class is possible because the new and delete operators can be used from within member functions and pointers are valid class members.

String operations often require knowing the string's length. The strlen() function defined in the string.h header file returns this value. Alternatively, we can implement a string as an object that contains a pointer to a buffer that holds the string itself and a variable that represents its length. A parameterized constructor can take care of initializing the string object storage using the new operator as well as its length parameter. The contents of the string passed by the caller are then copied into its allocated storage. This operation determines that the two data members associated with each string object are stored independently, however, they remain associated to the object and can be readily accessed as a pair.

Box 5.

```
//************************************************************
// C++ program to illustrate an object chain
// Filename: SAMP-05.CPP
//************************************************************

#include <iostream.h>

//***************************
//    classes
//***************************
class Division {
public:
 float Div(float param1, float param2, int opcode) {
 if (opcode == 4)
  return (param1 / param2);
 else
   return 0;
 }
};

class Multiplication {
private:
 Division obj_div;
public:
 float Mult(float param1, float param2, int opcode) {
 if (opcode == 3)
  return (param1 * param2);
 else
  return (obj_div.Div(param1, param2, opcode));
 }
};


class Subtraction {
private:
 Multiplication obj_mult;
public:
 float Sub(float param1, float param2, int opcode) {
 if (opcode == 2)
  return (param1 - param2);
 else
```

continued on following page

Box 5. continued

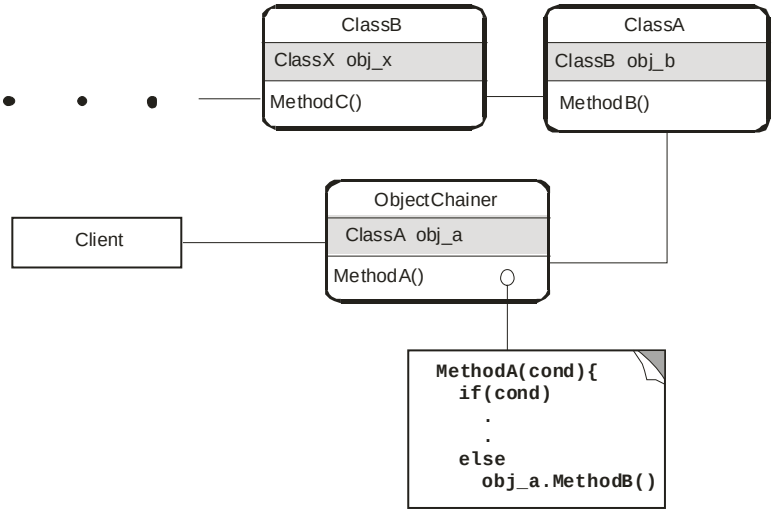
```
  return (obj_mult.Mult(param1, param2, opcode));
 }
};

class Addition {
private:
Subtraction obj_sub;
public:
 float Add(float param1, float param2, int opcode) {
 if (opcode == 1)
  return (param1 + param2);
 else
  return (obj_sub.Sub(param1, param2, opcode));
 }
};

class Operation{
private:
 float param1;
 float param2;
 int opcode;
 Addition obj_add;
public:
 Operation(float val1, float val2, int op) {
  param1 = val1;
  param2 = val2;
  opcode = op;
 }
 float SelectOp() {
 return (obj_add.Add(param1, param2, opcode));
 }
};

//***************************
//     main()
//***************************
main() {
 Operation obj_1(12, 6, 1); // Declaring objects of the
 Operation obj_2(12, 6, 2); // four established opcodes
 Operation obj_3(12, 6, 3);
 Operation obj_4(12, 6, 4);
```

continued on following page

Box 5. continued

```
 cout << "\n";
// Performing operation on objects
 cout << "Operation on obj_1: " << obj_1.SelectOp() << "\n";
 cout << "Operation on obj_2: " << obj_2.SelectOp() << "\n";
 cout << "Operation on obj_3: " << obj_3.SelectOp() << "\n";
 cout << "Operation on obj_4: " << obj_4.SelectOp() << "\n";

 return 0;
}
```

Figure 10. Object chain pattern



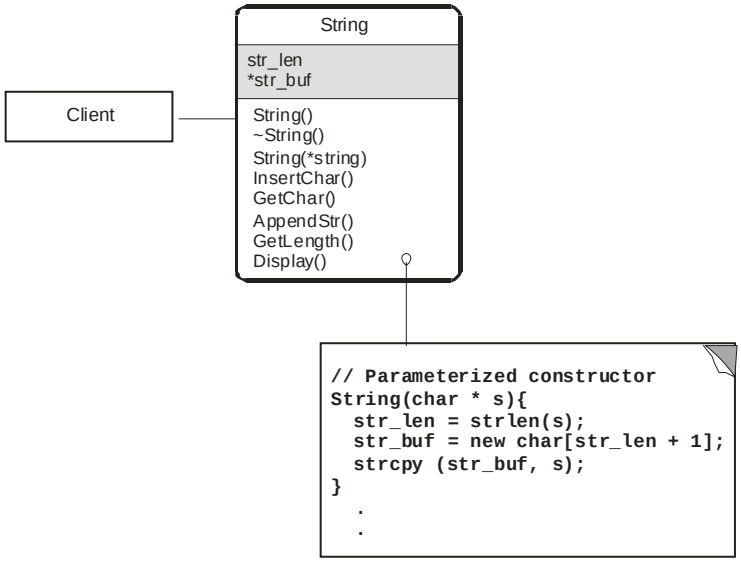
In addition to the parameterized constructor, the proposed class could have a default constructor that initializes both data members to zero whenever it is necessary. An explicit destructor method is also required in this case. The fact that the new operator is used to allocate space for the string buffer implies that the delete operator must be used to free the allocated memory. Other useful methods would be to get the length of the string and to insert a character in a string, to read a string character by specifying its index, and to append a new string to an existing one. Additional functionalities can be added by editing the class or by inheriting its methods. Figure 11 class diagram can serve as a template in this case.

The following program (Box 6) implements the string handling class template of Figure 11.

COMBINING FUNCTIONALITIES

In the implementation of libraries, toolkits, and application frameworks (Deutsch, 1989) we often come across a situation in which a given functionality is scattered among several classes. Rather than giving a client access to each one of these individual classes it is often a reasonable alternative to combine several methods into a single class which can then be presented and documented as standard interface.

Figure 11. String handler class template



A Mixer Pattern

One of the practical uses of multiple inheritance is in combining functionalities by creating a class that inherits from two or more classes. The inheriting class serves to mix and unify the methods of the participating base classes. The class in Figure 12 shows a pattern based on multiple inheritance into a mixer class.

The following code fragment shows how to implement multiple inheritance in the case of the mixer class in Figure 12:

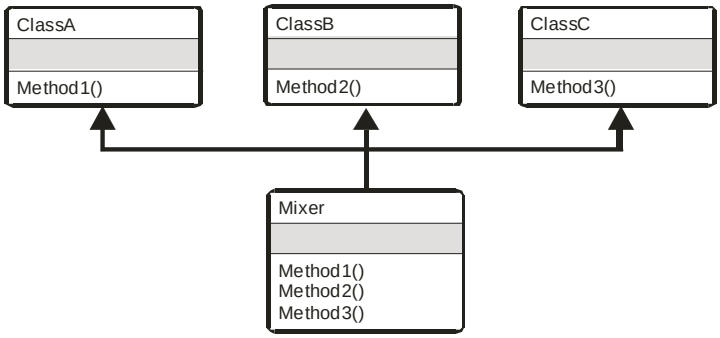
```
// Multiple inheritance
class Mixer : public ClassA, public ClassB,
public ClassC {
 // Implementation
};
```

When implementing multiple inheritance we must be careful to avoid situations which could lead to resolution conflicts; for example, inheriting a method that has the same name and interface in the parent classes, or inheriting multiple copies of the same method.

AN OBJECT-CLASSIFIER TEMPLATE

The objects in a class need not have identical signatures. A class can contain several parameterized constructors that create objects with different numbers or types of parameters. In this sense the constructors serve as object classifiers since each constructor executes according to the object's signature. The constructor can also store the object's type in a variable so that the object can be declassified during processing. For example, we wish to provide a class that calculates the area of various types of geometrical figures. Some figures such as the square require a single parameter, other figures such as the rectangle, have two parameters, and still others like the parallelogram have three parameters. If there is a parameterized constructor for each object signature, then the processing is automatically directed to the corresponding constructor, which can also preserve the object's type by storing an associated code. This action of the constructor is consistent with the notion of function overloading.

Figure 12. Mixer pattern for combining disperse functionalities



Processing routines can dereference the object type and proceed accordingly.

Implementing the Object Classifier

The following program (Box 7) shows the processing for implementing a class named GeoFigure with four constructors: a default constructor that zeroes all the variables, and three parameterized constructors, one for each object signature. The constructors of the GeoFigure class perform object classification as the objects are created.

Observe in the program SAMP-07.CPP that processing operations that are often the client's burden are now handled by the class. The classifier class encapsulates knowledge about each object type, which is encoded and preserved with its signature. Thereafter, client code need not classify objects into squares, rectangles, or parallelograms, since this information is held

Box 7.

```
//************************************************************
// C++ class template for an object classifier class
// Filename: SAMP-07.CPP
//************************************************************

#include <iostream.h>
#include <math.h>

//***************************
//    classes
//***************************
class GeoFigure {
private:
 float dim1;
 float dim2;
```

continued on following page

Box 7.

```
 float dim3;
 int fig _ type;
public:
// Declaration of four constructors for GeoFigure class
 GeoFigure();
 GeoFigure(float);
 GeoFigure(float, float);
 GeoFigure(float, float, float);
// Area() method uses object signature
 float Area();
};

// Parameterless constructor
GeoFigure::GeoFigure() {
 dim1 = 0;
 dim2 = 0;
 dim3 = 0;
 fig _ type = 0;
}
// Constructor with a single parameter
GeoFigure :: GeoFigure(float x){
 dim1 = x;
 fig _ type = 1;
}
// Constructor with two parameters
GeoFigure :: GeoFigure(float x, float y){
 dim1 = x;
 dim2 = y;
 fig _ type = 2;
}
// Constructor with three parameters
GeoFigure :: GeoFigure(float x, float y, float z){
 dim1 = x;
 dim2 = y;
 dim3 = z;
 fig _ type = 3;
}

float GeoFigure::Area() {
 switch (fig _ type) {
  case (0):
```

continued on following page

Box 7.

```
    return 0;
   case (1):
    return dim1 * dim1;
   case (2):
    return dim1 * dim2;
   case (3):
    return dim1 * (dim2 * sin(dim3));
   }
  return 0;
  }
//***************************
//     main()
//***************************
main() {
 GeoFigure fig0;       // Objects with different signatures
 GeoFigure fig1(12);
 GeoFigure fig2(12, 6);
 GeoFigure fig3(12, 6, 0.6);

// Calculating areas according to object signatures
 cout << "\nArea of fig1: " << fig1.Area();
 cout << "\nArea of fig2: " << fig2.Area();
 cout << "\nArea of fig3: " << fig3.Area();
 cout << "\nArea of fig0: " << fig0.Area() << "\n";

 return 0;
}
```

in the class and handled transparently by its methods. The objects created by the classifier class know not only their dimensions but also their geometrical type, which in turn defines the processing operations necessary for performing calculations such as the area.

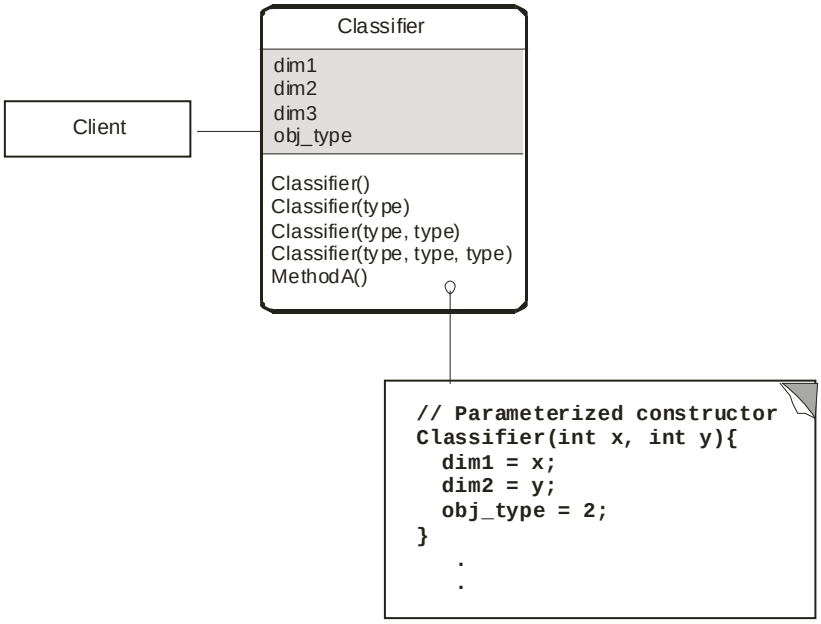
A classifier class is appropriate whenever there are objects with different signatures and their signatures determine the applicable processing operations or methods. Figure 13 is a generalized diagram for a classifier class; it can serve as a template for implementing this type of processing.

COMPOSING MIXED OBJECTS

Libraries, toolkits, and frameworks often provide two types of services to a client. A first level service performs the more elementary operations. A second-level service, called an aggregate or composite, allows combining several primitives to achieve a joint result. The pattern is referred to as a composite pattern. The composite is based on the notion of a Glyph, which is a class that encompasses all objects in

a document (Calder, 1990). Composite patterns are found in most object-oriented systems,

Figure 13. Object classifier class template



including the original View class of Smalltalk (Krasner & Pope, 1988). Also in financial applications, aggregation of assets in a portfolio have been modeled using a Composite class (Birrer & Eggenschwiler, 1993).

A Graphics Toolkit

For example, a drawing program provides primitives for drawing geometrical figures such as lines, rectangles, and ellipses, for displaying bitmaps, and for showing text messages. A second-level function (the composite) allows combining of the primitive elements into a single unit that is handled as an individual program component. In the context of graphics programming, the term "descriptor" is often used to represent a drawing primitive and the term "segment" to represent a composite that contains one or more primitives.

Often a toolkit designer gives access to both the primitive and composite functions. In other words, the programmer using the drawing toolkit mentioned in the preceding paragraph would be able to create drawings that contained any combination of primitive and composite objects using a single, uniform, interface. In this example it may be useful to think of a composite as a list of instructions that includes the methods of one or more primitives. Figure 14 shows an image that contains both primitive and composite objects. The composite objects consist of a rectangle, a bitmap, an ellipse, and a text message. The primitive objects are text, a rectangle, and an ellipse.

The class structure for the sample toolkit could consist of a class for every one of the primitive operations and a composite class for combining several primitives into a drawing segment. An abstract class at the top of the hierarchy can serve to define an interface. Figure 15 shows the class structure for the graphics toolkit.

Note in Figure15 that the abstract class Image provides a general interface to the toolkit. The class Segment contains the implementation of the segment-level operations, the methods CreateSegment(), DeleteSegment(), and DrawSegment(). The drawing primitives are the leaves of the tree

Figure 14. Primitive and composite objects in a graphics toolkit

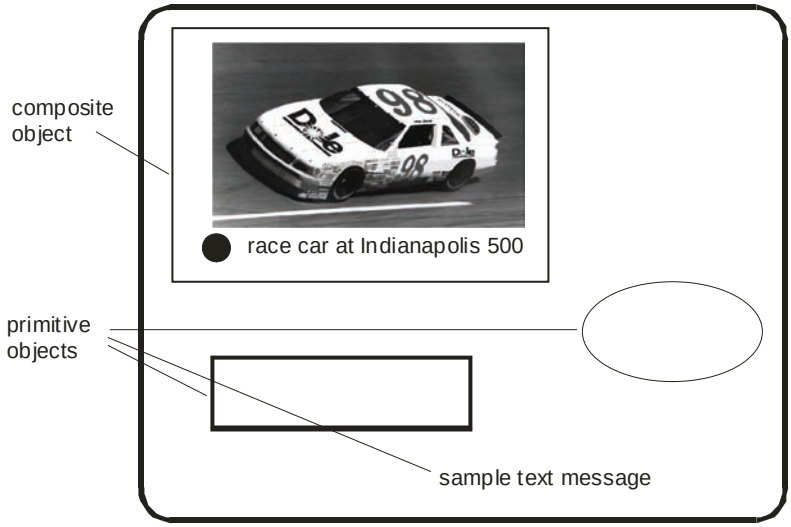
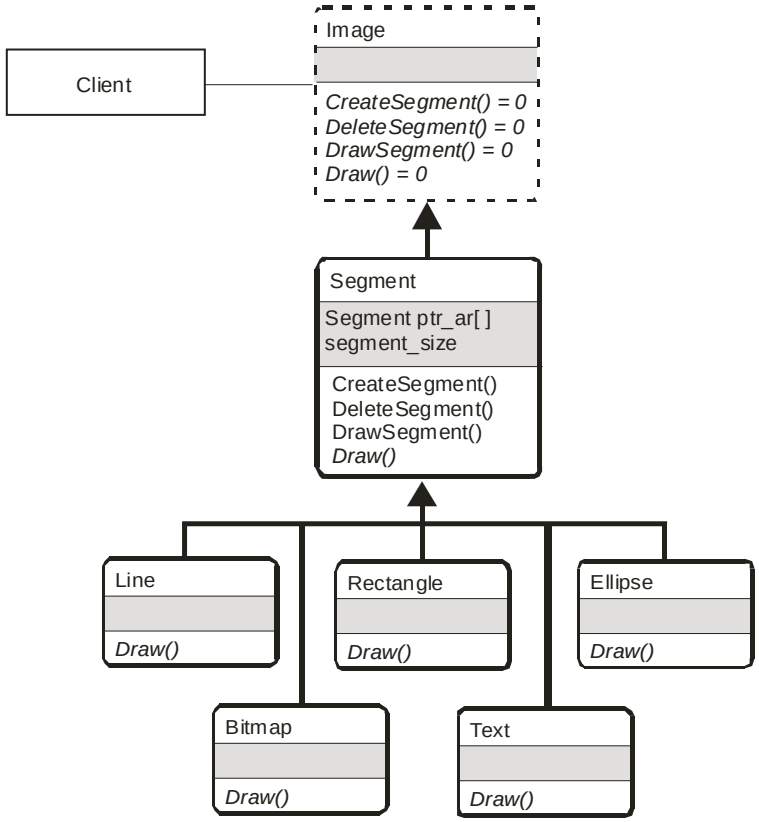


Figure 15. Tree structure for creating simple and composite objects



structure. Program SAMP-08.CPP (Box 8) is a partial implementation of the class diagram in Figure 15. Note that the classes Bitmap and Text were omitted in the sample code.

In the program SAMP-08.CPP the segment operation is based on an array of pointers. For this mechanism to work we need to implement run-time polymorphism since the composite (each instance of the Segment class) is created during program execution. In C++ run-time polymorphism can be achieved by inheritance and virtual functions. In this case the function Draw() is a pure virtual function in the abstract class Image, a virtual function in the class Segment, and is implemented in the leaf elements of the tree, which are the classes Line, Rectangle, and Ellipse. By making Draw() a simple virtual function we avoid making Segment into an abstract class. Therefore we can instantiate objects of Segment and still access the polymorphic methods in the leaf classes.

The actual code for implementing an array of pointers to objects has several interesting points. The array is defined in the private group of the Segment class, as follows:

```
Segment *ptr_ar[100];
```

This creates an array of pointers to the class Segment, and assigns up to 100 possible entries for each instantiation. The actual pointers are inserted in the array when the user selects one of the menu options offered during the execution of CreateSegment(). At this time a pointer to the Segment base class is reset to one of the derived classes, one of the Draw() methods at the leaves of the inheritance structure. The selected method is then placed in the pointer array named ptr_ar[]. For example, if the user selected the r (rectangle) menu option the following lines would instantiate and insert the pointer:

```
case ('r'):
    ptr_ss = &obj_rr; // Base pointer
set to derived object
  ptr_ar[n] = ptr_ss; // Pointer placed
in array
   n++; // Array index is bumped
   break;
```

Although it may appear that the same effect could be achieved by using a pointer to the method in the derived class, this is not the case. In this application a pointer to a derived class will be unstable and unreliable.

At the conclusion of the CreateSegment() method it is necessary to preserve with each object a count of the number of points that it contains. The seg_size variable is initialized to the number of pointers in the array in the statement:

```
seg_size = n;
```

At the conclusion of the CreateSegment() method, the array of pointers has been created and initialized for each object of the Segment class, and the number of pointers is preserved in the variable seg_size. Executing the segment is a matter of recovering each of the pointers in a loop and executing the corresponding methods in the conventional manner. The following loop shows the implementation:

```
for(int x = 0; x < seg_size; x++)
  ptr_ar[x]->Draw();
```

PATTERN FOR AN OBJECT FACTORY

By eliminating all the unnecessary elements in the class structure of Figure 15 we can construct a general class pattern for creating simple and composite objects. This version of the Composite class can be considered as a simple object factory which uses an array to store one or more pointers to objects. In addition, each object of the Composite class keeps count of the number of pointers

Box 8.

```
//************************************************************
// C++ program to illustrate the creation of simple and
// composite objects
// Filename: SAMP-08.CPP
//************************************************************

#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

//**************************
//    classes
//**************************
// Image class provides a general interface
class Image {
public:
 void virtual CreateSegment() = 0;
 void virtual DrawSegment() = 0;
 void virtual Draw() = 0;
};

class Segment : public Image {
private:
 Segment *ptr_ar[100];       // Array of pointers
 int seg_size;           // Entries in array
public:
 void CreateSegment();
 void DrawSegment();
 void DeleteSegment();
 void virtual Draw() { return; };
};


class Rectangle : public Segment {
public:
 void Draw() { cout << "\ndrawing a rectangle"; }
};

class Ellipse : public Segment {
public:
 void Draw() { cout << "\ndrawing an ellipse"; }
```

continued on following page

Box 8. continued

```
};

class Line : public Segment {
public:
 void Draw() { cout << "\ndrawing a line"; }
};

// Implementation of methods in Segment class
void Segment::CreateSegment() {
 char select;
 int n = 0;        // Entries in the array
// Objects and pointers
 Segment obj_ss;
 Segment *ptr_ss;
 Line   obj_ll;
 Rectangle obj_rr;
 Ellipse obj_ee;

 cout << "\n opening a segment...\n";
  cout << "Select primitive or end segment: "
 << "\n l = line"
 << "\n r = rectangle"
 << "\n e = ellipse"
 << "\n x = end segment"
 << "\n SELECT: ";
 do {
  select = getche();
  switch(select) {
  case ('l'):
   ptr_ss = &obj_ll;
   ptr_ar[n] = ptr_ss;
   n++;
   break;
  case ('r'):
   ptr_ss = &obj_rr;
   ptr_ar[n] = ptr_ss;
   n++;
   break;
  case ('e'):
   ptr_ss = &obj_ee;
   ptr_ar[n] = ptr_ss;
   n++;
```

continued on following page

Box 8. continued

```
    break;
   case ('x'):
    break;
   default:
    cout << "\nInvalid selection - program terminated\n";
    exit(0);
  }
  }
  while( select != 'x');
   seg_size = n;
   cout << "\n closing a segment...";
}

void Segment::DrawSegment() {
 cout << "\n  displaying a segment...";
 for(int x = 0; x < seg_size; x++)
  ptr_ar[x]->Draw();
 cout << "\n  end of segment display ...";
 return;
}
//***************************
//     main()
//***************************
main() {
 Segment  obj_s;
 Line   obj_l;
 Rectangle obj_r;
 Ellipse  obj_e;

// Creating and drawing a segment
 cout << "\n\nCalling CreateSegment() method";
 obj_s.CreateSegment();
 obj_s.DrawSegment();
// Drawing individual objects
 obj_l.Draw();
 obj_r.Draw();
 obj_e.Draw();

 return 0;
}
```

in the array. This count is used in dereferencing the pointer array.

Alternatively, the pointer array can be implemented without keeping a pointer counter by inserting a NULL pointer to mark the end of the array. This NULL pointer then serves as a marker during dereferencing. In either case, the corresponding methods in the leaf classes are accessed by means of the pointers in the array. Method selection must be implemented by dynamic binding. In C++ the polymorphic method must be virtual in the composite class. The pattern is shown in Figure 16.

A Simplified Implementation

We can simplify the concept of primitive and composite objects, as well as their implementation in code, by allowing a composite that consists of a single primitive object. For example, if in Figure 15 we permit a segment that consists of a single primitive, then the client needs never to access the primitives directly. This makes the interface less complicated. In many cases this option should be examined at design time.

RECURSIVE COMPOSITION

In previous sections we considered the case of a composite class that contains simple or composite objects. We also looked at the alternative of a composite object that consists of a single primitive as a way of simplifying the interface. However, we have not yet considered the possibility of a composite object containing another composite (Gamma et al., 1995). Based on the class structure shown in Figure 15 we can construct an object diagram in which a Segment can contain another Segment as shown in Figure 17. Note that in this case we have preserved the distinction between primitives and composites.

Implementation Considerations

In the previous example recursive composition is based on a nested reference to the CreateSegment() method of the Segment class. However, recursion is often accompanied by a new set of problems; this case is no exception. The first consideration is to access the CreateSegment() method. Three possibilities are immediately evident:

Figure 16. Pattern for an object factory class

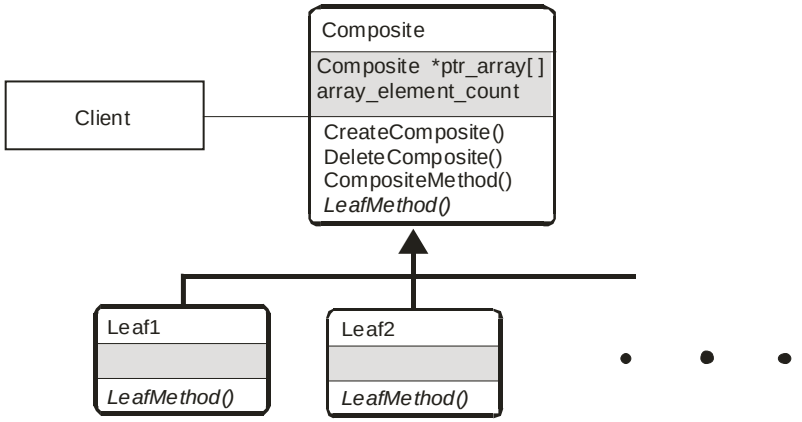
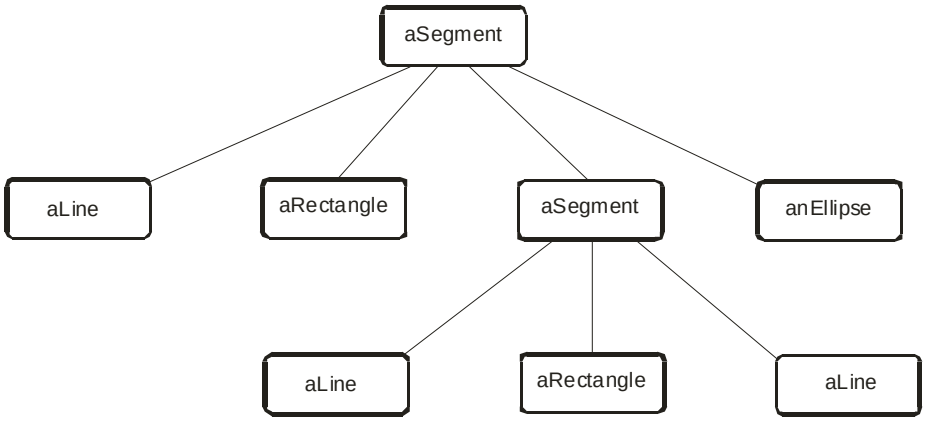


Figure 17. Object diagram for recursive composition



1. Since CreateSegment() is called from within the class, it can be referenced without instantiating a specific object.
2. We can access the CreateSegment() method by means of a this pointer. In fact, this is a different syntax but has the same result as the previous case. In both instances the current object is used.
3. We can create a new object and use it to access the CreateSegment() method.

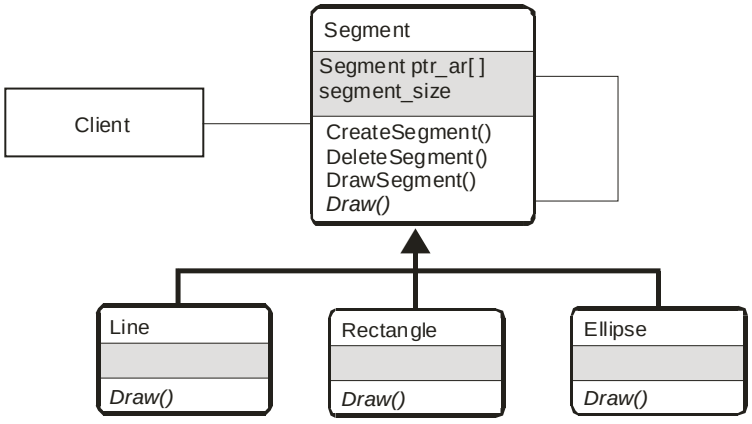
Which method is suitable depends on the problem to be solved. Figure 18 shows a class diagram for recursively accessing the CreateSegment() method using the original object.

The program SAMP-09.CPP (Box 9) shows the implementation of the class diagram in Figure 18.

Several points in the code merit comment. In the first place notice that recursion occurs on the same object originally referenced at call time. This is accomplished by means of the C++ this pointer, in the following statement:

```
this->CreateSegment();
```

Figure 18. Class diagram for recursive composition



Box 9.

```
//************************************************************
// C++ program to illustrate recursive composition
// Filename: SAMP-09.CPP
//************************************************************

#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

//***************************
//    classes
//***************************
class Segment {
private:
 Segment *ptr_ar[100];
 int seg_size;           // Entries in array
public:
 void CreateSegment();
 void DrawSegment();
 void DeleteSegment();
 void virtual Draw() { return; };
};

class Rectangle : public Segment {
public:
 void Draw() { cout << "\ndrawing a rectangle"; }
};

class Ellipse : public Segment {
public:
 void Draw() { cout << "\ndrawing an ellipse"; }
};

class Line : public Segment {
public:
 void Draw() { cout << "\ndrawing a line"; }
};
// Global variable for controlling recursive implementation
// of the CreateSegment() method
 int n;
 int instance = 0;
```

continued on following page

Box 9. continued

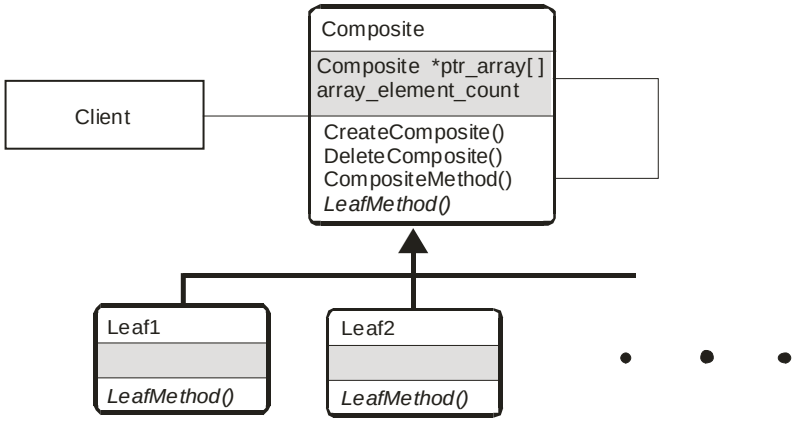
```
// Implementation of methods in Segment class
void Segment::CreateSegment() {
 char select;
// Entries in the array
// Objects and pointers
 Segment obj_ss;     // An object
 Segment *ptr_ss;     // Pointer to object
 Line   obj_ll;     // Object list
 Rectangle obj_rr;
 Ellipse obj_ee;

 if(instance == 0)
  n = 0;

  cout << "\n opening a segment...\n";
  cout << "Select primitive or end segment: "
  << "\n l = line"
  << "\n r = rectangle"
  << "\n e = ellipse"
  << "\n n = nested segment"
  << "\n x = end segment"
  << "\n SELECT: ";
 do {
  select = getche();
  switch(select) {
  case ('l'):
   ptr_ss = &obj_ll;     // Pointer to object initialized
   ptr_ar[n] = ptr_ss;    // and stored in array
   n++;
   break;
  case ('r'):
   ptr_ss = &obj_rr;
   ptr_ar[n] = ptr_ss;
   n++;
   break;
  case ('e'):
   ptr_ss = &obj_ee;
   ptr_ar[n] = ptr_ss;
   n++;
   break;
  case ('n'):
```

continued on following page

Box 9. continued

```
    cout << "\n   nested segment...";
    instance = 1;
    this->CreateSegment();
    cout << "\n   nested segment closed ...";
    cout << "\n SELECT: ";
    continue;

   case ('x'):
    break;
   default:
    cout << "\nInvalid selection - program terminated\n";
    exit(0);
  }
  }
  while( select != 'x');
   seg_size = n;
   cout << "\n closing a segment...";
   instance = 0;         // Reset instance control
}

void Segment::DrawSegment() {
 cout << "\n  displaying a segment...";
 for(int x = 0; x < seg_size; x++)
  ptr_ar[x]->Draw();
 cout << "\n  end of segment display ...";
 return;
}
//***************************
//     main()
//***************************
main() {
 Segment  obj_s;

// Creating and drawing a segment with possible nested
// segments
 cout << "\n\nCalling CreateSegment() method";
 obj_s.CreateSegment();
 obj_s.DrawSegment();

 return 0;
}
```

Figure 19. Pattern for recursive composition



The CreateSegment() method could have been accessed without the this pointer since it is allowed, within the same class, to access methods directly. When CreateSegment() is accessed recursively, all the local variables are automatically reinitialized. Since the program requires a count of the number of pointers in the pointer array, we made the iteration counter (variable n) a global variable and created a switch variable named instance. This variable is set to 1 when CreateSegment() is called recursively, determining that counter variable n is not cleared on entry to the method. The result is that n holds the number of pointers inserted in the pointer array, whether they were entered directly or recursively.

A Recursion Pattern

The pattern for recursive composition is similar to the one in Figure 16, except that in recursion there is an arrow pointing to the same composite class. This is shown in Figure 19.

CONCLUSION

Software design is one of the most laborious and time-consuming phases of program development. In object-oriented systems design reuse is based on classes and object structures that solve a particular design problem and on the assumption that these class structures can be applied to other similar problems. The most recent approach to design reuse is based on class associations and relationships called patterns or object models. In this sense a programming problem is at the origin of every pattern.

We have described the use of design patterns and template classes as reusable components in program design. The discussion has been complemented with class diagrams and examples implemented in code. The chapter introduces the notion of a class template as a structure that describes functionality and object relations within a single class, while patterns refer to structures of communicating and interacting classes.

REFERENCES

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A pattern language*. New York: Oxford University Press.

Beck, K., & Johnson, R. (1994, July). *Patterns generate architectures*. Paper presented at

the European Conference on Object-Oriented Programming, Bologna, Italy (pp. 139-149). Springer-Verlag.

Birrer, A., & Eggenschwiler, T. (1993, July). *Frameworks in the financial engineering domain: An experience report.* Paper presented at the European Conference on Object-Oriented Programming, Kaiserslautern, Germany (pp. 21-35). Springer-Verlag.

Calder, P. R., & Linton, M. A. (1992, October). *The object-oriented implementation of a document editor.* Paper presented at the Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings, Vancouver, British Columbia, Canada (pp. 1-15). ACM Press.

Coplien, J. O. (1992). *Advanced C++ programming styles and idioms.* Reading, MA: Addison-Wesley.

Deutsch, L. P. (1989). Design reuse and frameworks in the Smalltalk-80 system. *Software reusability, Volume II: Applications and experience* (pp 57-71). Reading, MA: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Krasner, G. E., & Pope, S. T. (1988, August/September). A cookbook for using the model-view controller user interface paradigm in Smalltalk 80. *Journal of Object-Oriented Programming, 1*(3)26-49.

Vlissides, J. M., & Linton, M. A. (1990, July). Unidraw: A framework for building domain-specific graphical editors. *ACM Transactions on Information Systems, 8*(3), 237-269.

KEY TERMS

Concealer Pattern: A class pattern designed to hide an inheritance structure so that its existence and operation becomes transparent to client code.

Chain Reaction Pattern: A class pattern that focuses on the property of classes that instantiate objects of other classes higher in the class hierarchy, thus starting a chain reaction in the production of objects.

Class Template: A structure of interactive objects that implements a particular functionality within a single class.

Design Pattern: A structure of interactive classes and communicating objects that provide a solution to a software design problem.

Interface Pattern: A pattern that provides access to other classes that implement a desired functionality.

Mixer Pattern: A class that inherits from two or more classes, thus providing a unified interface to the methods and objects in these classes.

Object Chaining: A chain of objects that successively examines an operand pass to them and makes some decision based on the result of this examination.

Object Classifier: A template for a class with constructors that create objects with different numbers and types of parameters.

Object Factory: A composite class that creates both simple and composite objects.

Unifier Pattern: A class pattern designed to present a unified and friendly interface to a set of classes that perform different operations.

This work was previously published in Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, edited by M. Syed; and S. Syed, pp. 388-432, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 2.4
Motivation in Component-Based Software Development

G. Chroust
J. Kepler University Linz, Austria

INTRODUCTION

Information systems are designed for the people, by the people. The design of software systems with the help of software systems is another aspect of human-computer interfaces. New methods and their (non-)acceptance play an important role. Motivational factors of systems developers considerably influence the type and quality of the systems they develop (Arbaoui, Lonchamp & Montangero, 1999; Kumar & Bjoern-Andersen, 1990). To some extent, the quality of systems is a result of their developers' willingness to accept new and (supposedly) better technology (Jones, 1995). A typical example is component-based development methodology (Bachmann et al., 2000; Cheesman & Daniels, 2001). Despite considerable publication effort and public lip service, component-based software development (CBD) appears to be getting a slower start than anticipated and hoped for. One key reason stems from the psychological and motivational attitudes of software developers (Campell, 2001; Lynex & Layzell, 1997). We therefore analyze the attitudes that potentially hamper the adoption of the component-based software development approach. Maslow's Hierarchy of Need (Boeree, 1998; Maslow, 1943) is used for structuring the motives.

BACKGROUND

The Human Side of Software Engineering

Kunda and Brooks (1999) state that "software systems do not exist in isolation ... human, social and organizational considerations affect software processes and the introduction of software technology. The key to successful software development is still the individual software engineer" (Eason et al., 1974; Kraft, 1977; Weinberg, 1988). Different software engineers may account for a variance of productivity of up to 300% (Glass, 2001). On the other hand, any other single factor is not able to provide an improvement of more than 30%. The influence of an individual's motivation, ability, productivity, and creativity has the biggest influence by far on the quality of software development, irrespective of the level of technological or methodological support. Therefore, it is worthwhile investigating for what reasons many software engineers do not fullheartedly accept component-based methods (Lynex & Layzell, 1997).

Software development in general introduced a new type of engineers who show marked differences when compared to (classical) engineers (Badoo & Hall, 2001; Campell, 2001; Eason et al., 1974; Kraft, 1977; Kunda & Brooks, 1999; Lynex & Layzell, 1997). The phenomenon is not fully understood yet but seems to have to do with the peculiarities of software (Brooks, 1986), the type of processes and environments needed to develop software (Kraft, 1977), and especially to the proximity of software development to other mental processes (Balzert, 1996).

Maslow's Hierarchy of Needs

Maslow's theory (Boeree, 1998; Huitt, 2002; Maslow, 1943; McConnell, 2000) provides a practical classification of human needs by defining a five-level Hierarchy of Needs (Figure 1).

The five levels are as follows:

- Basic Physiological Needs (Survival): At this level, the individual is fighting for survival against an adverse environment, trying to avert hunger, thirst, cold, and inconvenient and detracting physical work environments.
- Security (Physical, Economic ...): On this level, the individual is concerned with the stability of his or her future and the safety of the environment. Worries include job security, loss of knowledge, loss of income, health, and so forth.
- Social Environment: This category includes the need to have friends, belong to a group, and to give and receive love.
- Recognition: Individuals strive to receive appropriate recognition and appreciation at work and to be recognized as having a valuable opinion.
- Self-Fulfillment: This level is considered the highest stage attainable in the development of a person, drawing satisfaction from the realization of one's own contribution to a goal and one's fullfillment of their full potential as a human being.

Reuse and Component-Based Software Development (CBD)

An old dream in software development is to avoid unnecessary duplication of work by consistently and systematically reusing existing artifacts. Reuse promises higher productivity, shorter time-to-market, and higher quality (Allen, 2001; Cheesman & Daniels, 2001). Initially, ready-made pieces of software were made available; these delivered a defined functionality in the form of a black box (i.e., without divulging the internal structure to the buyer/user). They were called COTS (commercials off the shelf) (Voas, 1998). Later, an improved and more restricted concept was employed: software components (Bachmann et al., 2000; Cheesman & Daniels, 2001; Woodman et al., 2001). Software components have to

Figure 1. Maslow's hierachy of needs

fulfill additional requirements, restrictions, and conventions beyond the properties of COTS. To a user of a software component, only its interfaces and functionality are known, together with the assurance that the component obeys a specific component model. This component model defines how the component can be integrated with other components, the conventions about the calling procedure, and so forth. The internal structure, code, procedures, and so forth are not divulged—it is a black box.

Systematic, institutionalized CBD needs a change in the attitude of software engineers, different work organization, and a different organization of the whole enterprise (Allen, 2001).

Component-Based Development and Software Engineers' Needs

The acceptance of a new technology often meets with strong opposition caused by psychological motives, which can be traced to Maslow's Hierarchy of Needs.

Basic Physiological Needs

This level does not have any strong relevance; software engineering is a desk-bound, safe, non-endangering activity. We have to recognize, however, that very often software engineers have to struggle with adverse infrastructure (floor space, noise, etc.) (deMarco, 1985).

Security

The desire for security is threatened by numerous factors. The fears can be categorized into four groups:

Losing the Job or Position

- Job Redundancy: CBD promises considerably higher productivity and less total effort as a result of removing the redundancy of reimplementing already existing functions. This carries the thread of making an individual redundant, especially since the development of components very often is outsourced to some distant organization (e.g., India).
- Implementing vs. Composing: deRemer (1976) stressed the difference between implementing a module/component (programming in the small) and building (composing) a system out of components (programming in the large). He emphasized the need for a different view and for new approaches and tools. Programming in the large needs a systems view, making much of long-learned patterns of work obsolete, even counter-productive.
- Changed Job Profile: The necessity of integrating existing components requires a different mindset than one implementing some program from scratch (Vitharana, 2003). Does the software engineer have the ability or qualifications to fulfill the new profile of an integrator?
- Loss of Knowledge and "Guru" Status: In traditional development, considerable domain know-how and low-level development know-how rests in the heads of seasoned developers having developed software for many years. The use of components encapsulates and hides both implementation details and domain know-how. In addition, system development methods change. Much of the accumulated experience and know-how previously valuable to the employing institution becomes irrelevant.
- De-Skilling: In addition, certain de-skilling takes place at the lower level of software development (Kraft, 1977). The need for increased skills with respect to performing high-level composition activities often is not recognized and appreciated by the individuals.

Loss of Low-Level Flexibility

- Pre-Conceived Expectations: Components often do not provide exactly what the original requirements specified. A developer then is challenged to find a compromise between the user requirements and the available components—a job profile dramatically different from developing bespoke software (Vitharana, 2003). Engineers also have to live with good enough quality (Bach, 1997; ISO/IEC, 2004), as provided by the components and often are not able to achieve best quality. This is often difficult to accept emotionally.
- Revision of Requirements: Mismatches between stated requirements and available components make it necessary to revise and adapt requirements (Vitharana, 2003). Requirements are no longer set in stone, in contrast to the assumptions of classical development methods (e.g., the waterfall model).
- Uncertainty About Functionality of Components: By definition, the internal structure of a component is not revealed (Bachmann et al., 2000). Consequently, the developer has to rely on the description and claims provided by the component provider (Crnkovic & Larsson, 2002; Vitharana, 2003).

Lack of Confidence

- Distrust in Component Quality: Quality problems experienced in the past have created a climate of distrust with respect to other developers' software products. This feeling of distrust becomes stronger with respect to components, because their internals are not disclosed (Heineman. 2000; Vitharana, 2003). This situation becomes worse for so-called software of unknown provenance (SOUP) (Schoitsch, 2003). The current interest in open source programs is an indicator of a movement in the opposite direction.
- Questions About Usability: Besides the issue of quality, problems with portability and interoperability of components, as often experienced with COTS, also reduce the confidence in using components (Vecellio & Thomas, 2001).
- Loss of Control of System: Engineers usually like to understand fully the behavior of the designed system (the why) and exercise control over the system's behavior (the how). In CBD, due to the black-box character of the components, understanding and control can be achieved only to a limited extent, leaving a vague feeling of uncertainty.

Effort for Reuse vs. New Development

- Uncertainty Concerning the Outcome of the Selection Process: Successful CBD depends to a considerable extent on the probability and effectiveness of finding a component with (more or less) predefined properties (Vitharana, 2003). Occasionally, this search will not be successful, causing a delay in the development schedule and some lost effort spent in the search.
- Effort Estimates: In general, software engineers underestimate the effort needed to build a system from scratch and overestimate the cost and effort of adapting a system. The reasons seem to be the initially necessary effort to achieve a certain familiarity with the whole system before making even small adaptations, the learning curve (Boehm & Basili, 2000), the difficulty, and often also the unwillingness of becoming familiar with somebody else's thoughts and concepts (not-invented-here syndrome).

Social Environment

- Reluctance to Utilize Outside Intellectual Property: Our society extends the notion

of ownership to immaterial products like ideas and intellectual achievements. They are protected by copyright, trademark, and patent legislation. Plagiarism is objected to and usually not sanctioned (Kock, 1999; Sonntag & Chroust, 2004). Reusing someone else's ideas is often deemed inappropriate.

- Immorality of Copying: In school, copying as a form of reuse and teamwork is usually discouraged. This might later cause some reluctance to actively share knowledge and to make use of someone else's achievements (Disterer, 2000).
- Adopting a New Technology: The adoption of a new technology seems to follow an exponential law (Jones, 1995). It starts with a few early adopters, and others follow primarily because of personal communication. The tendency of software developers to be introverts (Riemenschneider, Hardgrave, & Davis, 2002) might delay such a dissemination process.
- Change of Work Organization: CBD needs a different work organization (Allen, 2001; Chroust, 1996; Cusumano, 1991; Wasmund, 1995) resulting in rearranged areas of responsibility, power distribution, and status, potentially upsetting an established social climate and well-established conventions.

Recognition

A strong motivator for an individual is recognition by the relevant social or professional reference group, usually a peer group (Glass, 1983).

- Gluing vs. Doing: On the technical level, recognition usually is connected to a particular technical achievement. Gluing together existing components will achieve recognition only for spectacular new systems—and these are rare. Similarly, an original composer will gain recognition; whereas a person simply arranging music into potpourris usually goes unrecognized.
- Shift of Influence and Power: Successful CBD needs a change in organization (Allen, 2001; Kunda & Brooks, 1999), making persons gain or lose influence, power, and (job) prestige, a threat to the established pecking order.
- The CBD Water Carrier: Organizations heavily involved in CBD often separate the component development from component deployment (Cusumano, 1991). Component development to a large extent is based on making existing modules reusable "components as you go" and "components by opportunity" (Allen, 2001) and not on creating new ones from scratch "components in advance". Jobs in the reuse unit (similar to maintenance units) (Basili, 1990) might be considered to require less know-how and thus receive lower prestige, despite the fact that these jobs often require greater know-how and experience than designing components from scratch.
- Contempt for the Work of Others: The inherent individuality of software development, together with a multitude of different solutions to the same problem (i.e., there is always a better way), and the low quality of many software products have tempted many software developers into contempt for anyone else's methods and work (not-invented-here syndrome).
- Rewarding Searching Over Writing: As long as the amount of code produced (lines of code) is a major yard stick for both project size and programmer productivity searching, finding and incorporating a component will be less attractive than writing it anew.
- Accounting for Lost Search Effort: There is no guarantee that even after an extensive (and time-consuming) search an appropriate component can be found (Vitharana, 2003). In this case, management must accept these occasional losses so as not to discourage searching for components (Fichman & Kemerer, 2001).

Self-Fulfillment

- Not Invented Here: The ability to design wonderful systems is a strong motivator for software engineers. This feeling goes beyond recognition of peers—one knows it oneself. This makes it difficult for developers to accept other people's work (Campell, 2001; Disterer, 2000) in the form of components.
- No More Gold Plating: The feeling of self-fulfillment often cannot live with the knowledge that a system still should or must be improved, leading to endless effort in gold plating a system before delivery (or even thereafter). Externally acquired components cannot be modified (i.e., gold plated) because of the inaccessibility of their code.
- No Creative Challenge: Gluing together components provided by somebody else does not fulfill many engineers' attempt for novelty and, thus, is not considered to be a creative challenge. The highly creative process of finding the best-fitting component, restructuring the system, and perhaps modifying the requirements for using existing components often is not appreciated.
- No More Lone Artists: Software engineers aspire to become a Beethoven or a Michelangelo and not the directors of a museum arranging a high-class exhibition. Someone remarked that many system features are not needed by the users but are just a monument of their designer's intellectual capability. Assembling components utilizes only someone else's achievement.
- Lack of Freedom: The limited choice of available components, the limitations of a component model, the need to obey predefined interfaces, and so forth restrict the freedom of development and often are seen as a limit to creativity.

FUTURE TRENDS

The fact that the software industry needs a large step forward with respect to productivity, quality, and time-to-market will increase the reuse of software artifacts and, as a consequence, will encourage the use of component-based development methods. Understanding the basic state of emotion of software developers will support efforts to overcome developers' reluctance to accept this methodology by emphasizing challenges and opportunities provided by the new methods, re-evaluating the importance and visibility of certain tasks, changing job profiles, and changing the reward and recognition structure.

The consequence might be that software designers wholeheartedly accept component-based methodologies not only as an economic necessity but also as a means of achieving the status of a great designer, as postulated by Brooks (1986). In turn, this could lead to a new level of professionalism in software development and would allow component-based development methods to be utilized fully in the field of software engineering.

CONCLUSION

Soft factors like motivation and psychological aspects often play a strong role even in a technical, seemingly rational field like software engineering. We have discussed and identified key soft factors that often account for the slow uptake of component-based software development methods and relate them to the framework of Maslow's Hierarchy of Needs. The users of software components were the focus of this discussion. There are some indications that for providers of components, a different emotional situation exists (Chroust & Hoyer, 2004).

Recognition of the different levels of resistance and their psychological background will, among

other aspects, allow approaching the problems in a psychologically appropriate form. The need of the software industry to come to terms with its problems of quality, cost, and timeliness makes this a necessity.

REFERENCES

Abran, A., Moore, J., Bourque, P., Dupuis, R., & Tripp, L.L. (Eds.). (2004). Guide to the software engineering body of knowledge: 2004 version tech. rep. DTR 19759. International Organization for Standardization, Geneva, Switzerland.

Allen, P. (2001). Realizing e-business with components. Chicago: Addison-Wesley.

Arbaoui, S., Lonchamp, J., & Montangero, C. (1999). The human dimension of the software process. In J.-C. Derniame, D. Ali Kaba, & D.G. Wastell (Eds.), Software process: Principles, methodology, and technology (pp. 165-200). New York: Springer.

Bach, J. (1997). Good enough quality: Beyond the buzzword. IEEE Computer, 30(8), 96-98.

Bachmann, F., et al. (2000). Volume II: Technical concepts of component-based software engineering (technical report). CMU/SEI-2999-TR-008, ESC-TR-2000-007.

Badoo, N., & Hall, T. (2001). Motivators of software process improvement: An analysis of practitioners' views (technical report). Hertfordshire, UK: University of Hertfordshire.

Balzert, H. (1996). Lehrbuch der Software-Technik, Software Entwicklung. Heidelberg, Germany: Verlag.

Basili, V.R. (1990). Viewing maintenance as re-use oriented software development. IEEE Software, 7(1), 19-25.

Boehm, W., et al. (2000). Software cost estimation with COCOMO II. NJ: Prentice Hall.

Boeree, C.G. (1998). Abraham Maslow, biography. Retrieved November, 2001, from http://www.ship.edu/~cgboeree/maslow.html

Brooks, F. (1986). No silver bullet—Essence and accidents of software engineering. Information Processing 86, IFIP Congress, 1069-1076.

Campell, J. (2001). Course on reuse: Impediments to reuse. Retrieved from http://www.cs.qub.ac.uk/~J.Campbell/myweb/misd/ node8.html#section008 40000000000000000

Cheesman, J., & Daniels, J. (2001). UML components. Amsterdam, The Netherlands: Longman, Addison-Wesley.

Chroust, G. (1996). Software 2001—Ein weg in die Wiederverwendungswelt. In F. Lehner (Ed.), Softwarewartung und reengineering—Erfahrungen und entwicklungen (pp. 31-49). Germany: Deutscher Universitätsverlag.

Chroust, G., & Hoyer, C. (2004). Motivational issues in creating reusable software artifacts. In R. Trappl (Ed.), Cybernetics and systems 2004 (pp. 417-422). Vienna: Austrian Soc. for Cybernetic Studies.

Crnkovic, I., & Larsson, S. (Eds.). (2002). Building reliable component-based software systems. Boston: Artech House Publshing.

Cusumano, M. (1991). Factory concepts and practices in software development. IEEE Annals of the History of Computing, 13(1), 3-31.

deRemer, F.K.H. (1976). Programming-in-the-large versus programming in-the-small. IEEE Tr. on Software Eng., (292), 80-86.

Disterer, G. (2000). Individuelle und Soziale Barrieren Beim Aufbau von Wissenssammlungen. Wirtschaftsinformatik, 42(6), 539-546.

Eason, K., Domodaran, L., & Stewart, T. (1974). Interface problems in man-computer interaction. In E. Mumford, & H. Sackman (Eds.), Human choice and computers (pp. 91-105). North Holland Publshing Company.

Fichman, R.G., & Kemerer, C. (2001). Incentive compatibility and systematic software reuse. Journal of Systems and Software, 57(1), 54.

Glass, R. (1983). Software runaways. NJ: Prentice Hall.

Glass, R. (2001). Frequently forgotten fundamental facts about software engineering. IEEE Software, 18(3), 112-110.

Halaris, J., & Spiros, T. (1996). Reuse concepts and a reuse support repository. Proceedings of the IEEE Symposium and Workshop on Engineering of Computer Based Systems, Germany.

Heineman, G.T., et al. (2000). Component-based software engineering and the issue of trust. Proceedings of the 22nd International Conference on Software Engineering. Limerick, Holland.

ISO/IEC. (2004). ISO 25000: Software and systems engineering: Software product quality requirements and evaluation (SQuaRE)—Guide to SQuaRE [technical report]. International Organization for Standardization. Geneva, Switzerland.

Jones, C. (1995). Why is technology transfer so hard? IEEE Computer, 28(6), 86-87.

Kock, N. (1999). A case of academic plagiarism. Comm. ACM, Vol. 42(7), 94-104.

Kraft, P. (1977). Programmers and managers. Heidelberg, Germandy: Springer.

Kumar, K., & Bjoern-Andersen, N. (1990). A cross-cultural comparison of IS designer values. Comm ACM, 33(5), 528-538.

Kunda, D., & Brooks, L. (1999). Human, social and organisational influences on component-based software engineering. ICSE'99 Workshop on Component-Based Software Engineering, Los Angelos. Retrieved July, 2004, from http://www.sei.cmu.edu/cbs/icse99/papers/19/19.htm

Lynex, A., & Layzell, P. (1997). Understanding resistance to software reuse. Proceedings of the 8th International Workshop on Software Technology and Engineering Practice, London.

Maslow, A. (1943). A theory of human motivation. Psychological Review, 50, 370-396.

McConnell, S. (2000). Quantifying soft factors. IEEE Software, 17(6), 9-11.

Riemenschneider, C., Hardgrave, B.C., & Davis, F. (2002). Explaining software developer acceptance of methodologies: A comparison of five theoretical models. IEEE Trans. on Software Engineering, 28(12), 1135.

Schoitsch, E. (2003). Dependable embedded systems—Vision und roadmap. In G. Fiedler, & D. Donhoffer (Eds.), Mikroelektronik 2003, Wien (p. 33). Vienna, Austria: ÖVE Schriftenreihe.

Sonntag, M., & Chroust, G. (2004). Legal protection of component metadata and APIs. In R. Trappl (Ed.), Cybernetics and systems 2004. Vienna, Austria: Austrian Soc. for Cybernetic Studies.

Thomas, S., Hurley, S., & Barnes, D. (1996). Looking for the human factors in software quality management. Proceedings of the 1996 International Conference on Software Engineering: Education and Practice (SE:EP '96). Otago, New Zealand.

Vecellio, G., & Thomas, W.M. (2001). Issues in the assurance of component based software. Retrieved July, 2001, from http://www.mitre.org/pubs/edge perspectives/ march 01/vecellio.html

Vitharana, P. (2003). Risks and challenges of component-based software development. Comm. ACM, 46(8), 67-72.

Voas, J. (1998). Certifying off-the-shelf software components. IEEE Computer, 11(6), 53-59.

Wasmund, M. (1995). The spin-off illusion: Reuse is not a by-product. Proceedings of the 1995 Symposium on Software Reusability, Seattle, Washington.

Weinberg, G. (1988). Understanding the professional programmer. New York: Dorset House.

Woodman, M., Benedictsson, O., Lefever, B., & Stallinger, F. (2001). Issues of CBD product quality and process quality. Proceedings of the 4th ICSE Workshop: Component Certification and System Prediction, 23rd International Conference on Software Engineering (ICSE). Toronto, Canada.

KEY TERMS

Commercial Off the Shelf (COTS): Software products that an organization acquires from a third party with no access to the source code and for which there are multiple customers using identical copies of the component.

Component-Based Development (CBD): In contrast to classical development (waterfall-process and similar process models), CBD is concerned with the rapid assembly of systems from components (Bachmann et al., 2000) where:

- components and frameworks have certified properties; and
- these certified properties provide the basis for predicting the properties of systems built from components.

Component Model: A component model specifies the standards and conventions imposed on developers of components. This includes admissible ways of describing the functionality and other attributes of a component, admissible communication between components (protocols), and so forth.

Maslow's Hierarchy of Needs: Maslow's Hierarchy of Needs (Boeree, 1998; Maslow, 1943; McConnell, 2000) is used as a structuring means for the various factors. It defines five levels of need:

- self-fulfillment
- recognition
- social environment (community)
- basic physiological needs (survival)
- security (physical, economic ...)

In general, the needs of a lower level must be largely fulfilled before needs of a higher level arise.

Soft Factors: This concept comprises an ill-defined group of factors that are related to people, organizations, and environments like motivation, morale, organizational culture, power, politics, feelings, perceptions of environment, and so forth.

Software Component: A (software) component is (Bachmann et al., 2000):

- an opaque implementation of functionality
- subject to third-party composition
- in conformance with a component model

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software and (2) the study of approaches as in (1) (Abran, Moore, Bourque, Dupuis & Tripp, 2004).

This work was previously published in Encyclopedia of Human Computer Interaction, edited by C. Ghaoui, pp. 414-421, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 2.5
Multimodal Modeling, Analysis, and Validation of Open Source Software Development Processes*

Walt Scacchi
University of California, Irvine, USA

Chris Jensen
University of California, Irvine, USA

John Noll
Santa Clara University, USA

Margaret Elliott
University of California, Irvine, USA

ABSTRACT

Understanding the context, structure, activities, and content of software development processes found in practice has been and remains a challenging problem. In the world of free/open source software development (F/OSSD), discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. Prior studies have revealed that development processes in F/OSSD projects are different in a number of ways. In this article, we describe how a variety of modeling perspectives and techniques are used to elicit, analyze, and validate software development processes found in F/OSSD projects, with examples drawn from studies of the software requirements process found in the NetBeans.org project.

INTRODUCTION

In the world of globally dispersed, free/open source software development (F/OSSD), discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. For

example, in our studies of software requirements engineering processes in F/OSSD projects across domains like Internet infrastructure, astrophysics, networked computer games, and software design systems (Scacchi, 2002, 2004, 2005), we generally find there are no explicit software requirements specifications or documents. However, we readily find numerous examples of sustained, successful, and apparently high-quality F/OSS systems being deployed on a worldwide basis. Thus, the process of software requirements engineering in F/OSSD projects must be different than the standard model of requirements elicitation, specification, modeling, analysis, communication, and management (Nuseibeh & Easterbrook, 2000). But if the process is different, how is it different, or more directly, how can we best observe and discover the context, structure, activities, and content software requirements processes in F/OSSD projects? This is the question addressed here.

Our approach to answering this question uses multimodal modeling of the observed processes, artifacts, and other evidence composed as an ethnographic hypermedia that provides a set of informal and formal models of the software development processes we observe, codify, and document. Why? First, our research question spans two realms of activity in software engineering, namely, software development and software process modeling. So we will need to address multiple perspectives or viewpoints, yet provide a traceable basis of evidence and analysis that supports model validation. Second, given there are already thousands of self-declared F/OSSD projects affiliated with OSS portals like SourceForge.net, Freshmeat.net, and Savannah.gnu.org, then our answer will be constrained and limited in scope to the particular F/OSSD projects examined. Producing a more generalized model of the F/OSS development process studied requires multiple, comparative project case studies, so our approach should be compatible with such a goal (Scacchi, 2002). Last, we want an approach to process modeling that is open to independent analysis, validation, communication, and evolution, yet be traceable to the source data materials that serve as evidence of the discovered process in the F/OSSD projects examined (cf. Kitchenham, Dyba, & Jorgensen, 2004).

Accordingly, to reveal how we use our proposed multimodal approach to model requirements processes in F/OSSD projects, we first review related research to provide the foundational basis for our approach. Second, we describe and provide examples of the modeling modes we use to elicit and analyze the processes under study. Last, we examine what each modeling mode is good for, and what kind of analysis and reasoning it supports.

RELATED RESEARCH AND APPROACH

There is growing recognition that software requirements engineering can effectively incorporate multiple viewpoints (Finkelstein, Gabbay, Hunter, & Nuseibeh, 1994; Leite & Freeman, 1991; Nuseibeh & Easterbrook, 2000) and ethnographic techniques (Nuseibeh & Easterbrook, 2000; Viller & Sommerville, 2000) for eliciting, analyzing, and validating functional and nonfunctional software system *product* requirements. However, it appears that many in the software engineering community treat the *process* of requirements engineering as transparent and prescriptive, though perhaps difficult to practice successfully. However, we do not know how large distributed F/OSSD projects perform their development processes (cf. Curtis, Krasner, & Iscoe, 1998).

Initial studies of requirements development across multiple types of F/OSSD projects (Scacchi, 2002, 2004) find that OSS product requirements are continuously emerging (Gans, Jarke, Kethers, & Lakemeyer, 2003; Gasser, Scacchi, Penne, & Sandusky, 2003; Truex, Baskerville, & Klein, 1999) and asserted after they have been implemented, rather than relatively stable and

elicited before being implemented. Similarly, these findings reveal requirements practice centers about reading and writing many types of communications and development artifacts as "informalisms" (Scacchi, 2002), as well as addressing new kinds of nonfunctional requirements like project community development, freedom of expression and choice, and ease of information space navigation. Elsewhere, there is widespread recognition that F/OSSD projects differ from their traditional software engineering counterparts in that F/OSSD projects do not in general operate under the constraints of budget, schedule, and project management constraints. In addition, OSS developers are also end users or administrators of the software products they develop, rather than conventionally separated as developers vs. users. Consequently, it appears that F/OSSD projects create different types of software requirements using a different kind of requirements engineering process, than compared to what the software engineering community has addressed. Thus, there is a fundamental need to discover and understand the process of requirements development in different types of F/OSSD projects.

We need an appropriate mix of concepts, techniques, and tools to discover and understand F/OSSD processes. We and others have found that process ethnographies must be empirically grounded, evidence-based, and subject to comparative, multiperspective analysis (Curtis et al., 1998; Finkelstein et al., 1994; Glaser & Strauss, 1967; Kitchenham et al., 2004; Nuseibeh & Easterbrook, 2000; Scacchi, 2002; Seaman, 1999). However, we also recognize that our effort to discover and understand F/OSSD processes should reveal the experience of software development newcomers who want to join and figure out how things get done in the project (Scacchi, 2005).

As participant observers in such a project, we find that it is common practice for newcomers to navigate and browse the project's Web site, development artifacts, and computer-mediated communication systems (e.g., discussion forums, online chat, project Wikis), as well as to download and try out the current software product release. Such traversal and engagement with multiple types of hyperlinked information provide a basis for making modest contributions (e.g., bug reports) before more substantial contributions (code patches, new modules) are offered, with the eventual possibility of proposing, changing, or sustaining the OSS system's architecture. These interactive experiences reflect a progressive validation of a participant's understanding of current F/OSSD process and product requirements (Bolchini & Paolini, 2004; Narayanan & Hegarty, 2002). Thus, we seek a process discovery and modeling scheme that elicits, analyzes, and validates multimode, hypertext descriptions of a F/OSSD project's requirements process. Furthermore, the process descriptions we construct should span informal through formal process models, and accommodate graphic, textual, and computationally enactable process media. Finally, our results should be in a form open to independent analysis, validation, extension, and redistribution by the project's participants.

MULTIMODE PROCESS MODELING, ANALYSIS, AND VALIDATION USING ETHNOGRAPHIC HYPERMEDIA

An ethnographic hypermedia (Dicks & Mason, 1998) is a hypertext that supports comparative, cross-linked analysis of multiple types of qualitative ethnographic data (cf. Seaman, 1999). They are a kind of semantic hypertext used in coding, modeling, documenting, and explaining patterns of social interaction data and analysis arising in contemporary anthropological, sociological, and distributed cognition studies. The media can include discourse records, indigenous texts, interview transcripts, graphic or photographic images, audio/video recordings, and other related information artifacts. Ideally, they also preserve

the form and some of the context in which the data appear, which is important for subsequent (re)analysis, documentation, explanation, presentation, and validation.

Ethnographic studies of software development processes within Web-based F/OSSD projects are the focus here. Ethnographic studies that observe and explain social action through online participant observation and data collection have come to be called "virtual ethnography" (Hine, 2000). Virtual ethnography techniques have been used to observe the work practices, compare the artifacts produced, and discover the processes of F/OSSD projects found on and across the Web (Elliott & Scacchi, 2003, 2005; Jensen & Scacchi, 2005a, b; Oza, Nistor, Hu, Jensen, & Scacchi, 2004; Scacchi, 2002, 2004, 2005). In particular, an important source of data that is examined in such studies of F/OSSD projects is the interrelated web of online documents and artifacts that embody and characterize the medium and continuously emerging outcomes of F/OSSD work. These documents and artifacts constitute a particular narrative/textual genre ecology (Spinuzzi & Zachry, 2000) that situate the work practices and characterize the problem solving media found within F/OSSD projects.

We have employed ethnographic hypermedia in our virtual ethnographic studies of F/OSSD projects. What does this mean, and what challenges or opportunities for requirements elicitation, analysis, and validation have emerged along the way? These questions are addressed below through examples drawn from a case study of the NetBeans.org OSSD project (Jensen & Scacchi, 2005a, b), which is one of the largest F/OSSD projects we have studied. The NetBeans.org project is a corporate sponsored OSSD project (Jensen & Scacchi, 2005a) focused on the development of an interactive development environment (IDE) for constructing application systems using Java enterprise beans technology. It is similar in size and scope to the Eclipse project founded by IBM, which is also developing a Java-based IDE.

As noted, the F/OSSD projects we study are found on the Web. Web sites for these projects consist of a network of hyperlinked documents or artifacts. Samples of sites we have studied include NetBeans.org, Mozilla.org, Apache.org, and GNUenterprise.org, among others (Elliott & Scacchi, 2003, 2004; Jensen & Scacchi, 2005a, b; Scacchi, 2002, 2004). A team of two to five researchers examines a project site (via browsing, searching, downloading, and cross-linking) over a period of 4-6 weeks initially, then periodically thereafter. The artifacts we examine include Web pages, e-mail discussion lists, bug reports, project to-do lists, source code files and directories, site maps, and more. These artifacts are an important part of the data we collect, examine, study, code, and analyze in order to identify F/OSSD work practices and development processes that arise in a given project.

We create a hypermedia of these artifacts in ways that allow us to locate the originating sources of data within the focal project's Web site. This allows us to maintain links to the source data materials that we observe as evidence of the process at hand, as well as to allow us to detect when these data sources have been updated or removed. (We also archive a local copy of all such data.) However, we create annotated and assembled artifacts that embed hyperlinks into these documents as part of our ethnographic hypermedia. As a result, multiple kinds of ethnographic records are created including annotated artifacts, rich hypermedia pictures, and ethnographic narratives. Juxtaposed about these records are other kinds of models including a process metamodel, attributed directed graph model, process domain ontology, and a formal, computationally enactable process model. Each is described next, and each is hyperlinked into an overall ethnographic hypermedia that provides cross-cutting evidence for the observed OSS requirements processes. This in turn provide us with an approach to mapping complex situations in various forms, which is consistent with recent

advances in grounded theory approaches to ethnographic study (Clarke 2003).

Annotated Artifacts

Annotated artifacts represent original software development artifacts like (publicly available) online chat transcripts that record the dialogue, discussions, and debate that emerge between OSS developers. These artifacts record basic design rationale in an online conversation form. The textual content of these artifacts can be tagged, analyzed, hyperlinked, and categorized manually or automatically (Rao, 2003). However, these conversational contents also reveal much about how OSS developers interact at a distance to articulate, debate, and refine the continuously emerging requirements for the software system they are developing. For example, Elliott and Scacchi (2003, 2005) provide conversational transcripts among developers engaged in a debate over what the most important properties of software development tools and components to use when building free software. They provide annotations that identify and bracket how ideological beliefs, social values, and community building norms constrain and ultimately determine the technical choices for what tools to use and what components to reuse when developing OSS. The following is an example of an excerpt of an online chat transcript found in a F/OSSD project where the developer (here identified anonymously as "ByronC") who is an outsider to the project lurking on the chat discussion and who advocates a strong belief for avoiding the use of nonfree software when developing F/OSS, as indicated by the (**boldface**) annotations we added.

<ByronC> Hello **(Outsider Critique-1**
<ByronC> Several images on the Web site seem to be made with nonfree Adobe software, I hope I'm wrong: it is quite shocking. Does anybody know more on the subject?
<ByronC> We should avoid using nonfree software at all costs, am I wrong? **(Extreme belief in free software (BIFS)-1)**
<ByronC> Anyone awake in here? **Outsider Critique-1)**

Basic ethnographic data like this draws our attention to look for practices within F/OSSD efforts to see if such beliefs do in fact constrain the choice of software tools used within F/OSSD processes.

Navigational Rich Pictures

Rich pictures (Monk & Howard, 1998) provide an informal graphical scheme for identifying and modeling stakeholders, their concerns and beliefs, objects and patterns of interaction. We extend this scheme to form navigational rich pictures constructed as a Web-compatible hypertext image map that denotes the overall context as the composition and relationships observed among the stakeholder roles, activities, tools, and document types (resources) found in a F/OSSD project. In the example figures that follow, we display the stakeholders/roles using human-like icons, their concerns or beliefs as clouds associated to the icons, and the objects and patterns or interaction as hyperlinked entities. Figure 1 displays such a rich picture constructed for NetBeans.org. Furthermore, associated with each hyperlinked entity is a *use case* (Cockburn, 2001) that we have constructed to denote an observable activity performed by an actor-role using a tool that consumes or produces a document/object type. An example use case is shown in Figure 2. Each other type of data also is hyperlinked to either a descriptive annotation or to a Web site/page where further information on the object type can be found.

Directed Resource Flow Graph

A directed resource flow graph denotes a recurring workflow pattern that has been discovered

Figure 1. A rich picture image map of the requirements and release process in the NetBeans.org F/OSSD project

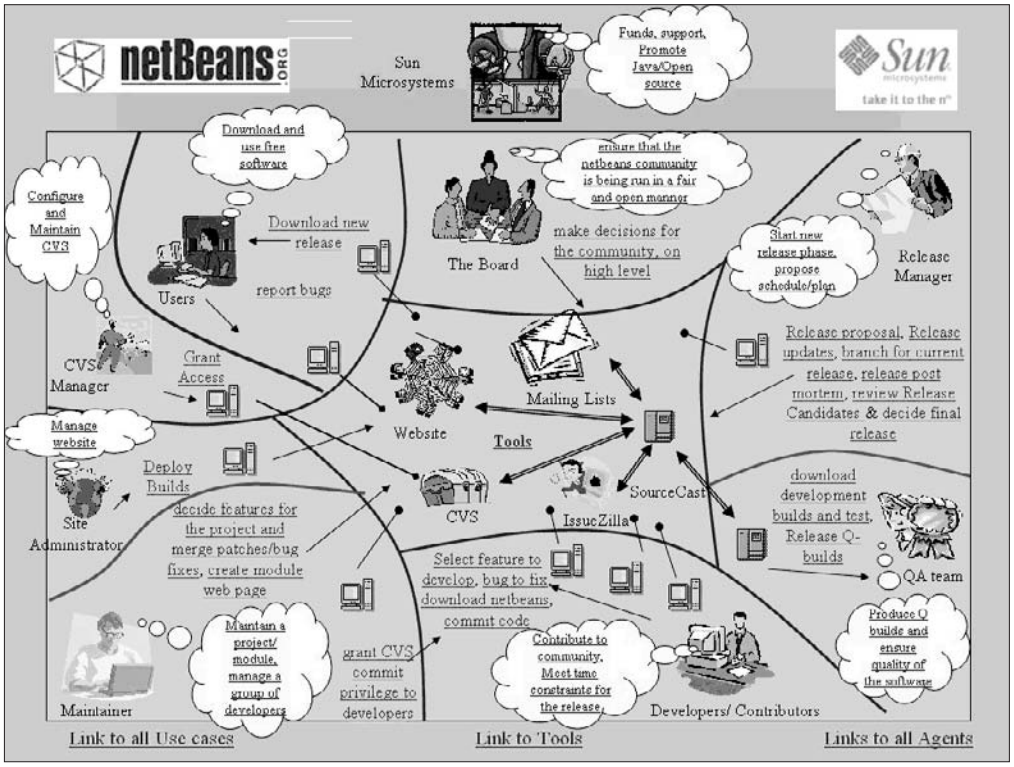
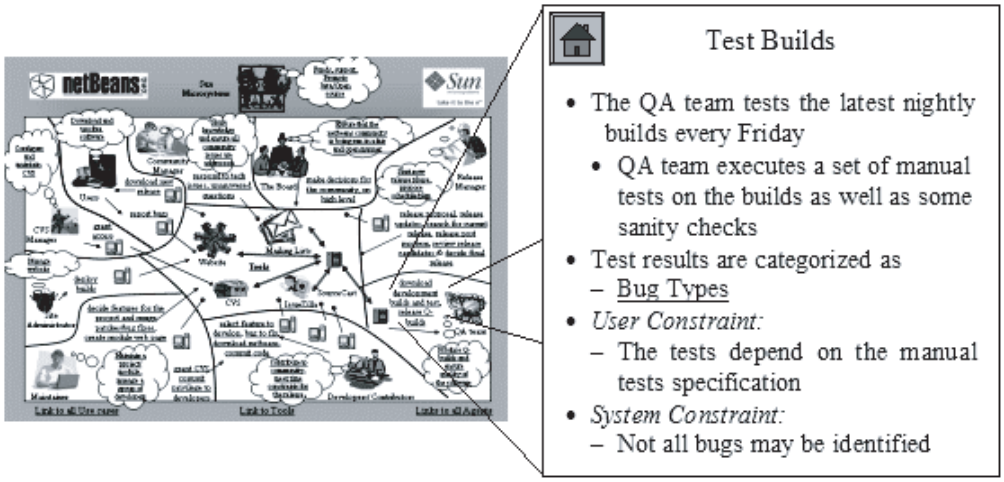


Figure 2. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case

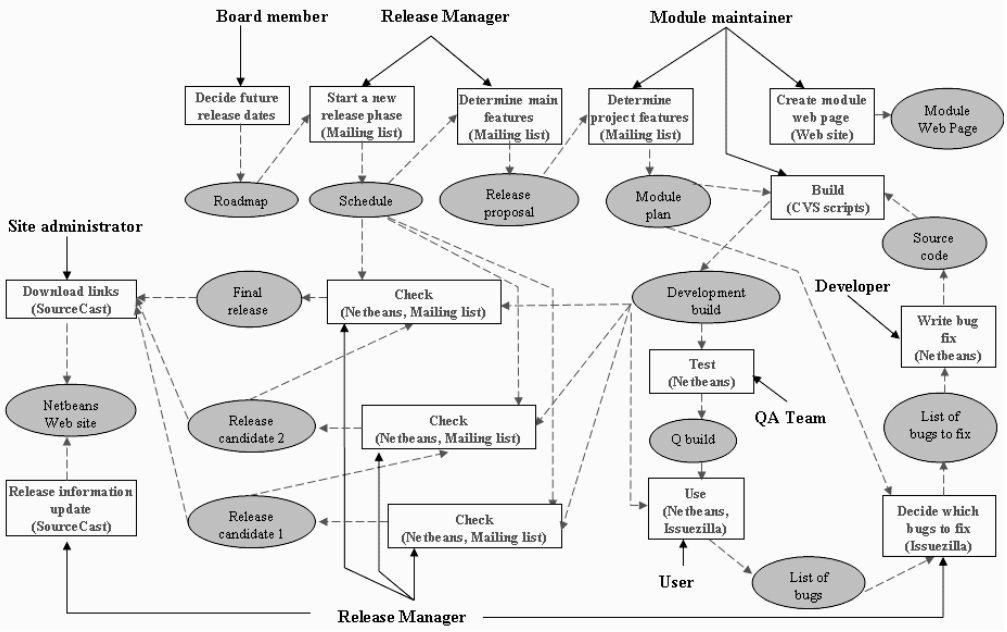


in an F/OSSD project. These workflows order the dependencies among the activities that actor-roles perform on a recurring basis to the objects/resources within their project work. These resources appear as or within Web pages on an F/OSSD project's Web site. For example, in the NetBeans.org project, we found that software product requirements are intertwined with software build and release management. Thus, the "requirements and release process" entails identifying and programming new/updated system functions or features in the course of compiling, integrating, testing, and progressively releasing a stable composition of source code files as an executable software build version for evaluation or use by other NetBeans.org developers (Elliott & Scacchi, 2003, 2005; Oza et al., 2004). An example flow graph for this appears in Figure 3. The code files, executable software, updated directories, and associated e-mail postings announcing the completion and posting the results of the testing are among the types of resources that are involved. Last, the rendering of the flow graph can serve as an *image map* to the online (i.e., on the NetBeans.org Web site) data sources from where they are observed.

Process Domain Ontology

A process ontology represents the underlying *process metamodel* (Mi & Scacchi, 1996; Noll & Scacchi, 2001) that defines the semantics and syntax of the process modeling constructs we use to model discovered processes. It provides the base object classes for constructing the requirements process (domain) taxonomies of the object classes for all of the resource and relation types found in the rich picture and directed resource flow graph.

Figure 3. An attributed directed graph of the resource flow for the NetBeans.org requirement and release process. Boxes denote tasks/actions, ellipses denote resources/objects, dashed lines denote resource flows, and solid lines and labels denote agent/stakeholder roles performing tasks that transform input resources into output resources.



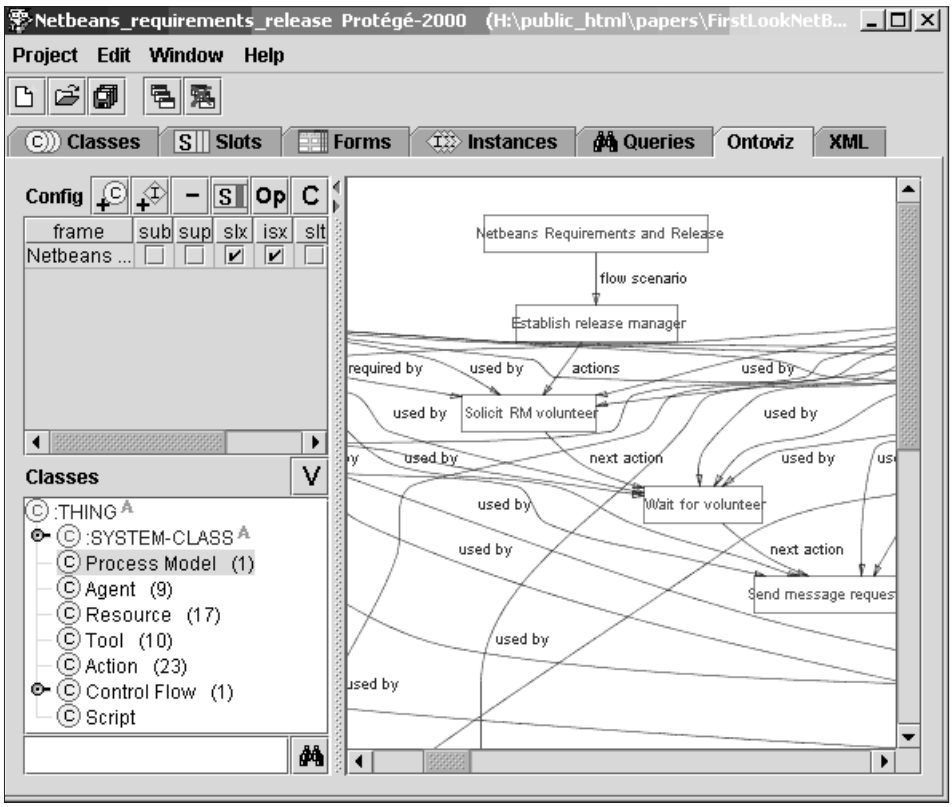
However, each discovered process is specific to an F/OSSD project, and knowledge about this domain is also needed to help contextualize the possible meanings of the processes being modeled. This means that a process domain entails objects, resources or relations that may or may not be have been previously observed and modeled, so that it may be necessary to extend to process modeling constructs to accommodate new types of objects, resources, and relations, as well as the attributes and (instance) values that characterize them, and attached methods that operationalize them.

We use an ontology modeling and editing tool, Protégé-2000 (Noy, Sintek, Decker, Crubezy, Fergerson, & Musen, 2001), to maintain and update our domain ontology for OSS requirements processes. Using Protégé-2000, we can also visualize the structure of dependencies and relations (Grinter, 2003) among the objects or resources in a semantic Web manner. An example view can be seen in Figure 4. Furthermore, we can create translators that can transform syntactic form of the modeling representations into XML forms or SQL schema definitions, which enables further process modeling and tool integration options (cf. Jensen & Scacchi, 2005b).

Formal Process Model and Its Enactment

A formal process model denotes a syntactically precise and semantically typed specification of the resource objects, flow dependencies, actor-roles, and associated tools that specifies an enactable

Figure 4. A view of the process domain ontology for the NetBeans.org software requirements and release process



(via interactive process-guided user navigation) hypertext representation we call an *organizational process hypertext* (Noll & Scacchi, 2001). This semantic hypertext, and its supporting run-time environment, enables the ability to walkthrough or simulate enactment of the modeled F/OSSD process as a process-guided, navigational traversal across a set of process linked Web pages. The semantic hypertext is automatically rendered through compilation of the process models that are output from the ontology editor in a process modeling language called PML (Noll & Scacchi, 2001). A PML-based model specification can be automatically checked for inconsistencies can be detected at compile-time or run-time. An example of an excerpt from such a model is shown in Figure 5. The compiled version of the PML produced a nonlinear sequence of process-linked Web pages, each one of which corresponds to one step in the modeled process. An example showing the result of enacting a process (action) step specified at the bottom of Figure 5 appears in Figure 6.

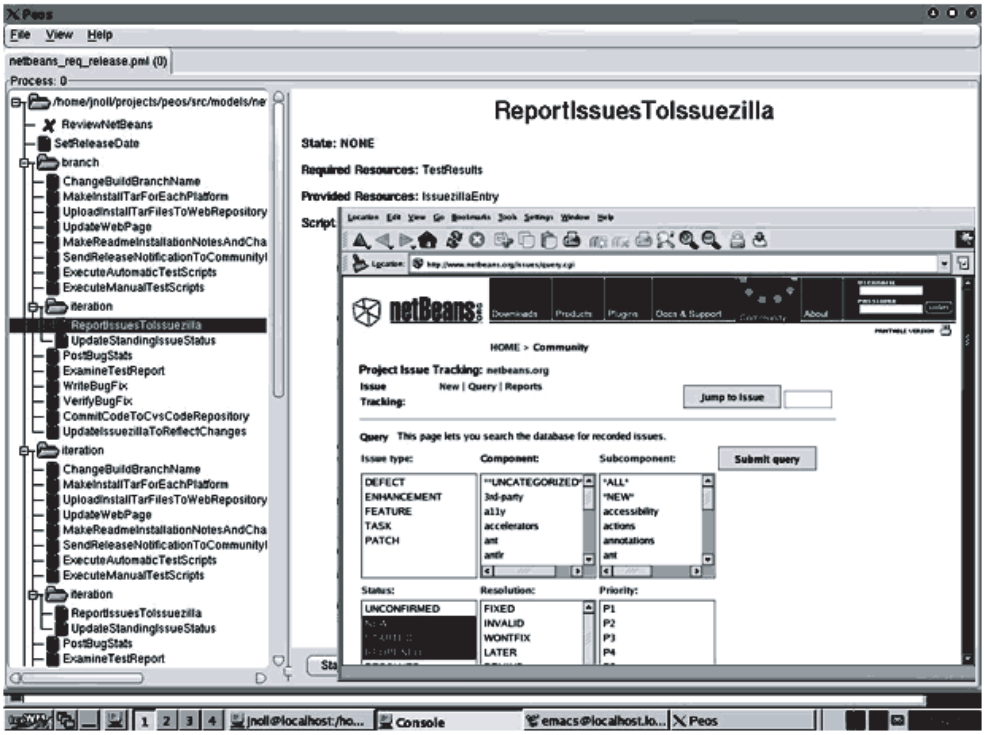
Constructing an Ethnographic Hypermedia Narrative for Process Validation

An ethnographic narrative denotes a final ethnographic hypermedia view. This is an analytical research narrative that is structured as a docu-

Figure 5. An excerpt of the formal model of the Netbeans.org requirements and release process coded in PML

```
...
sequence Test {
 action Execute automatic test scripts {
 requires { Test scripts, release binaries }
 provides { Test results }
 tool { Automated test suite (xtest, others) }
 agent { Sun ONE Studio QA team }
 script { /* Executed off-site */ } }
action Execute manual test scripts {
 requires { Release binaries }
 provides { Test results }
 tool { NetBeans IDE }
 agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio
developers }
 script { /* Executed off-site */ } }
iteration Update Issuezilla {
 action Report issues to Issuezilla {
  requires { Test results }
  provides { Issuezilla entry }
  tool { Web browser }
  agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio
developers }
  script {
```

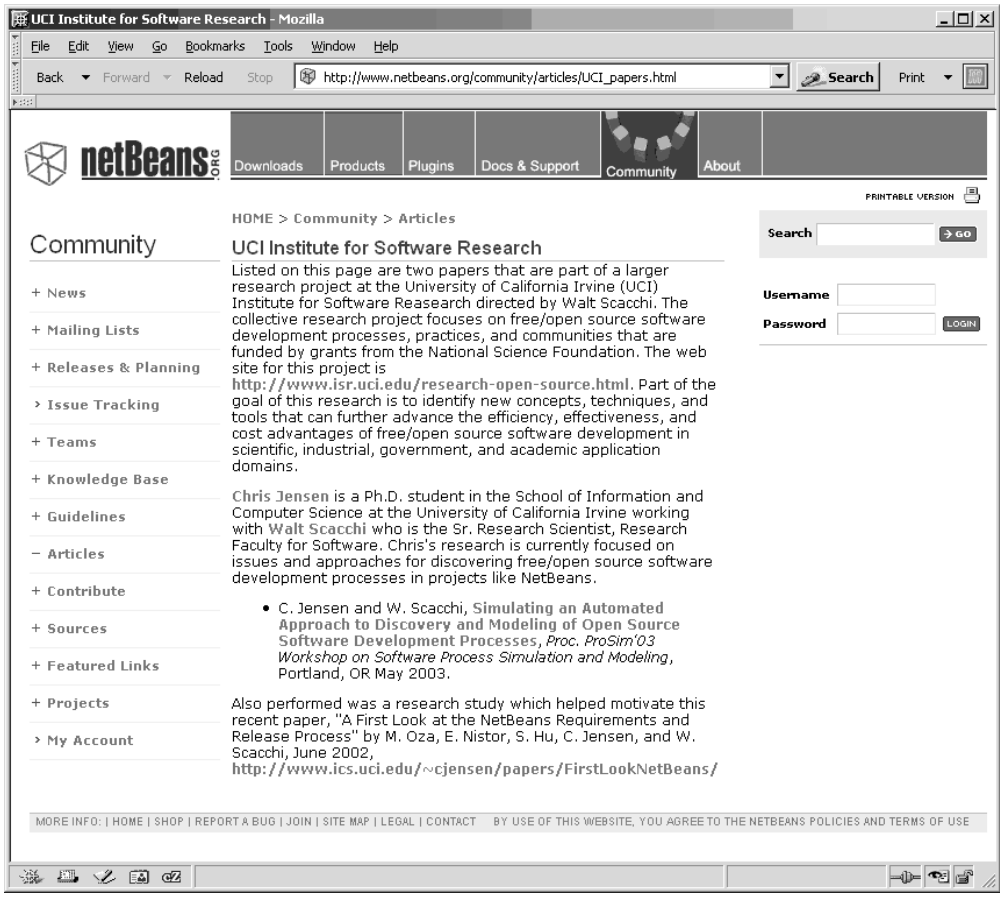
Figure 6. A screenshot displaying the result of the PML-based reenactment of one step ("Action Report issues to Issuezilla") in the NetBeans.org requirements and release process



ment that is (ideally) suitable for dissemination and publication in Web-based and printed forms. It is a composite model derived from selections of the preceding representations in the form of a narrative with embedded hyperlinked objects, and hyperlinks to related materials. It embodies and explains the work practices, development processes, resource types and relations, and overall project context as a narrative, hyperlinked ethnographic account that discovered at play within a given F/OSSD project, such as we documented for the NetBeans requirements and release process (Oza et al., 2004). In printed form, the narratives we have produced so far are somewhere between one-fourth to one-fifteenth the number of pages compared to the overall set of project-specific data (documents) at the first two levels of hyperlink connectivity; said differently, if the ethnographic report is 30 or so printed pages (i.e., suitable for journal publication), the underlying ethnographic hypermedia will correspond to a hypermedia equivalent to 120-450 printed pages.

The narrative is in a form intended for external review and validation by those not involved in the collection, modeling, and analysis activities, such as members of the project under study (NetBeans.org; see Figure 7). These external reviewers can read through the narrative during validation to see if there are gaps or inconsistencies, or to pose questions to the narrative's authors. When such shortfalls are detected or reported, then the task is to determine if the problem arises from either a gap in the modeling effort, or in its narrative rendering. Finally, the narrative and its hypermedia components are envisioned as open and living documents, so that feedback from the community may serve to keep them consistent with current practice, or to detect and report inconsistencies

Figure 7. Getting captured and analyzed process models out for validation and possible evolution by NetBeans.org project participants



that are in need of attention, update, or remediation, much like the software and artifacts found in the F/OSSD projects they describe.

DISCUSSION

We have learned a number of things based on applying and evaluating our approach to modeling development processes, such as those for software requirements, in different F/OSSD projects. First, no single mode of process description adequately subsumes the others, so there is no best process description scheme. Instead, different informal and formal descriptions respectively account for the shortcomings in the other, as do textual, graphic, and computationally enactable process representations. Second, incremental and progressive elicitation, analysis, and validation occur in the course of developing multimode requirements process models. Third, multimode process models are well-suited for discovery and understanding of complex software processes found in F/OSSD projects. However, it may not be a suitable approach for other software projects that do not organize, discuss, and perform software development activities in an online, persistent, open, free, and publicly accessible manner. Fourth,

multimode process modeling has the potential to be applicable to the discovery and modeling of software product requirements, although the motivation for investing such effort may not be clear or easily justified. Process discovery is a different kind of problem than product development, so different kinds of approaches are likely to be most effective.

Next, we observed that the software product requirements in F/OSSD projects are continually emerging and evolving. Thus, it seems likely that the requirements process in such projects is also continuous. Thus, supporting the evolution of multimode models of OSS requirements processes will require either automated techniques for process discovery and multimode update propagation techniques, or else the participation of the project community to treat these models as *open source software process models*, that can be continuously elicited, analyzed, and validated along with other F/OSSD project assets, as suggested in Figure 7, which are concepts we are currently investigating. However, it seems fair to note that ethnographic accounts are situated in time, and are not intended for evolution.

Last, each of the methods we used for modeling processes found in F/OSSD projects has been previously applied and shown to be useful by other researchers. Our effort brings these diverse approaches together in order to demonstrate and compare their individual and collective value. Thus, we find the multimode approach to modeling, analyzing, and validating F/OSSD processes provides a new threshold for research and practice that in turn give rise to new insights and findings that none of the individual approaches can realize on their own. Finally, we believe the multimode approach can be readily adopted, taught, and put into practices to support not only F/OSSD projects, but any software development project that is distributed and supported across the Web/Internet.

CONCLUSION

Ethnographic hypermedia are an important type of semantic hypertext that are well-suited to support the navigation, elicitation, modeling, analysis, and report writing found in ethnographic studies of F/OSSD processes. We have described our approach to developing and using ethnographic hypermedia to support the modeling, analysis, and validation of software development processes in F/OSSD projects like NetBeans.org, where multiple modes of informal to formal representations are involved. We find that this hypermedia is well-suited for supporting qualitative research methods that associated different type of project data, together with comparative analysis of process descriptions rendered in graphic, textual, and computationally enactable descriptions. We provided examples of the various kinds of hypertext-based process descriptions and linkages that we constructed in moving from abstract, informal representations of the data through a series of ever more formalized process models resulting from our studies.

Based on our efforts and results reported here, it appears that free/open source software development projects can benefit from the discovery, modeling, and validation of the development processes they practice, and that ethnographic hypermedia based representations of these processes provides an innovative scheme for capturing, representing, and evolving these representations in a manner that can be maintained and evolved in an open source way.

ACKNOWLEDGMENT

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 from the U.S. National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; and

others at ISR are collaborators on the research described in this article.

REFERENCES

Bolchini, D., & Paolini, P. (2004). Goal-driven requirements analysis for hypermedia-intensive Web applications. *Requirements Engineering, 9*, 85-103.

Clarke, A.E. (2003). Situational analysis: Grounded theory mapping after the postmodern turn. *Symbolic Interaction, 26*(4), 553-576.

Cockburn, A. (2001). *Writing effective use cases.* New York: Addison-Wesley.

Curtis, B., Krasner, H., & Iscoe, N. (1998). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Dicks, B., & Mason, B. (1998). Hypermedia and ethnography: Reflections on the construction of a research approach. *Sociological Research Online, 3*(3). Retrieved June 9, 2006, from http://www.socresonline.org.uk

Elliott, M., & Scacchi, W. (2003, November). Free software developers as an occupational community: Resolving conflicts and fostering collaboration. In *Proceedings of the ACM International Conference on Supporting Group Work* (pp. 21-30), Sanibel Island, Florida.

Elliott, M., & Scacchi, W. (2005). Free software development: Cooperation and conflict in a virtual organizational culture. In S. Koch (Ed.), *Free/open source software development* (pp. 152-172). Hershey, PA: Idea Group Publishing.

Finkelstein, A.C.W., Gabbay, D., Hunter, A., & Nuseibeh, B. (1994). Inconsistency handling in multi-perspective specifications. *IEEE Transactions on Software Engineering, 20*(8), 569-578.

Gans, G., Jarke, M., Kethers, S., & Lakemeyer, G. (2003). Continuous requirements management for organisation networks: A (dis)trust-based approach. *Requirements Engineering, 8*, 4-22.

Gasser, L., Scacchi, W., Penne, B., & Sandusky, R. (2003, December). Understanding continuous design in OSS projects. In *Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications*, Paris, France.

Glaser, B., & Strauss, A. (1967). *The discovery of grounded theory: Strategies for qualitative research.* Chicago: Aldine Publishing Co.

Grinter, R.E. (2003). Recomposition: Coordinating a web of software dependencies. *Computer Supported Cooperative Work, 12*(3), 297-327.

Hine, C. (2000). *Virtual ethnography.* Newbury Park, CA: Sage Publications.

Jensen, C., & Scacchi, W. (2005a, January). Collaboration, leadership, control, and conflict management in the NetBeans.org community. In *Proceedings of the 38th Hawaii International Conference on Systems Science*, Waikola Village, Hawaii.

Jensen, C., & Scacchi, W. (2005b, July-September). Process modeling across the Web information infrastructure. *Software Process--Improvement and Practice, 10*(3), 255-272.

Kitchenham, B.A., Dyba, T., & Jorgensen, M. (2004). Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering* (pp. 273-281), Edinburgh, Scotland. IEEE Computer Society.

Leite, J.C.S.P., & Freeman, P.A. (1991). Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering, 17*(12), 1253-1269.

Mi, P., & Scacchi, W. (1996). A meta-model for formulating knowledge-based models of software

development. *Decision Support Systems, 17*(4), 313-330.

Monk, A., & Howard, S. (1998, March-April). The rich picture: A tool for reasoning about work context. *Interactions, 5*(2), 21-30.

Narayanan, N.H., & Hegarty, M. (2002). Multimedia design for communication of dynamic information. *International Journal on Human-Computer Studies, 57*, 279-315.

Noll, J., & Scacchi, W. (2001). Specifying process-oriented hypertext for organizational computing. *Journal of Network & Computer Applications, 24*(1), 39-61.

Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., & Musen, M.A. (2001, March-April). Creating semantic Web contents with Protégé-2000. *IEEE Intelligent Systems, 16*(2), 60-71.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. In A. Finkelstein (Ed.), *The future of software engineering.* ACM and IEEE Computer Society Press.

Oza, M., Nistor, E., Hu, S. Jensen, C., & Scacchi, W. (2004, February). A first look at the Netbeans requirements and release process. Retrieved June 9, 2006, from *http://www.ics.uci.edu/cjensen/papers/FirstLookNetBeans/*

Rao, R. (2003, November). From unstructured data to actionable intelligence. *IT Pro,* pp. 29-35.

Scacchi, W. (2002, February). Understanding the requirements for developing open source software systems. *IEE Proceedings on Software, 149*(1), 24-39.

Scacchi, W. (2004, January). Free/open source software development practices in the computer game community. *IEEE Software, 21*(1), 59-67.

Scacchi, W. (2005). Socio-technical interaction networks in free/open source software development processes. In S.T. Acuña & N. Juristo (Eds.), *Peopleware and the software process (*pp. 1-27*).* World Scientific Press.

Seaman, C.B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering, 25*(4), 557-572.

Spinuzzi, C., & Zachry, M. (2000, August). Genre ecologies: An open system approach to understanding and constructing documentation. *ACM Journal of Computer Documentation, 24*(3), 169-181.

Truex, D., Baskerville, R., & Klein, H. (1999). Growing systems in an emergent organization. *Communications of the ACM, 42*(8), 117-123.

Viller, S., & Sommerville, I. (2000). Ethnographically informed analysis for software engineers. *International Journal Human-Computer Studies, 53*, 169-196.

ENDNOTE

* Previous version appeared in *Proc. First Intern. Conf. Open Source Software,* Genova, Italy, 1-8, July 2005.

This work was previously published in International Journal of Information Technology and Web Engineering, Vol. 1, Issue 3, edited by E. Damiani; G. Succi, pp.49-63, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 2.6
Conceptual Model Driven Software Development (CMDSD) as a Catalyst Methodology for Building Sound Semantic Web Frameworks

Thomas Biskup
Carl von Ossietzky University, Oldenburg, Germany

Nils Heyer
Carl von Ossietzky University, Oldenburg, Germany

Jorge Marx Gómez
Carl von Ossietzky University, Oldenburg, Germany

ABSTRACT

This Chapter introduces Hyperservices as a unified application model for Semantic Web frameworks, and proposes Conceptual Model-Driven Software Development as a means of easy adoption to them. Hyperservices are based on agent societies, provided with structured information by the Semantic Web, and using Web services as a collaboration and communication interface. Afterwards, the WASP model is proposed as a framework for implementing Hyperservices, also adding personalization rules to modify the agents' perception as well as the HIVE Architecture as Semantic Information Server infrastructure within the WASP framework. For easier adoption of these new models, Conceptual Model-Driven Software Development is proposed. It separates the conceptual aspects from the technical details by automatically generating executable code from models while the implementation details are hidden to the end user, the service developer.

OVERVIEW

The Semantic Web and its effects are a mainstream catalyst for current Web development. Its influence is felt across many areas of research and business development: Agent systems, knowledge management frameworks, ontology definitions, and other areas are all refined by new ideas from Semantic Web research (and vice versa). Since many complex topics are now combined with the goal of building the "Next Generation Internet", it becomes more and more important to build sound and flexible frameworks to abstract the implementation details of the underlying technologies.

As underlying technologies are still in a state of flux as far as their implementation details are concerned, it seems to be very important to find a simple yet appropriate meta-model for the overall architecture which can be used to follow a kind of model-driven approach: Model the required system in a meta-level and then derive the actual implementation by transforming the model into executable code (or even directly executing the model). This approach allows both the early adoption of Semantic Web technologies and a continuing evolution of the implementation details.

Research shows that the underlying methodology for defining Semantic Web-oriented frameworks can be defined very well. This chapter will explain the main streams which will be integrated towards the Semantic Web and more importantly show, based on a thorough requirements analysis, how Semantic Web-oriented systems might be structured in a simple meta-model, allowing more detailed specification as research progresses. A new software development methodology, named Conceptual Model-Driven Software Development or CMDSD for short, which is currently under development in our research team, is used to provide a notion of the appropriate meta-models which will allow the early adoption of Semantic Web technologies in standard industrial projects.

The following steps in this chapter will lead to an early-stage meta-model which might be used to connect Semantic Web frameworks in an easy and non-intrusive way with standard projects:

- The main research streams and technologies making up the Semantic Web are identified. Their interrelations and resulting requirements for frameworks and systems are shown.
- An in-depth requirements analysis concerning the architecture of Semantic Web systems and the must-have features of such features provides the groundwork for the definition of the cornerstones of future Semantic Web systems. It will be shown that the basic cornerstones are limited in scope, thus making it possible to define a very simple high-level meta-model for a model-driven strategy.
- An approach to build multi-platform Semantic Web frameworks based on the core technologies of Agents, Ontologies, Web Services, and Personalization frameworks is explained. This approach is generic enough to encompass most currently-existing frameworks and lends itself towards the integration of emerging standards. A new type of service, a Hyperservice, is derived from integrating these core technologies into a new type of service infrastructure.
- An overview of Model-Driven Architecture (MDA) and Model-Driven Software Development (MDSD) will be given. It provides the infrastructure for our extension of CMDSD (Conceptual Model-Driven Software Development) which strives to close the gap between technology expertise and conceptual requirements by building meta-models focused on the conceptual task and defining a transformation path to build complex systems from simple meta-models.

EMERGING SEMANTIC WEB TECHNOLOGIES REVIEWED

First we will give an overview of existing Semantic Web research and describe the current underlying problems which need to be solved in the new future to let the Semantic Web become a living reality: The main task will be to narrow or close the gap between the reality of the Web (a disjoined and tangled mass of loosely-coupled information resources) and the vision for the Web: a tightly-integrated and openly-structured information network with machine-readable data that allows autonomous agencies to create new applications empowered by this wealth of information. Currently problems already start in the early stages of developing systems for the Semantic Web: Existing models and theories are relatively complex (compared to established Web standards like XML and HTML), frameworks still have a very dynamic state, and interfaces between frameworks are bound to change more often than not (if they exist at all at this point of time). We propose a framework to allow researchers and developers to choose the level of detail, the type of technologies, and the amount of computing power they want to utilize for their proposed solutions. We will use the building blocks of this framework as the grounding for a flexible yet very simple meta-model that ignores technical complexity and favors the easy integration of existing technologies. We focus for our framework on a flexible abstraction layer, pattern-oriented architecture, and open interfaces to build on the successful foundations of the Web: ease of use, flexibility, and almost unlimited expression power. Agents are the central paradigm for software development using this architecture.

The Evolution of the Web

The Semantic Web is pushed by the World Wide Web Consortium (W3C) as the foundation for a true information society (Berners-Lee, 1998; Berners-Lee, Hendler, & Lassila, 2001). The efforts of the W3C are supported by a wide range of multi-national research efforts combining theoretical and practical experiences from information technology.

Nonetheless progress is slow, and even if research would yield results at much greater speed, the results still need to be implemented. Current research hints at much more expressive and thus also more powerful means to represent data and information, but the price is added complexity required to build the representations. We therefore focus on trying to provide simple models which completely hide the underlying complexity of Semantic Web technologies. The World Wide Web was successful because people basically overnight were enabled to share information, with simple technology. This allowed for the enormous growth in information resources which we now face. We strive to reproduce this pattern to guarantee the further growth of the Web (Berners-Lee, 2000) by providing an early example for a simple meta-model (with ample extension points) at the end of this chapter.

The Web of Systems

The World Wide Web in its current form is the largest information system ever built by humans. At the same time, it probably is also one of the least structured information systems ever built. There are billions of Web pages (not counting other resources like images, videos, sounds, CGI interfaces to large databases, and more), and almost none of them are structured in a standardized way. The "Deep Web" is even larger (Bergman, n.d.). Most pages are built with HTML and coupled in a very loose manner; links lead into oblivion as often as they do not. Most existing links do not provide much semantic information (e.g., what is the meaning of a specific link except "someone thought that two information resources should be connected in some way"). Most information is presented in a way that allows humans to use

it, although access to this information usually is a problem because it becomes harder and harder to find the few tidbits of information in the existing mess of data. Studies by search engine companies show that current problems are to be found at a more basic level: Even the simple requirements of HTML are rarely correctly fulfilled and semantic information provision is not a topic high on the list of most interest groups present on the World Wide Web (Web Authoring Statistics, n.d.).

Thus we argue that we need to find ways to evolve from the current World Wide Web (a Web of Systems, so named because there are many individual systems that usually are only connected by the simplest means, namely hyperlinks) to something more.

It would be foolish and dangerous to try too much at once. At the same time, it would be as foolish and dangerous to create artificial boundaries and introduce building blocks that limit our power of expressiveness. Thus we propose to search for architectures and frameworks that support gradual evolution without limiting the final goal. We find practical examples that support the viability of this approach: Modular programming has spawned object-oriented programming to be able to control complexity with more natural concepts. For certain problem areas, agent-oriented systems have been discovered to be an immensely powerful and very natural concept for defining solutions (Ciancarini & Wooldridge, 2001). Now the industry momentum offers a huge chance to solve one of the basic problems of agent societies: Communication by Web services promises to do away with the artificial system boundaries currently inhibiting large-scale distributed autonomous agent systems. And finally, model-driven approaches (either MDA or MDSD) (Stahl & Völter, 2005) seem to be highly appropriate to serve as the glue between the evolving and varying technologies.

The Web of Services

Web Services currently are the preferred integration technology for business software companies. Web Services (Christensen, Curbera, Meredith, & Weerawarana, 2001; Gottschalk, Graham, Kreger, & Snell, 2002) in theory offer a standard means to communicate between disparate systems and applications with absolute disregard for programming languages, computer hardware, and system-specific communication protocols. Based on XML (eXtensible Mark-up Language) (Biskup & Marx Gómez, 2005), this new and exciting standard promises a new way of defining interfaces, without sticking to implementation details and basic technical questions. Together with HTTP (Hypertext Transfer Protocol) (Gourley & Totty, 2002) and SOAP (Simple Object Access Protocol) (Mitra, 2001) as protocols, we face an enormous opportunity to bring together previously-separated building blocks for the next generation Internet. XML is the unifying data representation standard that could be used to basically encode any kind of information. HTTP and SOAP are simple yet flexible protocols that allow a system-independent communication between heterogeneous systems. More generally, the basic notion of application-level protocols is very important to be able to conceptualize a working architecture for a true Web of Services. While it currently is very difficult to connect different Web-based systems, future interfaces could greatly benefit from these standards (e.g., when trying to combine a flight information system and a hotel booking system).

While theoretically already very strong in practice, many Web Services standards are not yet available. The specifications currently evolve at a much faster pace than the implementations, and many of the core technologies (e.g., distributed transactions and security) are still in a state of continuing change. These movements will stabilize eventually, but in the meantime the permanent flux of changes creates investment risks which

cause many commercial endeavors to use as little as possible of the new technologies. In turn, this causes specifications to be created with a certain lack of practical experience, which in turn shows in the quality of early specification versions and again hampers adoption.

If models can be provided that collect the basic ideas of the standard and completely hide the implementation details, companies could start today with integrating the existing standards. A model-driven approach seems to be the most appropriate way for this task: The conceptual tasks could be modeled on a meta-level by using a simple and abstracted modeling technology, and some kind of transformation engine would take these models and either directly execute them or transform them into executable code. In this way systems could be easily upgraded to new versions of specific Web Service technologies, as the meta-models will probably only change in minor ways or not at all.

A Web of Services thus could become a tangible possibility. This could be the next important step for Web technologies—because Web services possess many powerful features ideally suited for industrial use and commercial success stories. This also could build the momentum to ensure the wide-spread use of, in our point of view, a very important technology. Current developments support this theory; most new API versions and programming systems supply some sort of Web Services integration (from ancient languages like COBOL to the most recent developments like .NET).

The Web of Semantics

All afore-mentioned efforts target one underlying and ever present goal: the Semantic Web, an information network of machine-readable data that allows autonomous agencies to gather data, turn it into information, reason about it, and come to conclusions. This information network will be traversed by intelligent agents to fuel new and exciting services (Joy, 2000; McIlraith, Son, & Zeng, 2001; Metcalfe, 2000). Humans will never be able to fully utilize the mass of data collected in the World Wide Web; thus we need to find new ways to turn all the data into something more than a loosely connected set of HTML pages. The basic building blocks for the Semantic Web are made up by:

- **Semi-structured data:** XML has been accepted as the means of choice to represent platform-independent data in a semi-structured way that allows for an open-ended way of describing data (Bray, Paoli, Maler, Sperberg-McQueen, & Paoli, 2000). Based on plain text (but powered by Unicode), XML enriches pure data with metadata to allow machines to use the data more effectively and in ways not initially coded into the data format.
- **Machine-readable data:** The current proposal for this building block relies on XML as a means of expression and has been named RDF (Resource Description Framework) (Brickley & Guha, 2000; Lassila, 2000). It should be noted that RDF has various means of representation, but XML seems to be the most natural for the World Wide Web and the most widely used right now. RDF allows describing resources, properties of resources, and relations between resources. RDF can be extended to create more complicated languages and at the same time provides powerful foundations for reasoning (being based on first-order logic). Interestingly, RDF takes a very pragmatic approach to provide a viable solution for information representation: Right away, it allows for inconsistence, incorrectness, and incompleteness in the represented information and takes it as given that data can lead to situations where agents will not be able to come to a decisive or correct conclusion. This pragmatism adheres to the concepts

that established the current Web, ease of use with an allowance for mistakes.

- **Ontologies** as a means to describe the relations between objects and to define standard hierarchies as descriptions of "the world": A lot of research is concerned with the question of what should be in an ontology language in order to once more find the best way of combining computing and expression power with ease of use. Ontology languages like SHOE (Heflin, Hendler, & Luke, 1999), DAML (Hendler & McGuiness, 2000), and DAML+OIL (DAML+OIL, 2000) hint at the power of future metadata structures.

So far major concerns in the World Wide Web community were to standardize the encoding of data and information. Retrieval, automated reasoning about information, connection of services, and basically all other means of exploiting this information pool were only moderately successful. The Web spawned a variety of search engines and meta-search engines, but these, together with shop systems and Web directories, cover the efficient means of access to the World Wide Web for humans. There were some experiments with agents and agent societies (Brickley & Guha, 2000), but so far these attempts failed to become wide-spread successes due to the lack of a unified information infrastructure and lack of standardized interfaces; CGI (the Common Gateway Interface) is hardly sufficient to build even semi-complex applications in an abstract and elegant way. Other experiments (De Bruijn, Fensel, Keller, & Lara, 2005; Harper & Bechhofer, 2005) hint at different exciting possibilities to enhance the knowledge acquisition process for users, but still lack the unified foundation required to build a whole generation of such service enhancements. The need for such a foundation is proven by some extensive first-generation systems (Lin, Harding, & Teoh, 2005)), which show the basic building blocks that will be required again and again, and thus are the primary target for standardization attempts.

To cope with this situation we propose a new model of regarding future applications building on the foundations mentioned so far, a unit of abstraction we have named Hyperservices.

Proposing Hyperservices as a Unified Application Model

We believe that the next important step will be to find a unifying, language- and system-independent architecture that allows for a convergence in current research areas surrounding the Semantic Web. When we talk about Web-based applications, we mean "based on Web technologies". Web technologies have been widely accepted and have managed to bring together disparate system structures. While this goal in the first moment might be very ambitious, we will present a very simple (one might think trivial) way of allowing all these technologies to be integrated into current run-time environments. Ample extension points will be provided, and we refrain from requiring any specific implementation technologies.

Looking at the components currently available, a unified application model based on agent societies seems to be in reach: The Semantic Web allows us to reason about information by structuring information appropriately. This provides the basis for "intelligent" agents (with "intelligence" on a pragmatic hands-on level). Web services introduce the interface for collaboration among systems. Agents are the natural extension to achieve autonomous systems (Benjamins, 2003). Currently we face a multitude of ontology languages (Benjamins, Fensel, & Asunción, 1998) and many models and theories to map information to efficient data models and retrieval algorithms, but these means will only see wide-spread use if they become easy to communicate to future users (e.g., programmers), based on standard architectures and easy to integrate into existing systems. Integration still is one of the main problems faced by current computer science (from the business perspective), but the Web

can only remain successful if it manages to stay commercially interesting (whether by drawing enough people to it to supply Internet Providers with customers, or by introducing truly successful e-business models is not that important). Thus the integration of these new models into existing structures will be the most important task from a business point of view.

Topics under current discussion (e.g., agent societies and the Semantic Web) will not be able to replace classic information systems (e.g., tax accounting, enterprise resource planning, and logistics). But if built in the right way, they will be able to enrich classic systems by providing added value. They will open up a new venue of information systems, built around the necessity to decide between precision and speed. The sheer size of the Web and its constant flux will make it impossible to store nearly enough data in local systems to allow for efficient information systems (in the classic sense). Thus it seems much more likely that future information systems will be build around the idea of semi-autonomous agents wandering across the Web, collecting information, reasoning about it, and yielding results, either continuously or until specified resource limits (e.g., time, bandwidth, or a financial budget) have been exhausted (Hendler, 2001).

The WASP Model

We propose a unified framework that is founded on four building blocks which, in our point of view, will be absolute necessities to populate the future Web with more powerful applications:

- *W*eb Services as a means of providing a unified communication interfaces between applications and agencies (Christensen et al., 2001; Dale, 2002; UDDI, 2001);
- *A*gents as a natural and central means to represent typical tasks and solutions for a distributed and constantly changing information environment;
- *S*emantic Web technologies as a means to provide data and information in a consistent manner that allows retrieval and reasoning; and
- *P*ersonalization technologies to customize processes to the needs of the individual user, an absolute necessary concerning the current (and future) size of the World Wide Web, lest it becomes impossible to separate useless from useful information.

(The initials of these core technologies also provide the acronym for our framework: WASP)

Agents will be the central building block of this architecture, because they implement the actual business logic. Web Services are the natural means of communication and collaboration for agents working under the described model; the Semantic Web is the environment (world) for these agents, and the personalization rules basically can be used to make up or modify the beliefs of the agents. Thus the described components integrate very nicely and in a very natural manner into the underlying agent paradigm.

The WASP framework will account for a variety of necessities explained in the next sections. In contrast to existing major endeavors in this area (Finin, Labrou, & Mayfield, 1997; Object Management Group, n.d.; O'Brien & Nicol, 1998), we plan to provide an architecture that focuses on:

- proactive information agents that collect information and provide results by using inference mechanism to reason about the existing information;
- high-level technical support for the application developer (e.g., communication, distribution, data storage);
- tight integration of Web technologies (RDF, Web Services, DAML, SOAP, etc.);
- independence from specific kinds of implementations (e.g., no specific communication language will be enforced); and

- focus on agents relying on the Semantic Web as the dominant information source.

Thus the following central paradigms will be of greatest importance:

Open Interfaces

Since it is impossible to enforce one true operating system, one true programming language, or one true CPU architecture for the network underlying the World Wide Web, it is of paramount importance to provide a powerful means of communication between the interacting agencies. SOAP and HTTP (as the underlying protocols), together with Web Services (as a means of interface specification), seem to be natural choices. The framework will provide a layer of abstraction to be able to disconnect from these particular technologies, should, for example, other protocols become more important.

Service Agencies

Agents seem to be a very natural way for describing typical scenarios of Web usage. They are the machine representation of human beings who right now have to do most of the work manually. Thus the WASP framework will provide means to define a variety of agents: mobile, autonomous, reactive, and so forth. To enhance the usefulness of the framework, it is set up to allow agents to be self-describing, thus automatically turning agents into services that can be used by others and integrated via standard interfaces. This allows for wide-spread service dissemination and easy integration with other systems.

It will be especially important to integrate current agent research into this framework layer; efforts like DAML (DARPA Agent Mark-up Language) allow for powerful modeling means to devise agents and agencies.

Data and Information Gathering

The framework must provide for means to accumulate data, compare it, and reason about it. Data might be persistent (to allow for agents with increasing reasoning power) or transient (to model short-effect tasks), and data should be interchangeable between different agents. It must be possible to integrate ontologies to allow for a solidified view of the "world" (in regards to the agent or agents).

Personalization Integration

It must be easy to integrate personalization technologies. At the most basic level, it should be possible to specify user preferences and dislikes, and to integrate them in the reasoning and retrieval process to improve the quality of the returned information.

The HIVE: Semantic Web Brokering Simplified for WASP Agents

Web servers have been the technical foundation for the success of the World Wide Web. Applications servers have been a successful model in abstracting from the everyday chores of building complex applications and thus form the basis for modern large-scale business applications. Thus it seems natural to evolve to Semantic Information Servers that provide a corresponding environment for Semantic Web agents specialized on utilizing the Semantic Web resources to provide information services to the end user.

Application servers offer persistence, transactions, distributed processing, and scalability if the software complies with a predefined component model (e.g., Java 2 Enterprise Edition / J2EE). This allows developers to focus on the actual task at hand, for example, implementing the business logic for a complex transaction portal. In our view, a similar model is required for Semantic Web applications based on agent societies. Different layers of abstractions will allow concentrating on

functional requirements and help to abstract from the details of the implementation. In the same way that a J2EE application server takes away the details of persistence from the developer, a Semantic Information Server can abstract from the details of, for example, storing semantic information, retrieving it, and reasoning about it. This holds true for other areas as well (e.g., information recovery from the Web, resource management for agents, and communication between members of local and remote agencies). Within the WASP framework, we intend to call the Semantic Information Servers a HIVE (not an acronym but rather a play of words continuing the WASP idea).

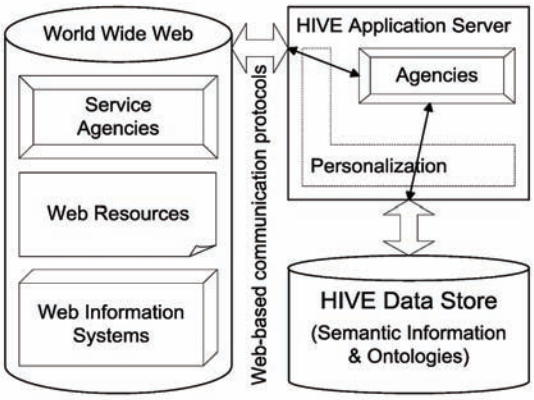
These ideas result in the infrastructure diagram in Figure 1.

Important Semantic Web Research Areas

In this section we intend to describe a few of the more important current research topics needing to be solved to further the development of Semantic Web services:

- Ontology integration and translation is a major problem for interconnecting distributed services and systems (Gruber, 1993; Heflin & Hendler, 2000; Heflin et al., 1999): How can differing ontologies for related topics be mapped on each other?
- Web Service orchestration, interoperation, and transaction handling needs to be standardized (UDDI, 2001; Web Services Choreography Requirements, 2003; Web Services Transaction, 2002).
- Standards to allow for personalization need to find wide acceptance (PICS, n.d.; Stahl & Völter, 2005; for discussions on currently-available, yet still rarely-used standards).

Figure 1. HIVE architecture



Remaining Challenges

Besides the technical questions which currently enjoy most attention, a multitude of additional topics needs to be investigated before distributed agent systems and the Semantic Web become truly viable. A few of them are:

- **Modeling challenges:** In which way can Semantic Web systems be modeled so that a broad user base will be able to understand and utilize these technologies? We make an early attempt to answer this question in the last major part of this chapter.
- **Cost challenges:** Who is going to pay for the resources being used in distributed agent networks? It is safe to assume that such agent services will be a lot more cost-intensive than the "simple Web information platforms of today" (e.g., Web servers).
- **Pricing challenges:** Already now there is a tendency to commercialize high-quality services. How will future information systems be rated in terms of usage fees if the component services of some complex service (e.g., the logistics service, the currency conversion service, and the mapping service for a complex online order service) each incur fees, but the user of the complex service might not necessarily add to the income of the complex service provider (e.g., because

the user decides against buying something after getting the shipment information)?
- **Business challenges:** What are viable business models for Semantic Web agencies and services?
- **Quality challenges:** How will services be able to guarantee a certain level of quality if they rely on the data collected in the Semantic Web, an information storage that will be as inaccurate as the currently-available World Wide Web (mostly because everyone will be able to put up whatever he or she deems correct and appropriate)?
- **Trust challenges:** How can I be sure that not only the quality of results gained by Semantic Web analysis is sufficient for me but also correct at all?
- **Workflow challenges:** How can complex workflows (like booking a complete holiday trip) be orchestrated when dynamic service directories, user preferences, potentially faulty information, and other factors need to be considered?
- **Performance challenges:** How must services be constructed to be able to retrieve useful data in a timely manner from the Semantic Web, a Web that is infinitely more complex to search compared to current search engines and technologies due to the far more involved complexity created by allowing inferences, conclusions, and reasoning about information?
- **Security challenges:** How can personal and private information be protected from prying eyes? What new security challenges arise from the architectural decisions made for the WASP framework?
- **Legal challenges:** Who will be held responsible for incorrect, imprecise, or faulty information derived from or modified by Semantic Web content?
- **Architectural challenges:** What are the best-of-breed software infrastructures/ application architectures to allow for a rapid dissemination of the technologies involved? How can user and developer acceptance be increased?

REQUIREMENTS ANALYSIS CONCERNING SEMANTIC WEB ARCHITECTURES

As explained in our example Semantic Web architecture, different technology layers usually will be found in a Semantic Web-based system. In this section, we will continue the example by analyzing the essential requirements to be fulfilled both by our similar architectures in order to be Semantic Web-ready. We will start at the bottom-most layer (the database tier) and work upwards from there. Differences compared to standard enterprise architectures will be explained in the individual sections.

Requirements for the HIVE Data Store

The HIVE Data Store will store and evaluate data gained by analyzing Semantic Web resources. To provide a useful and extensible model, the following functional and technical requirements must be taken into account:

- The HIVE Data Store should be independent from the specific type of storage (DBMS, RDF store, flat file, some kind of network service). Thus it will be possible to choose the best type of storage for a given situation. Additionally this is a basic requirement to be able to exchange data store implementations as technology and research continue to evolve.
- The HIVE Data Store must not assume that data is correct, complete, or unambiguous. The Internet by design is a place where data is provided in a spontaneous and improvised manner. Thus the Data Store must be able

to cope with such data. This is also a major difference from classical enterprise systems, where usually the utmost care is taken to insert only verified, correct, and unambiguous data into databases.

- The HIVE Data Store must provide inference support. The true value of the Semantic Web can be used only by analyzing the gathered data and drawing conclusions. Thus inference support is paramount. Nonetheless there must not be any assumptions about the specific inference approaches being used, again to allow flexibility and configurability.
- The HIVE Data Store must be able to access data from a variety of sources. This is absolutely necessary due to the size and varied nature of the underlying information sources. The Internet itself is just too large to be kept on one server or a whole farm of servers; while in most cases an application will not need to access all the information available on the whole of the Internet, for some of the more exciting Semantic Web applications it will be a necessity to be at least potentially able to access all available information.
- The HIVE Data Store must be able to integrate ontologies into its repository. Ontologies are the basic mechanism to define the meaning of concepts modeled in, for example, RDF structures. An important add-on functionality will be the ability to compare ontologies and map them onto each other to be able to integrate differing ontologies designed for the same problem area.
- The HIVE Data Store must include a facility to manage the credibility of sensitive information. Mission-critical data must only be accepted from sources that are able to authenticate themselves and prove their credibility (e.g., with certificates or similar mechanisms).
- The HIVE Data Store should be able to organize itself physically, to manage its resources, and to restructure information based on dynamic changes of the environment. This is an optional but highly recommended functionality, as it is to be expected that the Data Store of a widely-used system will grow with leaps and bounds. To remain efficient and to conserve resources, the Data Store itself has to take responsibility for this.
- The HIVE Data Store must explicitly consider data retention periods, along with models and algorithms for purging; otherwise, the data cached there will rapidly become stale and will overload the database.
- The HIVE Data Store must provide a management facility so that external users can examine the state of the server, the data, and the rules accumulated in the store. Additionally, typical functionality must be provided to configure the data sources and control resource allocation of the store. It should be noted that the management facility might have a very different outlook depending on the underlying storage mechanism being used for the specific server.

To be able to incorporate these widely varying requirements, an abstraction layer will be required through which all HIVE Data Store access operations will be routed. This will add functionality as required (e.g., by defining a HIVE Data Store proxy that collects requests, runs through a caching layer to increase performance, and then delegates unanswered requests to the actual physical or distributed Data Store).

Requirements for the HIVE Agent Server

The HIVE Application Server is responsible for running the various agents that implement the business logic side of the framework and access

all other components to achieve their goals. The following functional and technical requirements need to be taken into account:

- The HIVE Agent Server must provide a runtime environment for varying agents that share a common interface, as defined by the WASP framework. There must be support for both mobile and static agents to allow for a wide range of application scenarios.
- The HIVE Agent Server must provide a security layer that controls resource access for all agents operating on the server. This is an absolute necessity for servers incorporating mobile agents and allowing mobile agents to migrate from one server to another.
- The HIVE Agent Server must provide access to the HIVE Data Store in an abstracted and generalized manner. Agents must not be required to know about the details of data storage or inference.
- The HIVE Agent Server must provide access to the "outside world" so that the sensors of the agents involved can operate. The HIVE Agent Server may modify perceived data based on personalization and security rules.
- The HIVE Agent Server must allow for communication using a variety of agent communication languages (ACLs) to be able to integrate a variety of agent systems. This probably will include the necessity to provide translation services between different agent communication languages in order to allow communication between agents of different breeds.
- The HIVE Agent Server must provide a management facility so that external users can examine the state of the server and the agents running on it. The management facility must include all kinds of standard management functionality to allocate resources, control permission, direct agents and so on.
- The WASP Agent Interface must be structured so that existing agent frameworks can be easily integrated into the system while there also must be enough room to develop new concepts and architectures. Additionally, the interface must be extensible, so that incremental extensions may be made both forward and backward compatibly, allowing upgrades to clients or servers in either order.

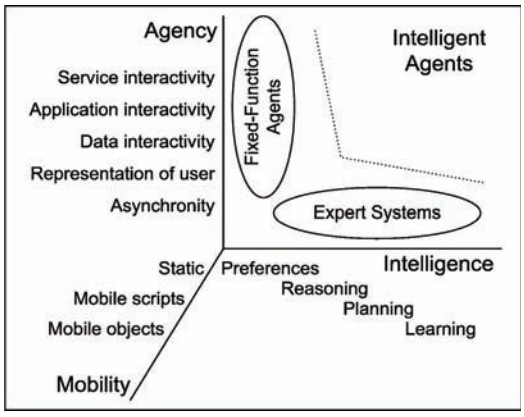
Interestingly, the requirements concerning the actual agents are very small; when examining, for example, the proposed scope of software agents in Gilbert, Aparicio, Atkinson, Brady, Ciccarino, Grosof, et al. (1995), the only functional requirements that must be provided by the HIVE Agent Server is mobility. Everything related to the intelligence aspects of agents can be modeled in the HIVE Data Store and the agent implementation. All other aspects are determined by the actual agent implementation.

Requirements for the HIVE Personalization Layer

The HIVE Personalization layer modifies both the perceived external environment (e.g., the Internet and other servers) and the perceived internal environment (especially the information gathered in the HIVE Data Stores). To provide a meaningful level of personalization, the following functionalities are of utmost importance:

- The HIVE Personalization Layer must provide permission-based personalization so that the perceived environment adjusts based upon the permissions of each individual agent. Permissions will have to be modeled on per-server-, per-agent- and per-user- base in order to be fully effective.
- The HIVE Personalization Layer must be able to handle agent-specific inference rules

Figure 2. Scope of intelligent agents (Adapted from Gilbert, et al., 1995)



in order to modify and control the inference process based on agent preferences.

- The HIVE Personalization Layer must be able to handle user-specific inference rules in order to modify and control the inference process based on agent preferences.

REQUIREMENTS FOR THE HIVE COMMUNICATION LAYER

The HIVE Communication Layer connects the HIVE Agents with external systems (e.g., the Internet, other servers, and so on).

The argumentation in the sections above shows that the HIVE Communication Layer also should serve another important purpose: The various components of the WASP Framework need to be connected in a system-independent and flexible way; by using the HIVE Communication Layer not only for inter-system but also for intra-system communication, several powerful synergy effects can be utilized:

- Communication uses but one protocol layer. This makes it much simpler to distribute agents, objects, and servers since no separate protocols are required for communication and information exchange. The implementation of the HIVE server itself is simplified, too.
- Intra-system services (e.g., resource management, process supervision, and more) can, if wanted, be modeled as agents to use the infrastructure provided by the WASP framework to implement the WASP framework itself. This is analogous to database systems that store database meta-information in their own database structures or compilers used to compile themselves, and serve as a good proof of concept for the simplicity and validity of the architecture, once implemented.
- The communication layer must integrate mechanisms to transparently handle both scalability of the system as a whole and to increase the fault tolerance when concerned with individual requests.

To be able to fulfill these expectations, a number of functional requirements must be considered:

- The HIVE Communication Layer must be implemented with a system-independent protocol and communication API. It must be possible to exchange atomic and complex information objects. Web Services and the associated communication protocols (e.g., SOAP) will serve nicely as a basis for the HIVE Communication Layer.
- The HIVE Communication Layer must be able to communicate with external systems through a standard protocol. Otherwise communication would be too limited for a reasonable system trying to support Semantic Web applications.
- The HIVE Communication Layer must provide some means to execute a lookup for external services and resources. Services like UDDI (Heflin & Hendler, 2000) and maybe also LDAP are typical candidates to be integrated. Although this might seem like a rather minor and trivial point at first

(especially given the fact that more or less established technologies like LDAP and UDDI already exist), it must be stressed that the integration of a powerful lookup service for other services and resources is of major importance, especially so when considering the stress on "fault tolerance" and "scalability".

Scalability is rather easy to achieve in some respects if a lookup mechanism exists: Requests for services, on one hand, can be dispatched to service endpoints based on a variety of load-balancing strategies and algorithms; only imagination limits the possible scenarios ranging from simple round-robin algorithms to elaborate agent-based schemes where agents monitor the CPU load on connected systems and use that information to control the dispatching of further incoming requests. Hardware-based load-balancing mechanisms can also be easily integrated as the endpoint of a service, and can represent a façade for a server cluster with a hardware load balancer handling the incoming requests. Naturally these techniques can also be combined.

To be able to effectively handle faults, several approaches can be implemented; the important concern is whether service faults on the actual service level (e.g., complete service failures due to unavailable servers, etc.) or on the logical level (e.g., service errors, etc.) are examined. The communication layer can easily enhance the fault tolerance of a Hyperservice-based system on the service level: A simple protocol handler extension can be integrated in order to allow the resubmission of service requests if a service endpoint fails to respond. This behavior can be enhanced by integrating a regular availability check into the service/resource registry to be able to filter out or deactivate temporarily-unavailable endpoints and thus reduce the overall network load. Errors on the logical level cannot be handled in a transparent level in the HIVE communication layer without extensive semantic descriptions of the services themselves. Since the idea of the HIVE architecture is to provide a simple and extensible architecture. For now, no logic fault handling at all will be integrated into the communication layer. Future research will have to show whether such extensions can be integrated in a simple yet effective manner.

Finally it is important to note that, as the internal HIVE services also can and should make use of the service and resource registries, the HIVE system itself transparently benefits from the integration of the features concerning fault handling and scalability as described above.

- The HIVE Communication Layer must be system- and programming language-independent. Again, Web Services and the associated protocols nicely fit this requirement.
- The HIVE Communication Layer must use standardized protocols to be able to connect to as many systems as possible.

Common Requirements

Finally there are some common requirements that are shared by all frameworks like, for example, compositionality, reusability and extensibility, that must be taken into account. Of special importance for the WASP framework is language independence (even if the reference implementation might use a specific language like Java or C#), in order to be able to connect as many systems as possible. Since these requirements are not unique or special to the presented topic, we will not elaborate them any further within the bounds of this chapter.

The Requirements Analysis in Review

In order to make the Semantic Web as accessible as possible, a highly-abstracted framework with open interfaces and simple overall concepts is

very important, especially since the underlying technologies mentioned above individually already require a high level of expertise to be able to use them effectively. We have shown that such an infrastructure framework is defined by a manageable amount of requirements, and that available standards (ready or just evolving) already cover a lot of ground. The next challenge is to provide a means of integration that is simple yet powerful and extensible at the same time. This will be the focus of the rest of this chapter.

MODEL-DRIVEN SOFTWARE DEVELOPMENT FOR THE SEMANTIC WEB

In this final section, we will try to give an answer on how to increase the acceptance of Semantic Web technologies by leveraging recent developments in the area of model-driven software development (MDSD). So far we have talked about how to simplify the development of Semantic Web-based systems, and this is the classical perspective of most approaches available today.

Now we would like to concentrate on a new way of approaching software development. Model-Driven Architecture (MDA) and Model-Driven Software Development (MDSD) both strive to simplify software development by automatically generating executable code from models (Stahl & Völter, 2005). These approaches seem to be well-suited for Semantic Web development considering the implied semantic definitions of models, agents, and ontologies, for example, provided by DAML. Nonetheless, one inherent weakness is left unconsidered by these approaches: Both MDA and MDSD require a sound understanding of the underlying technological and theoretical foundations, combined with a good amount of experience about how to model systems in the best way. This does not really help in transferring the conceptual knowledge that software customers possess into software systems as the developers still concentrate on building sound technological models and keep all the minute details under control.

We thus propose the next step in model-driven technologies, which we have named "Conceptual Model-Driven Software Development (CMDSD)".

The Notion of Conceptual Model-Driven Software Development (CMDSD)

The basic idea of our methodology of Conceptual Model-Driven Software Development (CMDSD) is to model systems through intuitive models on the conceptual level. These models must not be based on one global theory (like MDA with the Meta-Object Facility (MOF)), but rather utilize the most pragmatic theoretical foundation possible for the problem at hand. Additionally, the model should abstract as much as possible from the technological details of the system and focus on the conceptual facets which are possible. Models are aggregated into more complex models while great care is taken to provide a simple transformation from the individual model to the run-time system representation of the model. This allows software developers and architects to work together with customers and concentrate on the business side of the system; a simple and effective transformation from business problem to system representation is assumed. Existing design patterns support this notion.

Of greatest importance is the way in which CMDSD-based models are made available to the conceptual experts at hand: We require that models use a language (in the formal meaning) that comes as close as possible to the tools and ways used by the target audience to express their concepts. Thus the way of modeling will differ between areas of expertise. While this might feel like a disadvantage at first because software engineers will have to learn many differing modeling languages, we regard this as an advantage for a simple reason: The rules for transforming/execut-

ing these models need only be understood by a very small number of persons, namely those persons who build the actual transformers or interpreters. In contrast the user base for the concepts modeled typically will be very large in comparison. Thus it seems more reasonable to build models which are intuitive for the end user base and not the (intermediate) developers. Additionally it benefits the developers by forcing them to concentrate on the conceptual aspects of development and to hide the technical details.

One might argue that this is just a step back to the "model wars" in the years from 1990 to 2000 before the advent of UML. We argue that UML and similar OMG approaches did not really solve those problems as far as the actual specification of conceptual tasks goes. UML provides very effective models in order to specify technical details, but those technical details, most of the time, are of no interest at all to the conceptual experts who require a solution for their non-technical problems. The gap between available means to specify technical aspects and the lack of means to formally specify the purely conceptual parts of a problem leads to a high number of failing software projects (failing software projects are those that fail to meet their goals in time, money, or quality).

Current studies concerning success and failure of software projects seem to prove this point (Standish Group, n.d.): More than 75% of all failed software projects failed due to problems with requirements analysis and errors in the resulting specifications. We thus argue that more problem-oriented models are a must-have in order to facilitate the communication between technical and non-technical experts which work together on a complex project.

Additionally, our CMDSD methodology requires Rapid Prototyping (Reilly, 1996) as a means to be easily able to verify specifications.

Using CMDSD to Build Semantic Web Servers

How does all this help with Semantic Web development? Many of the underlying technologies for the Semantic Web are highly complicated and very involved. Rare is the architect or developer who understands agent technologies, Web service specifications (in their ever-growing numbers), and the details of building efficient application servers all at once. If architects and developers trained in more than one of these topics are already rare, how can we then assume that the end users for Semantic Web technologies (e.g., the people building Websites today) will be able to grasp the varying concepts and details of implementation?

As explained in the sections above, one of the research directions currently being actively investigated is the research in semantically-rich descriptions for Web services. Now imagine a development system that can understand existing semantic descriptions of services, load them, and generate definition and aggregation interfaces for such services. Suddenly developers and customers can work together in a semantically-rich environment to model the business side of systems and, by using a framework like the WASP framework, can mostly ignore the technological side as the underlying framework handles configuration and assembly.

While we still need to work on the actual implementation of a server that can handle arbitrary agent frameworks and expose them as Web Services, a specialized server for one type of agent system is within reach. Additionally, many projects will be able to be handled today by using but one technology, as the integration of differing agent systems requires quite a bit of research in the future. The next section will show how such a system can be integrated into existing application and Web servers with CMDSD methods.

A CMDSD-Based Approach for Agent and Service Integration

First of all, we specify the most important requirements that must be fulfilled in order to make it simple to use Semantic Web technologies. Please note that these definitions are flexible and based on our point of view; different focuses will yield different definitions. We intend to base our initial CMDSD-based model on the following requirements:

1. The model must be formulated in such a way that current Web developers will feel immediately comfortable using its syntax.
2. The model must hide the complexities of both the underlying technology and the integration questions popping up as soon as actual integration into an application or Web server comes to mind.
3. The model must allow for flexible data structures to both be sent to the agents and be received from the agents.
4. It must be possible to modify the data in both directions.
5. Standards should be used to implement the model transformation/executions.

To fulfill these requirements, we make the following decisions:

1. The model will be formulated in "an XML-like syntax similar to HTML" (see below). This will allow current Web developers to easily grasp the concepts. As a side effect, the model definitions for now will be included in the definitions for HTML pages. The special tags defined to describe WASP models will be shown below.
2. The model for now will only fulfill the most basic requirements of a basic IO system. Complex logic will be hidden in the underlying agents and processes. The model is thus only concerned with retrieving data from some source and sending (other) data to a sink (which provides the results as a response).
3. Technically we will assume that some kind of extension of the underlying server that analyzes the HTML code before it is sent to the client, extracts the WASP-specific definitions, processes them, and re-integrates the results of those definitions. Possible technical solutions could be Apache Plug-Ins (Bloom, 2002), Java Servlet Filters or Java Servlets (Bell, Loton, Allamaraju, Dalton, Brown, Harbourne-Thomas, et al., 2002) or ISAPI extensions for the Microsoft Internet Information Server (Genusa, Addison, & Clark, 1999).
4. XML will be used to represent data (Bray et al., 2000).
5. Simple transformations will be handled by XSL transformations (Cagle, Cornig, & Diamond, 2001). Complex data transformations will require specialized agents or Web services.
6. Services and agents will be represented by URNs (Uniform Resource Names) (URI Planning Interest Group, n.d.) in the models. The interpretation of those models will, for now, be server-specific, although future systems should strive to standardize URNs in the same way as XML namespaces.

Based on these requirements and definitions, we now will provide a simple example for a Hyperservice call. Note that in the HTML snippet in Figure 3, all WASP- (and model-) specific elements use the "wasp:" namespace.

The interpretation of the given code snippet is intuitive if you are even slightly versed in HTML; and now about the meaning of XSLT:

- The tag "<wasp:hyper service="urn:wasp:service:GetStockQuotes">" defines a hyperservice call. The service itself is identified by the given URN. The specifics of connecting

Figure 3.

```
…
<table width="100%">
<wasp:transform xslt="/xslt/
CreateRowsFromStockQuotes.xslt">
  <wasp:hyper service="urn:wasp:service:GetStockQuotes">
    <wasp:parameter name="verbosity" value="detailed"/>
    <wasp:parameter uses="stockMarketLocation"/>
  </wasp:hyper>
</wasp:transform>
</table>
…
```

to the service are completely hidden from the user (e.g., the Web page author) and must be configured in the server environment. A definition of the interface of that service also exists separately and is fulfilled implicitly by the next two lines of code (see below).

- Two parameters are defined:
  - The parameter "verbosity" is set to the value "detailed" which (intuitively) means that a detailed list of stock quotes is requested.
  - The parameter "stockMarketLocation" is defined dynamically (as indicated by the attribute "uses" instead of "value" as in the previous example); "uses" implies that the parameter value is taken from the request parameters. This allows defining parameters dynamically by, for example, building dynamic URLs or submitting parameters with a form.
- The result of the service request is transformed using XSLT as defined in the enclosing tag "<wasp:transform xslt="/xslt/CreateRowsFromStockQuotes.xslt">". This special tag applies an XSL transformation to the body of the tag (which in turn is produced by dynamically executing the service call explained above). Again an implicit knowledge about the result structure is assumed. Given a meaningful XSL transformation, the stock quote list (in XML format) will be transformed into a user-friendly and nicely formatted HTML representation of the data.

In order to allow for more complex structures, we will allow extended parameter definitions so that the results of calls can be, in turn, reused as parameters for other calls. Assuming the existence of a service to transform stock quote lists from XML format to HTML, we could rewrite the code above in the way Figure 4 shows.

In this revised example, the parameter named "list" is derived by calling the above-mentioned stock quote service and using the result of that service as the input for the second service. In a certain way, we pipe inputs and outputs in a comparable way to UNIX shell commands. The true power of these rather simple models becomes obvious when chaining several services (Figure 5).

In this example, three input parameters are processed: For a given user ("userID"), a destination ("destination"), and an indicated price limit ("maxPrice"), first the list of preferred airports is determined by calling one service, and then the available flights are searched by calling a second service, and combining both dynamic parameters and results from user interaction. In this way complex scenarios can be based on a very simple model.

Figure 4.

```
…
<table width="100%">
<wasp:hyper
service="urn:wasp:service:CreateRowsFromStockQuotes">
  <wasp:parameter name="list">
    <wasp:hyper service="urn:wasp:service:GetStockQuotes">
      <wasp:parameter name="verbosity" value="detailed"/>
      <wasp:parameter uses="stockMarketLocation"/>
    </wasp:hyper>
  </wasp:parameter>
</wasp:hyper>
</table>
…
```

Figure 5.

```
…
<table width="100%">
<wasp:hyper service="urn:wasp:service:FindFlights">
  <wasp:parameter name="airportList">
    <wasp:hyper
service="urn:wasp:service:FindPreferredAirports">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter name="benefits">
    <wasp:hyper
service="urn:wasp:service:FindPersonalBenefits">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter uses="destination"/>
  <wasp:parameter uses="maxPrice"/>
</wasp:hyper>
</table>
…
```

Also the level of abstraction is very high: Assume, for example, that all XML data structures are connected to an ontology. If service A requires input conforming to ontology o_A and receives its input from a service B which in turn is only able to produce results conforming to ontology o_B the runtime environment can (transparently for the model builder!) search for a transformation rule $o_A \rightarrow o_B$ and apply it, all this without requiring any in-depth knowledge of the "end user" (the HTML modeler/designer). Additionally, it will be possible to optimize the system by changing the configuration and not the model; the interpreter for the model will automatically take into account the new settings. Finally a change of technologies also is transparent for the "end user" (e.g., the page designer) as the special tags introduced above are independent of the implementation.

The final aspect of flexibility to be introduced within the context of this chapter is a way to an

actual service lookup. To allow for this, we now also allow the service URN to be provided dynamically (Figure 6).

Another simple extension of the model focuses on the "hyperlinked" nature of the Web: By not only accepting special URNs but generic URIs, it easily becomes possible to link separate systems into a more complex system. The following example illustrates this by using external URIs for some services from the previous example (Figure 7).

Again most of the underlying complexity is moved to the run-time environment which has to determine the actual service descriptions, match it against the expected interfaces, call the remote service with an appropriate protocol, accept the results, and so forth. Even details like session-based authentication might be handled transparently for the page designer by the underlying framework.

The formal semantics of the model are beyond the scope of this chapter but should be intuitive enough to derive for the interested reader.

Implementing the Proposed CMDSD Model

The most basic component required to implement the model proposed in the previous sections is the actual tag parser component. If, for example, a J2EE application server were used as the run-time environment, it would be a natural fit to implement the special WASP tags as a Tag Library (Brown, 2003) because tag libraries integrate in a very natural way into the run-time environment. The more complicated components concern the actual integration of services and agents. A way

Figure 6.

```
…
<table width="100%">
<wasp:hyper>
  <wasp:parameter name="service">
    <wasp:hyper service="urn:wasp:service:ServiceLookup">
      <wasp:parameter name="serviceName"
value="FindFlights"/>
      <wasp:parameter name="typeSpecialization"
value="lastMinute"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter name="airportList">
    <wasp:hyper
service="urn:wasp:service:FindPreferredAirports">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter name="benefits">
    <wasp:hyper
service="urn:wasp:service:FindPersonalBenefits">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter uses="destination"/>
  <wasp:parameter uses="maxPrice"/>
</wasp:hyper>
</table>
…
```

Figure 7.

```
…
<table width="100%">
<wasp:hyper>
  <wasp:parameter name="service">
    <wasp:hyper service="http://www.quinscape.de/services/
ServiceLookup">
      <wasp:parameter name="serviceName"
value="FindFlights"/>
      <wasp:parameter name="typeSpecialization"
value="lastMinute"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter name="airportList">
    <wasp:hyper service="https://www.foo.bar/
FindPreferredAirports">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter name="benefits">
    <wasp:hyper
service="urn:wasp:service:FindPersonalBenefits">
      <wasp:parameter uses="userID"/>
    </wasp:hyper>
  </wasp:parameter>
  <wasp:parameter uses="destination"/>
  <wasp:parameter uses="maxPrice"/>
</wasp:hyper>
</table>
…
```

needs to be found to describe service descriptions from agent specifications/implementations. An efficient system needs to be implemented to allow for ontology comparisons and transformations. Finally, a performance analysis of the implementation is a necessity in order to estimate how well the implementation scales under real-world conditions. Current research projects work on answering these questions.

SUMMARY AND CONCLUSION

In this chapter, we have shown how viable Semantic Web Frameworks can be developed with currently-existing technologies. We have underlined the importance of providing ways to allow for an easy adoption of Semantic Web technologies lest all efforts are moot due to not being accepted by the public. We have defined our notion of Hyperservices and presented the WASP architecture which facilitates the creation of extendable Semantic Web systems. We have analyzed the requirements for general Semantic Web frameworks and used the results thus gained in order to describe a conceptual model for our WASP framework. In order to facilitate these endeavors we have described our "Conceptual Model-Driven Software Development (CMDSD)" approach and given examples to illustrate how easy a model for accessing Hyperservices can be defined.

By providing such a foundation, we have disconnected undergoing research in the Semantic

Web area from the need to start making these technologies available for a wider audience.

BACKGROUND INFORMATION

This chapter is based on published and well-accepted papers by the authors as well as their continuing research in the areas of Semantic Web, Agent Systems, and Conceptual Model-Driven Software Development, among them Biskup, Marx Gómez, and Rautenstrauch (2005), Biskup and Marx Gómez (2004a, 2004b), and Biskup and Marx Gómez (2005).

REFERENCES

Bell, J., Loton, T., Allamaraju, S., Dalton, S., Brown, S., Harbourne-Thomas, A. et al. (2002). *Professional Java Servlets 2.3*. Birmingham, UK: Wrox Press Ltd.

Benjamins, R. (2003). *Agents and the semantic Web: A business perspective*. Paper presented at Agent Technology Conference (ATC, 2003), Barcelona, Spain.

Benjamins, V. R., Fensel, D., & Asunción, G. P. (1998). Knowledge management through ontologies. In *Proceedings of the second international conference on practical aspects of knowledge management (PAKM-98)* (pp. 5.1-5.12).

Bergman, M. K. (n.d.). *The deep Web: Surfacing hidden value*. Retrieved February 12, 2006, from http://citeseer.ist.psu.edu/bergman00deep.html

Berners-Lee, T. (1998). *Semantic Web Roadmap.* Retrieved February 12, 2006, from http://www.w3c.org/DesignIssues/Semantic.html

Berners-Lee, T. (2000). *Weaving the Web.* London: TEXERE Publishing ltd.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic Web. *Scientific American, May 2001 issue*. Retrieved February 20, 2006, from http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Biskup, T., & Marx Gómez, J. (2004a). Building blocks of a semantic Web framework—Requirements analysis and architectural implications. In *Proceedings of 3rd International Workshop on Web Semantics—WebS'04, in conjunction with the 14th International Conference on Database and Expert Systems Applications DEXA 2004—Zaragoza, Spain* (pp. 214-218).

Biskup, T., & Marx Gómez, J. (2004b). Component requirements for a universal semantic Web framework. *Semantic Web and Information Systems, AIS SIGSEMIS Bulletin, 1*(3) (October Issue), 25-28.

Biskup, T., & Marx Gómez, J. (2005). *Building a semantic Web framework with reusable and configurable core technologies.* Paper presented at IRMA 2005, San Diego, CA.

Biskup, T., Marx Gómez, J., & Rautenstrauch, C. (2005). The WASP framework—Bridging the gap between the Web of systems, the Web of services, and the Web of semantics with agent technology. *International Journal of Intelligent Information Technologies (IJIIT) 1*(2), 68-82.

Bloom, R. B. (2002). *Apache Server 2.0. The complete reference*. Berkeley, CA: Osborne McGraw-Hill.

Bray, T., Paoli, J., Maler, E., Sperberg-McQueen, C. M., & Paoli, J. (Ed.). (2000). *Extensible Markup Language (XML) 1.0. W3C Recommendation.* Retrieved February 12, 2006, from http://www.w3c.org/TR/REC-xml

Brickley, D., & Guha, R. V. (Ed.). (2000). *Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation.* Retrieved February 12, 2006, from http://www.w3c.org/TR/RDF-schema

Brown, S. (2003). *Professional JSP tag libraries.* Birmingham, UK: Wrox Press Ltd.

Cagle, K., Cornig, M., & Diamond, J. (2001). *Professional XSL*. Birmingham, UK: Wrox Press Ltd.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web Services Description Language (WSDL) 1.1. W3C Note*. Retrieved February 12, 2006, from http://www.w3c.org/TR/wsdl

Ciancarini, P., & Wooldridge, M. J. (Ed.). (2001). First International Workshop on Agent-Oriented Software Engineering. *Lecture Notes in Computer Science, Vol. 1957.* Berlin, Germany: Springer.

Dale, J. (2002). *Exposing Web services.* Paper presented at Agentcities.NET iD2, Lisbon, Portugal.

DAML+OIL (2000). Retrieved February 12, 2006, from http://www.daml.org/language/

De Bruijn, J., Fensel, D., Keller, U., & Lara, R. (2005). Using the Web service modelling ontology to enable semantic e-business. *Communications of the ACM, 48*(12), 43-47.

Finin, T., Labrou, Y., & Mayfield, J. (1997). KQML as an agent communication language. In J. M. Bradshaw (Ed.), *Software agents* (pp. 291-316). Cambridge, MA: AAAI/MIT Press.

Genusa, S., Addison, B., & Clark, A. (1999). *Special edition using ISAPI.* Indianapolis, IN: Que Publishing.

Gilbert, D., Aparicio, M., Atkinson, B., Brady, S., Ciccarino, J., Grosof, B., et al. (1995). *IBM intelligent agent strategy.* IBM Corporation.

Gottschalk, K., Graham, S., Kreger, H., & Snell, J. (2002). Introduction to Web services architecture. *IBM Systems Journal, 41*(2).

Gourley, D., & Totty, B. (2002). *HTTP: The definitive guide.* O'Reilly.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition, 5*(2), 199-220.

Harper, S., & Bechhofer, S. (2005). Semantic triage for increased Web accessibility. *IBM Systems Journal, 44*(3), 637-648.

Heflin, J., & Hendler, J. (2000). Dynamic ontologies on the Web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000),* (pp. 443-449) Menlo Park, CA. Cambridge, MA: AAAI/MIT Press.

Heflin, J., Hendler, J., & Luke, S. (1999). *SHOE: A knowledge representation language for Internet applications* (Tech. Rep. No. CS-TR-4078 / UMIACS TR-99-71). College Park, Maryland: University of Maryland, Department of Computer Science.

Hendler, J. (2001). Agents and the semantic Web. *IEEE Intelligent Systems, 16*(2), 67-73.

Hendler, J., & McGuiness, D. L. (2000). The DARPA agent markup language. *IEEE Intelligent Systems, 15*(6).

Joy, B. (2000). Shift from protocols to agents. *IEEE Internet Computing, 4*(1), 63-64.

Lassila, O. (2000). The resource description framework. *IEEE Intelligent Systems, 15*(6), 67-9.

Lin, H. K., Harding, J. A., & Teoh, P. C. (2005). An inter-enterprise semantic Web system to support information autonomy and conflict moderation. In *Proceedings of IMechE 2005: Vol. 219, Part B* (pp. 903-911).

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic Web services. *IEEE Intelligent Systems, 16*(2), 46-53.

Metcalfe, B. (2000). The next-generation Internet. *IEEE Internet Computing, 4*(1), 58-59.

Mitra, N. (Ed.) (2001). *SOAP Version 1.2, Part 0: Primer, Part 1: Messaging Framework, Part*

2: Adjuncts. *W3C Working Draft*. Retrieved February 12, 2006, from http://www.w3c.org/TR/soap-part0, http://www.w3c.org/TR/soap-part1, http://www.w3c.org/TR/soap-part2

*Object Management Grou*p (n.d.). Retrieved February 12, 2006, from http://www.omg.org

O'Brien, P. D., & Nicol, R. C. (1998). FIPA—Towards a standard for software agents. *BT Technology Journal, 16*(3), 51.

Platform for Internet Content Selection (PICS) (n.d.). Retrieved February 12, 2006, from http://www.w3.org/PICS

Reilly, J. P. (1996). *Rapid prototyping: Moving to business-centric development*. Coriolis Group (Sd). Boston/London: International Thomson Computer Press

Stahl, T., & Völter, M. (2005). *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management*. Heidelberg, Germany: dpunkt. verlag.

The Standish Group (n.d.). *Chaos Chronicles Version 3*. Retrieved February 12, 2006, from http://www.standishgroup.com/chaos/index.php

UDDI Technical White Paper (2001). Retrieved February 12, 2006, from http://www.uddi.org

URI Planning Interest Group, W3C/IETF (n.d.). *URIs, URLs, and URNs: Clarifications and Recommendations 1.0*. Retrieved February 12, 2006, from http://www.w3.org/TR/uri-clarification/

Web Authoring Statistics (n.d.). Retrieved February 12, 2006, from http://code.google.com/webstats/index.html

Web Services Choreography Requirements 1.0 (2003). *W3C Working Draft*. Retrieved February 12, 2006, from http://www.w3.org/TR/ws-chor-reqs

Web Services Transaction (WS-Transaction) (2002). Retrieved February 12, 2006, from http://www-106.ibm.com/developerworks/library/ws-transpec/

This work was previously published in Architecture of Reliable Web Applications Software, edited byM. Radaideh; and H. Al-Ameed, pp. 194-221, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.7
Formal Modeling and Specification of Design Patterns Using RTPA

Yingxu Wang
University of Calgary, Canada

Jian Huang
University of Calgary, Canada

ABSTRACT

Software patterns are recognized as an ideal documentation of expert knowledge in software design and development. However, its formal model and semantics have not been generalized and matured. The traditional UML specifications and related formalization efforts cannot capture the essence of generic patterns precisely, understandably, and essentially. A generic mathematical model of patterns is presented in this article using real-time process algebra (RTPA). The formal model of patterns are more readable and highly generic, which can be used as the meta model to denote any design patterns deductively, and can be translated into code in programming languages by supporting tools. This work reveals that a pattern is a highly complicated and dynamic structure for software design encapsulation, because of its complex and flexible internal associations between multiple abstract classes and instantiations. The generic model of patterns is not only applicable to existing patterns' description and comprehension, but also useful for future patterns' identification and formalization.

INTRODUCTION

Design patterns are a powerful tool for capturing software design notions and best practices, which provide common solutions to core problems in software development. Design patterns are a promising technique that extends reusability of software from code to design notions. A representative work of design patterns is initiated by

Gamma and his colleagues in *Design Patterns: Elements of Reusable Object-Oriented Software* in 1994 (Gamma, Helm, Johnson, & Vlissides, 1995). Design patterns may speed up the development process by providing tested and proven development paradigms. Reusing design patterns helps to prevent subtle issues in large-scale software development and improves code readability for architects and programmers. Design patterns can contribute to the definition, design, and documentation of class libraries and frameworks, offering elegant and reusable solutions to design problems, and consequently increasing productivity and development quality (Gamma et al., 1995; Wang, 2007a). Each design pattern lets some aspects of the system structure vary independently of other aspects, thereby making the system more robust to a particular kind of change.

Design patterns are used to be modeled and specified in natural language narratives, object-oriented programming languages, and UML diagrams. The traditional means are either inherently ambiguous or inadequate (Lano, Goldsack, & Bicarregui, 1996; Vu & Wang, 2004; Wang & Huang, 2005). The major problems in current methodologies for pattern specification are identified as follows:

- **The lack of a unified and generic architecture of patterns as a multilayered complex entity with a set of abstract and concrete classes and their interrelations:** Patterns have been classified in three categories known as the *creational*, *structural*, and *behavioral* patterns (Gamma et al., 1995). However, the theories for the nature of patterns and their generic architecture are yet to be sought.
- **The lack of abstraction:** Almost all patterns are described as a specific and concrete case in natural language, UML diagrams, or some formal notations. However, no generic mathematical model of patterns is rigorously established, which may form a deductive basis for deriving concrete and application-specific patterns.
- **The lack of uniqueness:** In the conventional pattern framework, there are different patterns that may be implemented by similar code; Reversely, the same pattern may be implemented in various ways.
- **The use of unstructured semantic means to denote highly complicated design knowledge in patterns:** The informal descriptions of patterns puzzle users and cause substantial confusions. Even the creators of patterns demonstrate inconsistent over the semantics of certain patterns.

The authors perceive that the above fundamental problems can be alleviated by introducing formal semantics for design patterns and their generic mathematical models (Wang, 2002, 2003, 2006a-c, 2007a-c). This approach allows for unambiguous specifications, enables reasoning about the relationships between abstract and concrete patterns, and promotes a coherent framework for the rapidly growing body of software patterns in software engineering (Beck, Coplien, Crocker, & Dominick, 1996; Bosch, 1996; Wang, 2002, 2006a, 2007a). This article presents a generic model of design patterns and a formal specification method for design patterns using Real-Time Process Algebra (RTPA) (Wang, 2002, 2003, 2007a). The approach proposed in this article aimed at the following objectives:

- **It is generic:** The same pattern model can be adopted to specify any existing and future pattern, particularly user defined patterns. To some extent, the general pattern model is the pattern of patterns.
- **It is formalized:** The mathematical semantics and formal notation system are based on RTPA (Wang, 2002, 2003, 2007a).
- **It is expressive:** Only 34 notations are used to denote class association relationship and specify patterns from three facets known

as the architecture, static behaviors, and dynamic behaviors of patterns.

- **It is structured:** Patterns are described from high-level to detailed-level via stepwise refinement using a coherent set of notations.

In this article, existing pattern specification techniques are reviewed in the following section. The RTPA methodology for pattern specification is introduced. A generic model of design patterns is rigorously modeled. Based on the mathematical semantics and general model, case studies on deriving specific pattern specifications are presented using three well known patterns such as the State, Strategy, and the MasterSlave patterns.

APPROACHES TO SOFTWARE PATTERN DESCRIPTION

A number of pattern modeling methodologies have been proposed, such as the layout object model (Bosch, 1996), the constraint diagrams (Lauder & Kent, 1998), the *language for pattern uniform specification* (Eden, Gil, Hirshfield & Yehudai, 2005), *meta-models* (Pagel & Winter, 1996; Sunye, Guennee, & Jezequel, 2000), *object calculus* (Lano et al., 1996), *pattern visualization techniques* (Lauder et al., 1998), and the *design pattern modeling language* (Mapelsden, Hosking, & Grundy et al., 1992). This section briefly reviews three major pattern specification methods and comparatively analyzes their strengths and weaknesses.

The Layout Object Model

The layout object model (LayOM) (Bosch, 1996) is an extension of object-oriented languages containing components that are not supported by the conventional object models such as layers, categories, and states. It supports the representation of design patterns in object-oriented programming languages.

In LayOM, layers are used to encapsulate objects and intercept messages that are sent to and by the objects. The layers are organized into classes and each layered class represents a concept, such as a relation with another object or a design pattern. A *state* in LayOM is a dimension of the abstract object state that is an externally visible abstraction of the internal, concrete state of objects. A *category* is defined as a client category that describes the discriminating characteristics of a subset of possible clients. *Relations* in LayOM are denoted by structural, behavioral, and application-domain relations.

For example, the *adapter* design pattern can be described in LayOM, as shown in Figure 1, which converts the interface of a class into another interface that is expected by its clients. The adapter layer can be used to class adaptation by defining a new adapter class consisting only of two layers.

In LayOM, a one-to-one mapping between the design and implementation of a pattern is provided. Another advantage of it is that there is no requirement for defining a method for every method that needs to be adapted. However, a

Figure 1. The adapter pattern described in LayOM (Bosch, 1996)

```
class Adapter
layers
adapt: Adapter (accept mess1 as newMessA,
accept mess2 and mess3 as newMessB);
inh: Inherit (Adaptee);
end;
```

disadvantage of the adapter technique is that the arguments of the message will be passed as is, which is not flexible to cover semantic analyses. Another drawback is the implementation overhead, because messages sent to or from an object need to pass all the defined layers.

The Constraint Diagrams

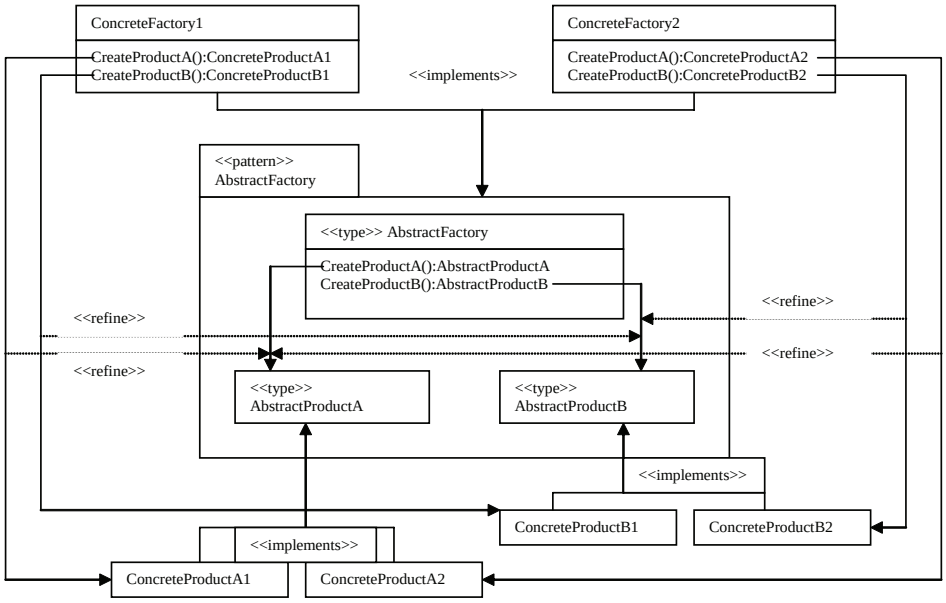
The *constraint diagram* (CD) (Lauder & Kent, 1998) denotes a pattern by three separate models known as the role, type, and class. A *role* model in CD is the most abstract and depicts layer that describes the essential spirit of a pattern without specific details. A *type* model in CD refines to the role model with abstract states and operation interfaces forming a domain-specific refinement of the pattern. A *class* in CD model implements the type model, thus deploying the underlying pattern in terms of concrete classes.

An *abstract factory pattern* deployed as a constraint model is shown in Figure 2. The core of the pattern is represented as a role mode, further refined by a type model, and implemented by a class model. However, a graphical model is not enough for a precise and unambiguous specification. There is still a need to describe additional constraints about the objects in the model. Otherwise, ambiguities cannot be avoided in the model (OMG, 1997).

LePUS: Language for Patterns Uniform Specification

Language for patterns uniform specification (LePUS) is a declarative language based on logic introduced in 1997 (Eden et al., 2005). It models class relationships and semantics, and facilitates reasoning with higher order sets. LePUS permits concise description of complex software artifacts. In LePUS, a program is represented primarily as a set of ground entities and relationships among them. Various interactions and associations that occur between participants of design patterns are abstracted classes and functions. A uniform set of classes of a dimension *d* is denoted by the class of dimension *d*+1. Total relations are functions that describe the relations between two sets of entities. Bijective and regular correlations between sets of functions, classes, and hierarchies may also be

Figure 2. The constrain diagram of an abstract factory pattern (Lauder & Kent, 1998)



modeled. A specification of the *strategy* pattern in LePUS is shown in Figure 3.

The advantage of LePUS is its higher order logic means. However, it is difficult to directly map a LePUS specification of patterns into executable programs. In addition, patterns specified in LePUS are application specific rather than generic.

Other Approaches

Meta-models for design pattern instantiations and validations are proposed in Pagel and Winter (1996) and Sunye et al. (2000) without supporting for code generation. In Florijn, Meijers, and Wionsen (1997), the *fragment-based technique* allows the representation and composition of design patterns. A design pattern proof techniques is proposed in Lano et al. (1996) using the *object calculus* as a semantic framework. The *design pattern modeling language* (DPML) is proposed in Mapelsden et al. (1992) as a visual language for modeling patterns and their instances.

A common weakness of the methods proposed so far is that they concentrate only on specific pattern descriptions. The essence of pattern structures and the generic pattern theory are overlooked. Therefore, there is still a need to seek a more powerful means and methodology that help users to utilize pattern theories and models freely in pattern-based system design.

THE RTPA METHODOLOGY FOR PATTERN MODELING AND SPECIFICATION

In the preceding section, it can be seen that existing methods for pattern specifications were inadequate and inefficient in generic pattern specifications. This section adopts RTPA to formally specify design patterns, which is a set of new mathematical notations for formally describing system architectures, static and dynamic behaviors (Wang, 2002, 2006b).

RTPA is a mathematic-based software engineering notation system and a formal method for addressing problems in software system specification, refinement, and implementation. RTPA provides an expressive means for the formal and explicit description of software patterns in order to enhance the readability of pattern architecture, semantics, and behaviors. In RTPA, a software system is perceived and described mathematically as a set of coherent processes. RTPA encompasses 17 meta processes and 17 algebraic process combination rules known as the process relations. Detailed description of RTPA may be referred to Wang (2002, 2003, 2007a).

Figure 3. The LePUS Specification of the strategy pattern (Eden et al., 2005)

$$
\begin{array}{l}
\exists context, client \in C \\
operations \in F \\
Algorithm, Configure - Contexts \in 2^F. \\
Strategies \in H \\
clan(operation, context) \wedge \\
clan(Algorithm, Strategies) \wedge \\
tribe(Configure - Context, client) \wedge \\
Invocation^{->}(operation, Configure - Context) \wedge \\
Invocation^{+->}(Algorithm, context) \wedge \\
Argument - 1^{->}(Algorithm, context) \wedge \\
Creation^{->H}(Configure - Context, Strategies) \wedge \\
Assigment^{->H}(Context, Configure - Context, Strategies) \wedge \\
Reference - to - Single^{<->}(context, Strategies)
\end{array}
$$

The Generic Model of Classes in RTPA

A unified notion of classes can be formally described by RTPA as given in "Wang (2007a)." The fundamental types of classes are the Abstract Classes (ACs) and the Concrete Classes (CCs). The former is a class that serves as a general and conceptual model to be inherited but not be instantiated. The latter is an ordinary class derived from an AC that can be instantiated.

A generic AC specified in RTPA is shown in Figure 4. The architecture part is used to specify the architectural attributes of the abstract class, which include internal variables shared by the hierarchy of classes. The static behavior part of an AC is used to define the method signatures of the AC class, where detailed implementation will be left to be done in derived concrete classes. It is noteworthy that an AC has no dynamic behavior in the specification because ACs cannot be instantiated.

Similarly, a generic CC can be formally specified in RTPA, as shown in Figure 5. The CC implements the dynamic behaviors that can be instantiated and executed in a derived object. Possible events that drive a method in a CC can be classified into message, time, and interrupt. More formal treatment of classes and their relational operations may be referred to RTPA (Wang, 2002, 2003, 2007a) and concept algebra (Wang, 2006a, 2007c).

Figure 4. Abstract class specification in RTPA

$$
\begin{aligned}
&\text{AbstractClassST} \triangleq \text{ ClassIDAC}\\
&\qquad\qquad \| \text{ Architecture}\\
&\qquad\qquad \| \text{ StaticBehaviors}\\
&\qquad\qquad \| \text{ DynamicBehaviors}\\
&\text{ClassIDAC.Architecture} \triangleq \text{ClassIDS::} \mathop{R}_{i=1}^{n} \text{<Attribute(i) : ST>}\\
&\text{ClassIDAC.StaticBehaviors} \triangleq \mathop{R}_{j=1}^{m} \text{(MethodS(j) (\{I\}; \{O\}))}\\
&\text{ClassIDAC.DynamicBehaviors} \triangleq \varnothing
\end{aligned}
$$

Figure 5. Concrete class specification in RTPA

$$
\begin{aligned}
&\text{ConcreteClassST} \triangleq \text{ ClassIDCC}\\
&\qquad\qquad \| \text{ Architecture}\\
&\qquad\qquad \| \text{ StaticBehaviors}\\
&\qquad\qquad \| \text{ DynamicBehaviors}\\
&\text{ClassIDCC.Architecture} \triangleq \text{ClassIDS::} \mathop{R}_{i=1}^{n} \text{<Attribute(i) : ST>}\\
&\text{ClassIDCC.StaticBehaviors} \triangleq \mathop{R}_{j=1}^{m} \text{(MethodS(j) (\{I\}; \{O\}))}\\
&\text{ClassIDCC.DynamicBehaviors} \triangleq\\
&\qquad \mathop{R}_{j=1}^{m} \text{<@event(j) } \hookrightarrow \text{ ClassIDCC.MethodS(j) (\{I\}; \{O\})>}
\end{aligned}
$$

The Generic Model of Patterns in RTPA

A pattern is a highly reusable and coherent set of complex classes that are encapsulated to provide certain functions. According to "Gamma et al. (1995)," patterns can be classified into the *creational*, *structural*, and *behavioral* ones. Using RTPA notations and methodology, a pattern is denoted by three parallel components known as the *architecture*, *static*, and *dynamic* behaviors, as shown in Figure 6. The architecture of a pattern specifies how many classes and components are used to compose the pattern and what relationships are among those components. The static behaviors of a pattern define what kinds of components are used to compose this pattern and what rules all components should abide. The dynamic behaviors of a pattern describe how those components interact and collaborate to realize functionality at run-time.

In Figure 6, the *architecture* of the generic pattern can be refined by a set of component logic models (CLMs), which describes the structures of a class, particularly its attributes (Wang, 2002). The *static behaviors* of the generic pattern are refined by a set of processes that are corresponding to each of the methods within the class. The process behaviors are denoted by RTPA meta-processes

Figure 6. The generic mathematical model of design patterns (Wang, 2007a)

PatternST ≙
{
  Architecture : ST
  || StaticBehaviors : ST
  || DynamicBehaviors: ST
}

PatternST.ArchitectureST ≙
{
  <Interfaces>
  || <Implementations>
  || <Instantiations>
  || <Associations>
}

PatternST.ArchitectureST.InterfacesST ≙
 PatternID**ST**::
 {
  $\overset{n}{\underset{i=1}{R}}$<Attributes(i) : RT>
  ||$\overset{m}{\underset{j=1}{R}}$<AbstractClass(j) : AC>
 }

PatternST.ArchitectureST.ImplementationST ≙
{$\overset{q}{\underset{k=1}{R}}$<ConcreteClass(k) : CC> }
PatternST.ArchitectureST.InstantiationsST ≙
{$\overset{r}{\underset{v=1}{R}}$<Instatiatin(v) : CC> }
PatternST.ArchitectureST.AssociationST ≙
{
  $\overset{m}{\underset{j=1}{R}}$(Interface(j)AC : SC // SC is a system class
    | $\overset{p}{\underset{m=1}{R}}$<Interface(j) AC.$M_m$: Interface(j')AC)
  || $\overset{q}{\underset{k=1}{R}}$<Implementation(k)CC : AC(j)>
  || $\overset{r}{\underset{v=1}{R}}$ <Instantiation(v)CC : CC(k)>
}

and their combinations using process relations for manipulating internal attributes or interacting with external components of the generic pattern. The *dynamic behaviors* of the generic pattern are refined by event-driven processes deployed by the system.

The generic pattern model may be treated as a super metapattern in object-oriented system design and programming, which models any specific software pattern at four levels known as the *interface*, *implementation*, *instantiations*, and *associations*. According to the generic model of patterns, the features of patterns lie in the hierarchical architectures as described by PatternST.ArchitectureST, as shown in Figure 6. It is noteworthy that a class is usually modeled as a two-level structure with only the class *interface* and *implementations* in literature (Taibi, & Ngo, 2003; Vu & Wang, 2004). However, the four-level hierarchical model introduced here reveals the nature of how classes may be used to form complex patterns via *instantiations* and *associations*.

The *interface* of a pattern, PatternST.ArchitectureST.InterfaceST, isolates users of the pattern from its internal implementation. Users may only access the pattern via its interface. This mechanism enables the implementation of the pattern independent of its users. Whenever the internal implementation needs to be changed, it is transparent to the users of the pattern as long as the interface keeps the same (Wang & Huang, 2005). The interface of a pattern specifies the communication protocol among pattern components. Although instances could extend their behaviors beside those interface defined in this part their communication should abide those definition. The interface is the only access point of a component inside a pattern.

Because a pattern is a highly reusable construct of a software entity, the *implementation* of a pattern, PatternST.ArchitectureST.ImplementationST, is kept at a generic abstract class until the pattern is invoked by a specific application or instantiation. In other words, because a pattern is a generic model of reusable functions, specific behaviors in an execution instance are dependent on run-time information provided by users of the pattern.

The forth component in the generic pattern hierarchy is the internal *associations*, which is used to model the interrelationships among other three-level abstractions of classes and interfaces within the pattern. The *associations* of the pattern define relationships among all components in the

pattern. Component collaborations are the soul of patterns that capture component collaborations. The flexibility, reusability, and differences of patterns are embodied by the associations (Wang & Huang, 2005). A comprehensive set of pattern association rules may be referred to Concept Algebra (Wang, 2006b, 2007c).

The generic model of patterns or the meta pattern model describes software patterns in a coherent, concise, and unambiguous way. External relationship among patterns could be deduced by this formula as well, when a super pattern is considered beyond all the component patterns. The method developed in this section helps readers avoid the drawbacks of conventional patterns, in order to develop more efficient, reusable, flexible, and predicable software systems.

CASE STUDIES ON FORMAL SPECIFICATIONS OF PATTERNS IN RTPA

This section applies the RTPA pattern specification methodology in three case studies. The *State* and *Strategy* patterns are used to demonstrate that differences between these two patterns can be clearly distinguished by RTPA specifications, while other methods rendering vague message to users. The *MasterSlave* pattern (Buschmann, 1995) is used to demonstrate the expressive power of RTPA specifications. All cases studies show that not only conventional design patterns, but also newly discovered patterns can be precisely specified by RTPA.

Formal Specification of the State Pattern

The *State* pattern allows an object to alter its behavior when its internal state changes. The structure of this pattern proposed in Gamma et al. (1995) is shown in Figure 7.

A formal model of the state pattern can be derived on the basis of the generic pattern model developed in Figure 6. Corresponding to Figure 7, the RTPA specification of the *State* pattern is given in Figure 8.

Formal Specification of the Strategy Pattern

The *Strategy* pattern defines a family of algorithms, encapsulates them in a coherent structure, and makes them interchangeable (Gamma et al., 1995). This pattern lets the algorithm vary independently from clients that use it. The structure of the Strategy pattern is shown in Figure 9.

A formal model of the Strategy pattern can be derived on the basis of the generic pattern model developed in Figure 6. Corresponding to Figure 9, the RTPA specification of the strategy pattern is shown in Figure 10.

Contrasting the State and Strategy patterns in UML, it is noteworthy that both patterns share almost identical structures. In other words, UML

Figure 7. The UML structure of the State pattern

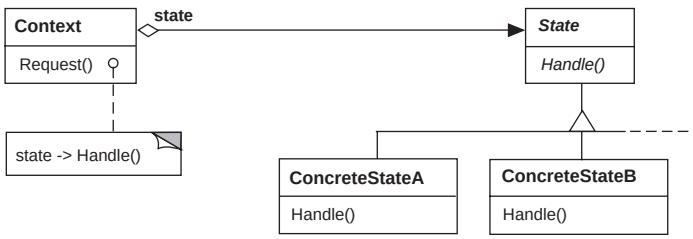
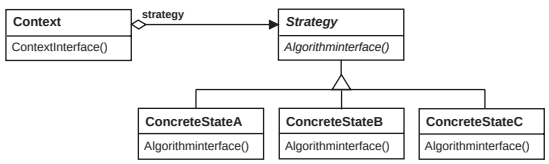


Figure 8. The RTPA specification of the State pattern

```
StatePatternST ≙
{
   Architecture: ST
 || StaticBehaviors: ST
 || DynamicBehaviors: ST
}

StatePatternST.ArchitectureST ≙
{
   <Interfaces>
 || <Implementations>
 || <Instantiations>
 || <Associations>
}

StatePatternST.ArchitectureST.InterfacesST ≙ StateST::
{  n
   R <Attributes(i) : RT>
   i=1

 || <AbstractState : AC>
}

StatePatternST.ArchitectureST.ImplementationsST ≙
{
   ConcreteContextCC : CC
   m
|| R <ConcreteState(j)CC : AbstractStateAC>
} j=1

StatePatternST.ArchitectureST.InstantiationsST ≙
{
   ContextCC : CC
   q
|| R <State(k)CC : ConcreteState(k)CC>
} k=1

StatePatternST.ArchitectureST.AssociationsST ≙
{
 // Inheritance
   n
   R ConcreteState(i)CC : AbstractStateAC
   i=1
 // Composition
   r
 || R (ConcreteContextCC.M_v : ConcreteStateCC(v) |
   v=1          ConcreteContextCC(v)
              || ConcreteStateCC(v))
 // Delegation
   n
 || R (ConcreteContextCC(i) ↳ ConcreteStateCC(i))
} i=1
```

Figure 9. The UML structure of the Strategy pattern



class diagrams, as shown in Figures 7 and 9, may not be able to discriminate the differences between these two patterns. This results in the main confusions, ambiguities, and difficulties in pattern comprehension and applications using UML-based methodologies.

However, the specifications in RTPA, as shown in Figures 8 and 10, can clearly capture the differences in the sections of class associations by

Figure 10. The RTPA specification of the Strategy pattern

```
StrategyPatternST ≙
{
   Architecture: ST
 || StaticBehaviors: ST
 || DynamicBehaviors: ST
}

StrategyPatternST.ArchitectureST ≙
{
   <Interfaces>
 || <Implementations>
 || <Instantiations>
 || <Associations>
}

StrategyPatternST.ArchitectureST.InterfacesST ≙ StrategyST
::
{
   n
   R <Attributes(i) : RT>
   i=1
 || <AbstractStrategy : AC>
}

StrategyPatternST.ArchitectureST.ImplementationsST ≙
{
   ConcreteContextCC : CC
   m
|| R (ConcreteStrategy(j)CC : AbstractStrategyAC)
} j=1

StrategyPatternST.ArchitectureST.InstantiationsST ≙
{
   ContextCC : CC
   q
|| R <Strategy(k)CC : ConcreteStrategy(k)CC>
} k=1

StrategyPatternST.ArchitectureST.AssociationsST ≙
{
 // Inheritance
   n
   R ConcreteStrategy(i)CC : AbstractStrategyAC
   i=1
 // Composition
 || ( ConcreteContextCC.M : ConcreteStrategyCC |
              ConcreteContextCC
            || ConcreteStrategyCC )
 // Delegation
 || ConcreteContextCC ↳ ConcreteStrategyCC
}
```

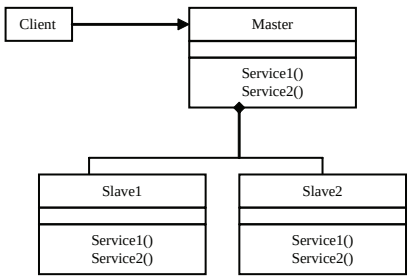
different composition approaches. The RTPA models show that in the former, all concrete state classes are instantiated by concrete classes simultaneously, where the concrete state objects have the same lifecycle as those of the concrete context objects; while in the latter, only one concrete strategy object alive within any point of time in the concrete context object lifecycle. This is the essential difference between those two patterns, which cannot be expressed explicitly by UML syntaxes and semantics.

Formal Specification of the MasterSlave Pattern

It is observed that traditional methods for pattern modeling are focused on existing and specific patterns proposed in "Gamma et al. (1995)." However, most practical patterns in software engineering are user-defined rather than pre-specified. Therefore, a generic pattern model is needed to support users to deductively model new patterns in practice.

A system pattern, *MasterSlave* (Buschmann, 1995) as shown in Figure 11, is presented in this subsection in order to demonstrate the application of the generic pattern model and the expressive power of RTPA for modeling patterns. The *MasterSlave* pattern handles the computation of replicated services of a software system to achieve fault tolerance and robustness. Replication of services and the delegation of the same task to several independent slave servers is a common strategy to handle fault-tolerant requirements in safety-critical software systems.

Figure 11. The UML structure of the MasterSlave pattern



The *MasterSlave* pattern consists of two kinds of components: the master and the slaves. Clients of the pattern interact with the master component directly. However, the master component does not implement services by itself. It delegates the services to a number of slave components, where at least two identical slave components exist in the system with the same set of functionality. The slave components are completely independent of each other, and they may use different strategies for providing the designated service. The master component delegates a requested service to all slave components and chooses one of the most suitable responses as the result for the client.

A formal model of the MasterSlave pattern can be derived on the basis of the generic pattern model developed in Figure 6. Corresponding to Figure 11, the RTPA specification of the MasterSlave pattern is given in Figure 12. The derived pattern precisely describes the architecture and associations between member classes of the *MasterSlave* pattern. Several strategies may be available for the master component to select results provided by the slaves, such as the result first returned, the majority result returned by all slaves, or the average result of all slaves.

CONCLUSION

This article has reviewed existing pattern specification methods and problems yet to be solved. A generic mathematical model of patterns has been presented using Real-Time Process Algebra (RTPA). Based on it any design patterns, either system-specified or user-defined, can be derived. With the RTPA support tool, a pattern specified in RTPA can be automatically translated into code in programming languages (Tan, Wang, & Ngolah, 2006).

Figure 12. The RTPA specification of the MasterSlave pattern

```
MasterSlavePatternST ≙
{
   Architecture : ST
 || StaticBehaviors : ST
 || DynamicBehaviors : ST
}

MasterSlavePatternST.ArchitectureST ≙
{
   <Interfaces>
 || <Implementations>
 || <Instantiations>
 || <Associations>
}

MasterSlavePatternST.ArchitectureST.InterfacesST ≙
MasterST ::
{
   ||(i=1..n) <Attributes(i) : RT>
  || <AbstractMasterComponent : AC>
}

MasterSlavePatternST.ArchitectureST.ImplementationsST ≙
{
  <ConcreteMasterComponentCC : CC>
|| ||(j=1..m) <ConcreteSlaveComponent(j)CC : CC>
}

MasterSlavePatternST.ArchitectureST.InstantiationsST ≙
{
 ConcreteMasterInstanceCC: AbstractMasterComponentAC
 || ||(k=1..q) <ConcreteSlaveInstance(k)CC :
    ConcreteSlaveComponent(jN)CC>
}

MasterSlavePatternST.ArchitectureST.AssociationsST ≙
{
 // Inheritance
 ConcreteMasterComponentCC :
                AbstractMasterComponentAC
 // Delegation
 || ( ConcreteMasterComponentCC ↳
            ||(j=1..m) ConcreteSlaveComponent(j)CC)
 // Aggregation
 || (||(k=1..q) ConcreteMasterComponentCC.M :
    ConcreteSlaveComponentCC (k) |
                ConcreteMasterComponentCC
                ⇵ ConcreteSlaveComponentCC)
}
```

This work has revealed that a software pattern is a highly reusable design encapsulation that encompasses complex and flexible internal associations between a coherent set of abstract classes and instantiations. The generic model of patterns has provided a pattern of patterns. It is not only applicable to existing patterns' modeling and comprehension, but also useful for future patterns' identification and formalization.

ACKNOWLEDGMENT

The authors would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. We would like to thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Beck, K., Coplien, J. O., Crocker, R., & Dominick, L. (1996, March). Industrial experience with design patterns. In *Proceedings of the 19th Intel. Conf. on Software Engineering,* (pp. 103-114). Berlin: IEEE CS Press.

Bosch, J. (1996). Relations as object model components. *Journal of Programming Languages, 4*(1), 39-61.

Buschmann, F. (1995). *The MasterSlave Pattern, pattern languages of program design.* Addison-Wesley.

Eden, A. H., Gil, J., Hirshfeld, Y., & Yehudai, A. (2005). *Towards a mathematical foundation for design patterns (*Tech. Rep.). Dept. of Computer Science, Concordia University, Montreal, Canada.

Florijn, G., Meijers, M., & Wionsen, P. V. (1997). Tool support for object-oriented patterns. In *Proceedings of the 11th European Conference on Object-Oriented Programming* (ECOOP'97)(pp. 472-495), Jyvaskyla, Finland.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object oriented software*. Reading, MA: Addison-Wesley.

Lano, K., Goldsack, S., & Bicarregui, J. (1996). Formalizing design patterns. In *Proceedings of the 1st BCS-FACS Northern Formal Methods Workshop* (p. 1).

Lauder, A., & Kent, S. (1998). Precise visual specification of design patterns. In *Proceedings of the 12th European Conference on Object-Oriented Programming* (ECOOP'98), (LNCS, 1445, pp. 114-134). Springer-Verlag.

Mapelsden, D., Hosking, J., & Grundy, J. (1992). *Design pattern modeling and instantiation using DPML*, (Tech. Rep.). Department of Computer Science, University of Auckland.

OMG. (1997). *Object Constraint Language Specification 1.1.*

Pagel, B. U., & Winter, M. (1996). Towards pattern-based tools. In *Proceedings of the EuropLop'96*, (pp. 3.1-3.11).

Sunye, G., Guennec, A. L., & Jezequel, J. M. (2000). Design patterns application in UML. In *Proceedings of the 14th European Conference on Object-Oriented Programming* (ECOOP'00)(pp. 44-62), Sophia Antipolis, France.

Taibi, T., & Ngo, D. C. L. (2003). Formal specification of design patterns–a balanced approach. *Journal of Object Technology, 2*(4), 127-140.

Tan, X., Wang, Y., & Ngolah, C. F. (2006, May). Design and implementation of an automatic RTPA code generator. In *Proceedings of the 2006 Canadian Conference on Electrical and Computer Engineering (CCECE'06),* (pp. 1605-1608). Ottawa, Canada: IEEE CS Press.

Vu, N. C., & Wang, Y. (2004, May). Specification of design patterns using real-time process algebra (RTPA). In *Proceedings of the 2004 Canadian Conference on Electrical and Computer Engineering* (CCECE'04), (pp. 1545-1548). Niagara, Falls, Ontario: IEEE CS Press.

Wang, Y. (2002, October). The real-time process algebra (RTPA). *Annals of Software Engineering: An International Journal, 14*, 235-274.

Wang, Y. (2003). Using process algebra to describe human and software behaviors. *Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy, 4*(2), 199-213.

Wang, Y. (2006a, July). On concept algebra and knowledge representation. In *Proceedings of the 5th IEEE International Conference on Cognitive Informatics (ICCI'06),* (pp. 320-331). Beijing, China: IEEE CS Press.

Wang, Y. (2006b). On the informatics laws and deductive semantics of software. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 167-171.

Wang, Y. (2006c, July). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation, Invited Plenary Talk. In *Proceedings of the 1st International Conference on Rough Set and Knowledge Technology* (RSKT'06), (pp. 69-78). Lecture Notes in Artificial Intelligence, LNAI 4062. Chongqing, China: Springer-Verlag.

Wang, Y. (2007a, July). *Software engineering foundations: A software science perspective*. CRC Book Series in Software Engineering (Vol. II). USA: CRC Press.

Wang, Y. (2007b, January). The theoretical framework of cognitive informatics. *The International Journal of Cognitive Informatics and Natural Intelligence (IJCiNi), 1*(1), 1-27. Hershey, PA: IGI Publishing.

Wang, Y. (2007c). Keynote speech, on theoretical foundations of software engineering and denotational mathematics. In *Proceedings of the 5th Asian Workshop on Foundations of Software*, Xiamen, China, (pp. 99-102).

Wang, Y., & Huang, J. (2005, May). Formal models of object-oriented patterns using RTPA. In *Proceedings of the 2005 Canadian Conference on Electrical and Computer Engineering* (CCECE'05), Saskatoon, Canada, (pp. 1822-1825). IEEE CS Press.

This work was previously published in International Journal of Cognitive Informatics and Natural Intelligence, Vol. 2, Issue 1, edited by Y. Wang, pp. 100-111, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 2.8
Building an LMS with Ubiquitous Software

Michael Rees
Bond University, Australia

Charles Herring
G-Netech Pty Ltd, Australia

ABSTRACT

Teaching institutions around the world are using large, unwieldy, and expensive learning management system (LMS) packages that are beginning to have profound effects on their whole organizations. Such LMS packages in turn go to great lengths to interoperate with the desktop information productivity software that almost all institutions use, Microsoft Office System. Since a very large part of the instructional content is generated in Office, it seems sensible to investigate whether straightforward extensions of the Office System could become an LMS in their own right. This chapter describes research and development that integrated Microsoft Office System, SharePoint Windows Services, and SharePoint Portal server (SPS) as the heart of an off-the-shelf LMS. Already designed to work closely with Office, SPS features are compared against the list of features expected of an ideal LMS. Where gaps in the LMS features were discovered in SPS, a number of small extensions of standard Office applications were proposed to fill these gaps and create a credible LMS. These Microsoft tools and custom extensions were put to use in teaching and administration during a 2-semester (8-month) trial at Bond University. The SPS installation was hosted in partnership with G-Netech Pty Ltd. This Bond University/G-Netech SharePoint Alliance project (BUGSA or SharePoint Alliance) was able to call upon combined research, development, and teaching expertise provided by the partners. The outcomes of the short trial support the concept of the Office System as a viable LMS.

INTRODUCTION

When one surveys the choice of learning management system (LMS) software, one finds a very lop-sided situation. What were the two leading examples of a LMS deployed across the higher edu-

cation institutions of the world, WebCT (WebCT, 2006) and Blackboard (Blackboard Academic Suite, 2006), have now merged into one company although for the time being the two products are still differentiated. Together these products account for 70-80% of the total market. Significant in the context of this chapter is that with a decade of development both products are available in enterprise versions comprising collections of major components, which include content management systems, portals, and communication tools. Such all-encompassing feature sets inevitably lead to high costs and significant training needs.

Despite such dominance, there is a long tail of alternative software for the primary LMS function. At the tail's end are the local in-house systems that each institution has developed with their own technical and teaching staff. Of late, there has been a significant effort by a group of 80 institutions worldwide to work together on producing a common learning environment called Sakai (2006). The Sakai Project uses an interesting variant of the open source development model called the community source model, which involves some financial inputs. What remains are open source systems with their growing group of users and several less significant commercial LMS packages that typically target the smaller institutions.

From their own product descriptions, the supplier companies extol the fact that an integrated LMS is an enterprise level system. Such systems by implication therefore require very significant support in terms of technical design, administration, and staff training in addition to content design and creation teams for the support of the teaching staff. It is not surprising that the founder of the WebCT LMS, Goldberg (2004), in his keynote presentation at AusWeb 2004 indicated that, in his experience for many institutions the LMS, has become the most important information system on a university campus after the payroll system! By implication, however, this entails a large cost in software acquisition, technical support, ongoing maintenance, and integration with other institutional information systems.

It is the authors' contention that an important avenue of LMS development has largely been ignored, that of adapting existing off-the-shelf content management and interactive communication systems intended for use by organisations in general across all industry and government sectors. One major example of this class of systems is the Microsoft SharePoint technologies (2006) aimed at generic Web portal and information repository use. As will be described in detail in this chapter, a very great advantage of SharePoint is its tight integration with the ubiquitous Microsoft Office applications. These are the main information content creation tools used by educators across the world and are licensed by very many institutions.

In the pages that follow, a case is made for adopting SharePoint as the foundation of an LMS. Examples and case studies show how this can be achieved. Some new developments that are happening now, and planned changes to Microsoft Office over the next months, are discussed. This shows that future trends will lend credence to the use of off-the-shelf software to compete with the highly specific and less flexible LMS packages currently in use. Leveraging existing software in this way helps in reducing software acquisition costs and training needs.

BACKGROUND

The authors have worked together for a number of years in a research centre environment where software development of new leading-edge solutions was the norm. Often these software packages were built from scratch unnecessarily, and the authors soon became proponents of the intelligent deployment of straightforward off-the-shelf software that can often be extended to provide support for new solutions. This comment applies to apparently mundane applications such

as Microsoft Word, Excel, and PowerPoint. These applications can often be tailored for significant specific needs by writing embedded extensions in Visual Basic for Applications that manipulate the inherent document object models.

During their years of research collaboration, the authors have also been using and gaining experience with the often-ignored Microsoft Office System SharePoint technology (Microsoft SharePoint, 2006). Various research and development projects used SharePoint for a wide range of different purposes (Herring & Rees, 2001). The uses range from innovative collaboration environments based solely on the Windows desktop and Windows Explorer (Herring, Rees, Loch, & Rhodes, 1998) to distributed software engineering support (Herring et al., 2001) and research into military command and control systems (Barros, Herring, Hildebrandt, & Rees, 2000) as a replacement for a very complex, purpose-built collaboration system.

In October 2003, the release of Microsoft Office System 2003 significantly enhanced the SharePoint technology. The existing portal server became SharePoint Portal server 2003 (SPS) and a significant reduction in licence costs made this software more accessible. SPS is designed so that it can scale across a large enterprise. At the same time, the underlying server technology has the capability to be effective for situations where only tens of users are involved. SPS needs Windows Server 2003 as the operating system together with the SQL Server 2000 database.

The underlying heart of SPS uses a free software enhancement for Windows Server 2003 and the IIS Web server called Windows SharePoint Services (WSS) that provides an extensive set of features to provide highly interactive collaborative Web sites. An added bonus allows WSS to be used stand-alone without SPS and may even utilise the free Microsoft SQL Server Desktop Environment (MSDE) for effective, small-scale deployments. As will be discussed in detail in a later section WSS Web sites provide substantial overlap with the requirements of an LMS as defined below.

From the authors' experience, it appears that SharePoint is viewed as a tool only suitable for the corporate environment where teams are engaged in typical commercial activities such as production, sales, and marketing. The authors attempt to show that SharePoint technologies are much more flexible than that, and can find a role as the foundation of an LMS system. This is particularly true of the primary role of delivering educational materials in an effective manner for teaching and learning.

With the imminent launch of Office System 2003, the Brisbane office of Microsoft Australia launched the *Spotlight on Office 2003* software competition to showcase powerful new uses of the software. The authors teamed up with G-Netech Pty Ltd to enter the "Education Services" software into the competition. The Education Services system consisted of a managed Sharepoint Portal Server and custom applications developed by G-Netech Pty Ltd staff with some Bond University student involvement. This university developed subject-specific Windows Sharepoint Services sites and used them as part of the trial. Deployed during May through September 2003, just prior to the Office System 2003 launch in October that year, Education Services was selected as one of three finalists in the competition, and also won a prestigious Asia-Pacific Solution Developer Award.

Following on the education services case study Bond University approached G-Netech to initiate a further trial of SharePoint with participants from several faculties at the university. This became the SharePoint Alliance (SPA) case study that lasted for two semesters, a period of 8 months at Bond where there are three full 14-week semesters in each calendar year. The SPA case study added further experiences and examples to the authors' expertise and lends additional credence to the SharePoint LMS role.

Over recent months, Microsoft has begun to realize that SharePoint can even play other supporting roles with enterprise LMS systems such as Blackboard. The discussion concludes with comments on these plans and the type of resources that are likely to emerge. In addition, the next major version of SharePoint will play an even more important central role in the new Office 2007 System expected early in that year. Some early indications of the additional benefits are presented.

This article first compares the features that SharePoint technologies provide out of the box with the features of an LMS and shows the significant overlap. Next, the basis of the SharePoint Alliance (SPA) Trial conducted at Bond University is discussed. One of the authors taught a sample university subject using SPA and the experiences and outcomes are described in some detail. The paper ends with a discussion of the SPA Trial results with suggestions for improvements if the approach is adopted.

LEARNING MANAGEMENT SYSTEM CHARACTERISTICS

Bond University is Australia's first private university established in 1989. Bond has about 3,000 students across four broadly defined faculties and prides itself on high quality, small-class teaching. Any adoption of an LMS is intended to act as an out-of-class supplement to the face-to-face teaching that all students receive in class. Even in this supporting role, the list of characteristics that must be present in an LMS turns out to be the same as for distance learning except that the emphases on different LMS components are different.

At Bond, only a very small number of subjects are offered in a distance-learning mode. To date mainly static Web sites are used to support classes in a somewhat ad hoc manner. The LMS is intended to replace this teaching support with a consistent, interactive collection of educational materials accessible online in 24x7 mode. Indeed at this time of writing Bond is starting the implementation of BlackBoard as the campus-wide LMS. However, there are positive signs that the early SharePoint experiences described here will carry forward into a joint use of BlackBoard. This is described further in the Future Trends section below.

Establishing a succinct set of characteristics that a LMS should possess is not straightforward when taking into consideration the wide range of teaching styles. Kennedy (2005), in surveying students' reaction to the open source Moodle (2006), gives a compact set of features outlined in Table 1 split across three broad support categories: tools for communication, students and course contents. The lists are comprehensive but are biased towards technical descriptions such as drop box and wiki, for example.

Table 1. Kennedy's learning management system features

1. Communication Tools	**2. Student Tools**	**3. Course Tools**
1.1 E-mail	2.1 My notes	3.1 Staff information
1.2 Student list	2.2 Student drop box	3.2 Assessment
1.3 Discussion board	2.3 Change your information	3.3 External links
1.4 Virtual classroom	2.4 Student calendar	3.4 Announcements
1.5 Student pages	2.5 Student manual	3.5 Course map
1.6 Group homepages	2.6 Student homepage	3.6 Study material
	2.7 Wiki	3.7 Search

When Bond University began the process to choose an LMS a conventional Web-based teaching and learning working group was established (one of the authors was a member). The chapter authors put forward the education services enhancement to SharePoint before this group as a contender for the LMS. As part of this process group members were asked to draw up an ideal list of LMS features. Table 2 shows the feature list drawn up by the authors augmented with comments indicating whether WSS and/or SPS can support each feature. This second table incorporates the three tool sets from Table 1 into a single list.

It is interesting to contrast this list to one given in Wikipedia (Virtual Learning Environment, 2006), only discovered after Table 2 was first drawn up. Note that virtual learning environment (VLE) is the preferred term (changed recently from Managed Learning Environment) but with a note of several other equivalent terms, LMS is one of them. Only feature 9, assessment support, is missing from the SharePoint capabilities and to a lesser extent feature 8, full support of class lists. However, it is recognized that assessment support and centralized grade books play significant roles in teaching and learning. Class list management too is one of the larger administrative tasks for teaching staff, and any support of this activity is to be welcomed.

SharePoint ALLIANCE TRIAL

A small steering group of staff formed the Bond University-G-Netech SharePoint Alliance (SPA) trial to take place over two teaching semesters (semesters 2 and 3) between May and December

Table 2. Learning management system feature list

	Feature	WSS Support	SPS Support
1	Structured access to and search of repository of learning materials with upload capability	Full support via document libraries per class	Institution-wide repositories
2	Cross-site searching for learning object names and content	Full support per class site	Institution-wide search
3	Fine-grained secure access based on user roles (staff, tutor, student, class representative, and so on) down to individuals	Full support	Full support
4	Subject events (schedules) for class times, assessment deadlines, reminders, and so on	Full support per class	Some support; extension required
5	Notices: announcements, news, task lists, surveys	Full support per class	Full support across institution
6	Discussion groups: inter-class communications, frequently asked questions	Full support	Full support at institution level
7	Notifications of Web site changes: additions, modifications, deletions	Full support via e-mail alerts	Full support at institution level
8	Class lists and group membership (tutorial, workshop, presentation)	Some support via users and contacts lists	Full support via active directory
9	Assessment submission and marks reporting	Very limited support	Simple workflow for document submission only
10	Template based creation of subject Web sites and Web site components	Full support	Full support
11	Real-time class communication; instant messaging, audio, and video	Some support via MS instant messenger integration	As per WSS

2004 (Bond University operates three full teaching semesters each year so that a 6-semester university Bachelors degree can be completed in two calendar years). With staff and equipment resources not being available from within Bond University itself it was decided to outsource the hosting of the SPA trial to a local IT company, G-Netech Pty Ltd. Four faculties, business, law, humanities and information technology, contributed equally to the very modest costs of the SPA trial.

Eventually staff in business, information technology, and humanities used SPA for teaching or coordinating student work. During the trial period, the university was being audited by the Australian Universities Quality Audit organization and some material pertinent to this administrative task was also entered into the SPA portal. A number of staff used additional SPA sites for planning and preparation of information content of various kinds.

Bond University contracted with G-Netech Pty Ltd to provide a managed server at a cost of USD 750/month to host the SharePoint Portal. The server used was a standard Dell dual-processor, one gigabyte RAM, 73-gigabyte RAID disk system as typically used for hosting. Setup and installation of all required software took less than one man day of labour. Microsoft provides universities with heavily discounted prices that make this enterprise level system extremely attractive to deploy. At the time of the SPA trial, the annual cost for an SPS license was about USD 3,750. From a systems administration perspective, once the portal is initially set up, the administration of the information structure and content creation can be delegated to a group of end users. Using the role-based security of SharePoint the senior user group can in turn delegate further, thus spreading the administrative load and providing administrative control at the appropriate positions within the information and teaching hierarchy.

A sample portal home page is shown in Figure 1. This demonstrates some of the top-level features available to all users. A single sign-on for each user assigns roles that control the visibility of content and dictate access permissions. Major portal content areas are shown across the heading section and are easily customised. Depending on the roles assigned to them, users can perform a defined set of actions, or possibly none of the actions, listed in the left column. Figure 1 shows the actions available to an administrator. The main content in the central column shows the modules known as Web parts selected by the page author. In this case, the enterprise-wide news service is displayed together with the events Web part. News items can be targeted at specific individual or sets of roles so that list of news items each user sees is highly customised.

Two other major features appear at the top right of Figure 1. The enterprise search feature allows a number of search scopes to be defined ranging from portal areas, the whole portal, and specified external sources that are constantly indexed by the portal search mechanism. The second feature is "My Site," which is an individual, editable Web site that can be allocated to each user including all students. This Web site makes available a comprehensive set of portal Web parts for each user. A sample My Site is shown in shown in Figure 2.

The My Site user is given a powerful individual Web site that can be used to collect a wide variety of information related to the portal such as links, news items, tasks, and e-mail from an exchange server, and public and private documents. Different collections of information can be made available in the public and private views of the Web site. New pages, each containing several Web parts, can be added to the site and cross linked. Searching is also possible across the user's own My Site. At a single stroke, the My Site features satisfy the student tools section of Table 1.

Using the spotlight on office work mentioned above the first author was able to build subject Web sites very quickly for the SPA trial. A screenshot of the home page of the sample site is shown in Figure 3. Note that by default all WSS sites are

Figure 1. SharePoint portal home page

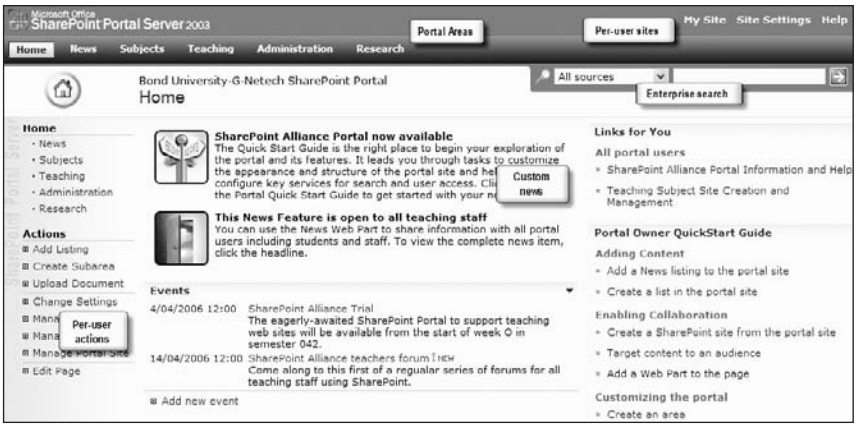


Figure 2. Sample My Site

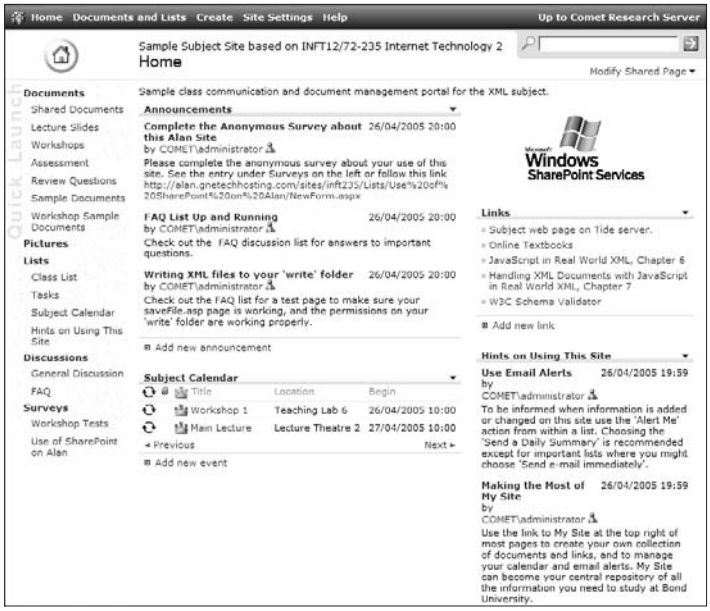
split into three columns. Other layouts such as the more typical two columns are possible.

Several of the important LMS features from Table 2 are shown in Figure 3. On the left appears the main content navigation showing document libraries (repositories) for various types of subject study materials, subject calendar, outstanding tasks, class list, discussion forums, and surveys. The main page content appears in the central column, which shows the important announcements with automatic expiry dates and the upcoming events in the subject calendar–vital information for the students. On the right is shown a series of useful hyperlinks to both internal and external teaching resources and some hints. The search box in the top right is standard on all pages in a WSS site. Searching for keywords in titles and document contents plays a vital role in improving the user experience of any LMS.

Thus, the portal and Web site contents in Figure 1, Figure 2, and Figure 3 exhibit all the LMS features of Table 2 except for nine. The instant messaging requirement of feature 11 relies on the use of Windows Messenger or MSN Messenger symbols appear to the right of user names the user can click to contact that user with Windows or MSN Messenger.

Nevertheless it can be seen that the SharePoint technologies otherwise provide the bulk of the features one would expect to find in a full-featured

Figure 3. Sample Windows SharePoint services site for a subject



LMS. It was on this basis that some staff at Bond decided to instigate a trial of SharePoint with a view to determine its suitability primarily for teaching delivery, but also for the related teaching preparation and administration processes.

WEB SITE AND PORTAL COMPONENTS

According to the Wikipedia definition, "portlets are reusable Web components that display relevant information to portal users." SPS supports the portlet concept but refers to them as "Web parts." SPS and WSS are delivered with a standard set of about 20 useful Web parts such as documents and links lists, calendars, announcements, tasks, discussion boards, surveys, and so on. Every new list that a Web site owner creates becomes a new Web part. Once created the Web parts can be arranged in columns making up a page by simple dragging and dropping into position.

Web part glue facilities are also incorporated so that Web parts can be linked together with the output of one Web part being input to another. Simple glue can be applied without programming. More powerful Web parts can be coded to work with the Web part object model available to the standard application program development tools such as Visual Studio.

The two SharePoint technologies acting together provide the dual architecture that is typical of enterprise portal software that is used to implement an LMS:

- The outer or community level where institutional information is located plus an information set potentially customized for each individual user.
- The inner Web sites where educators locate information and interact with tools specific to a teaching class, educational activity such as content preparation and planning, or administrative processes.

In the case of Blackboard, for example, these functions are embodied in two separate major modules that are almost separate products in their own

right. In the case of SharePoint, the community level is a single product, SharePoint Portal server 2003, and the inner Web sites sit above the IIS Web server and constitute an additional software layer for which is a standard component of the Windows operating system.

SharePoint Portal server provides a number of important features at the university community level controlled by an administrator with portal responsibilities:

- The overall portal information structure is made visible via institutional areas and regions, and a single sign-on gives access according to each user's role.
- Each page at the top portal level varies dynamically according to individual user role so users see a view personalized for them.
- Users are provided with their own Web site called "My Site"; users can copy and link to resources and learning objects and create and upload their own Web part content.
- Cross-portal news and announcements targeted at particular user audiences defined by the institution.
- Information islands located in individual Web sites are aggregated to provide a cross-portal search with users only able to access information specific to their role.

One of the authors acted as the portal administrator for the SPA trial, but with less than 200 portal users the news feature was used only a little, and searching was allowed across the whole portal content. Other features such as single sign-on, My Site and portal user roles were exercised much more fully.

As previously mentioned software developers can build custom Web parts to provide virtually any functionality both in terms of integration within the SharePoint system or to interface to external systems. An example customized Web part was developed by one of the authors to permit a consolidated event calendar view for each user. The motivation for this was to give students the ability to see all of their lectures, labs, and any other appointments on a single event list in one view within their My Site pages. This demonstrated the flexibility of extending the system to provide new learning management specific features.

Another example of extending the Office System was the development of an "Office research pane" within Word that provided the ability for a user to query the Bond University Library catalogue directly. Figure 4 shows how a query to the library information system appears within the Word research pane. Students can easily follow the hyperlinks associated with the library resources and insert references into their Word documents with one click.

One of the new applications in the Office System suite is InfoPath. InfoPath is an XML-based "smart" forms creation and deployment application. InfoPath is integrated with SharePoint form libraries into which the form templates are placed.

Figure 4. Bond University Library catalogue search in the word research pane

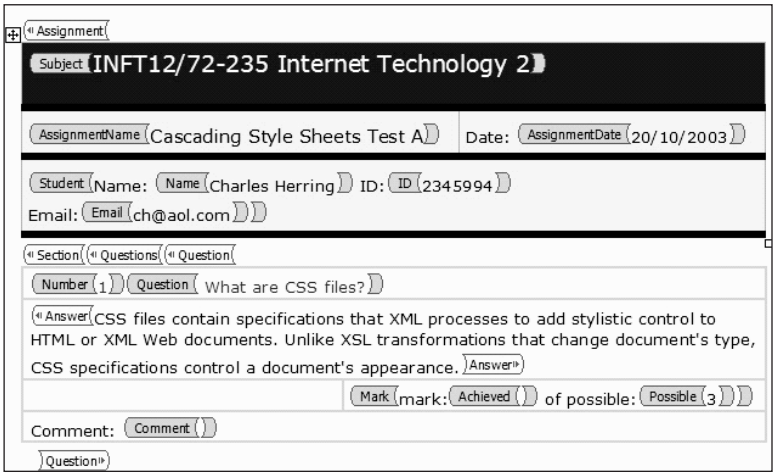
Clicking on an InfoPath form opens a Web page in which the user can enter the data required by the form. A submit button uploads the form contents to the Form library. From here, users can view and manipulate data originating from InfoPath forms. An InfoPath assessment and submission form was developed to demonstrate this capability and to allow students to submit details of an assignment and attach the assessment document set to the form.

Finally, a Word "smart document" was developed that showed how it is possible to build, submit, and mark assessments. The assessment is created as a Word document with specially marked smart sections where students enter their answers. On completion, the students upload their documents to an assessment document library, a drop box. There the assessor opens these smart documents in Word. Figure 5 shows the document opened in Word with the smart tags displayed, normally these tags would be hidden during actual assessment. The smart code assists the marker by reading correct answers from a database and displaying them in a form in the Word task pane (not shown) with fields for the assessor to allocate marks for each answer section and enter comments. A submit button merges the marks and comments into the students' original documents which are placed in another document library from where the students can download them. This solution required a specific test (on Cascading Style Sheets) to be created with correct answers and marks allocation. Custom code was then added to create the task pane functionality. As a result of this experience the authors realise that a wizard approach is needed to make such assessments generic. However, it is pleasing to see how little code is required.

Of course, the survey Web part built into WSS can be used very easily to build quizzes and online tests. The surveys can be exposed at the correct times and each student limited to one attempt. Once completed the collected survey responses (quiz answers) can be hidden from other students (or exposed if need be). However, there is no built-in assessment process. The assessor must view each response and allocate marks manually. Nevertheless, the survey Web part does contain a useful graphical summary of responses that can be used to assess student opinion in an automatic way.

From the initial analysis and from experience it is clear that SPS and WSS are weak in terms of support for class assessment. A set of specific assessment Web parts need to be written to march the features of other LMS packages. However, the

Figure 5. Assessment document in word using smart tags



two extensions described above help to fill the gap in SharePoint assessment support.

SAMPLE SUBJECT ON SPA

At Bond, where the primary teaching delivery is face-to-face in small classes, the need for high-cost multimedia-based interactive educational material is substantially reduced. The instructors provide the teaching interaction, do the demonstrations, act the roles, encourage student group activity, and so on. Of course study materials and external electronic resources still need to be available on the subject Web site.

The first author created four subject Web sites during the trial, two in each semester. The features used were substantially those shown in Table 2:

- Document libraries for subject description, lecture slides, lecture notes, tutorial and workshop handouts, assessment sheets, marks, data files, and all other documents used in the subject. Students mostly receive printed copies of these materials
- Lists for class members, class calendar, and upcoming tasks
- Several additional Web part pages that contain additional subject content.
- At least two discussion groups, the unmoderated general discussion group and a moderated FAQ group for more formal queries
- A number of surveys to elicit student opinion and for simple supervised online test submission
- Hyperlink lists to internal and external educational resources

For instructors, document library and list creation is as simple as clicking on the "create" link at the top of the subject home page and selecting from the list of options that includes a custom list creation mechanism. Each document library and list is a new Web part and can be placed in any Web part page. Document libraries also become Web folders so that Windows Explorer can be used to drag and drop documents between folders on the instructor's local machine and the subject sites. All Office applications can open and save directly to Web folders, while most Windows applications exhibit this behaviour as well. Little or no staff training is needed as this is a simple extension of the normal document management activities on any Windows machine.

Creating announcements, new class events, tasks, links, and other list contents is a simple matter of filling in a form for each item. Microsoft Excel is fully integrated with SharePoint and can be used to download or upload any list. Teaching and learning content creation could not be simpler.

Of course, many of the same benefits apply to student access to the teaching Web sites. Their ability to change lists and document libraries on the subject site is appropriately restricted, but on their own My Site they have access to much the same functionality. In common with most of the popular LMS software no student training is needed apart from the occasional demonstration in class to overcome the usual initial reticence to alter information on a Web site.

In the opinion of the instructor, probably the most useful feature of SharePoint is the e-mail alert capability. For any list or document library, the user can nominate to be informed by e-mail of changes in content. The e-mail can be sent immediately, or most usefully, in a daily or weekly summary consolidated from all nominated lists. Hyperlinks in the e-mail take the user directly to the list items in question so that the new information can be viewed directly without having to navigate from the home page of the site. The benefits to students, too, should be obvious, and in several surveys students nominated this feature as the most beneficial. However, despite constant urgings not all students by any means expended the effort (only 3 clicks per list) to switch on e-mail alerts. In fact, the surveys show that only

just over 50% of the students used e-mail alerts during the subject.

The end of semester surveys yielded detailed results that show strong student support for SharePoint as a beneficial learning tool. Only a few representative samples are presented here. From a class of 19 students there were 15 responses to the online survey. One important question asked students to give an indication of how often they accessed SPA and Figure 6 shows the distribution. Assuming a 5-day studying week the results indicate the site was accessed 1-2 times each day.

Again, from a survey students were asked to rate from 1 (low) to 10 (high) how well SPA supported various class activities, the results were: communicating with the lecturer 7.9, communicating with classmates 6.7, during practical assignments 7.7 and overall 6.8. Staff-student communication came top with a somewhat surprising practical assignment support a close second. This latter result is probably due to the general discussion group where students often help each other (usually in the early hours of the morning just before an assignment deadline), and a FAQ discussion group where the instructor answers questions about problems with the practical assignments.

When asked about the good features of the SharePoint sites the top three, in order, were e-mail alerts, document libraries, and class calendar. The three least useful features in order were My Site, Web page layout, and login problems. It was disappointing to see the apparently very useful My Site not being used to any great extent. A possible explanation is the short time limit of the SPA trial, and the students knowing that the My Site information they gathered was only likely to be temporary. In addition the mechanism for downloading their carefully gathered site content at the end of the subject was not spelt out in detail by the instructor. This is a lesson that all instructors using LMS packages that offer individual student sites will do well to remember.

The login problems stemmed from the need to allocate students additional user accounts for the SPA trial—our technical services group were not able to allow the on-campus authentication system to be accessed from the external G-Netech hosting site. The problems with the Web site layout are more puzzling since the site templates have been carefully designed. A possible explanation is that once a document in a library or an additional Web part page is opened the connection with the Web site can be lost unless the content creator is careful and consistent.

The built-in SharePoint page access log analysis can be a useful educational tool for the instructor. A part of the access log for a subject Web site over the semester is shown in Figure 7. A student with a very low access count compared to other students might need some additional help, especially one with a negligible value in the

Figure 6. Distribution of site accesses per week

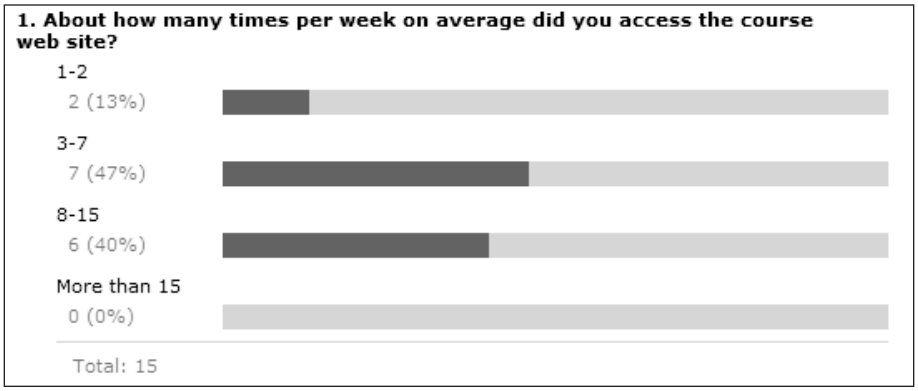


Figure 7. Subject Web site page access log

User	Total Hits	Recent Month
gnetech\jostensj	355	84
gnetech\jwatson	123	2
gnetech\jwinberg	223	73
gnetech\kkha	482	402
gnetech\mdavidse	976	383
gnetech\mrees	1038	341
gnetech\omisje	423	140
gnetech\wlee	557	257

last month of the subject! With Bond semesters consisting of 70 weekdays and taking the average value of 500 page accesses over the semester gives a figure of seven page accesses per day. If the student figures shown in Figure 6 are accurate this means between 3 and 7 pages are viewed on each visit to the subject site.

Note that the instructor in this example, mrees, has the highest access count, which is to be expected. A detailed statistical analysis has not been performed but a simple inspection of eventual overall grades against site access counts shows the students with the highest grades always have very high page access counts.

Probably the primary use of the log data is to determine which pages are the most popular so that the most valuable resources can be identified. Other interesting data gathered shows the students' choice of operating system and Web browser on their own machines. Being information technology students, the range of different browsers is wider than normal. Fortunately, another major benefit of SharePoint is its ability to work with all popular browsers.

FUTURE TRENDS

The usefulness of SharePoint technologies to support LMS features is likely to lead to increased deployments in higher education institutions. The cost of the SharePoint Portal server and SQL Server is likely to be an obstacle for smaller installations in the first instance. A more likely scenario is the use of Windows SharePoint Services to begin a small-scale LMS deployment where no SPS or SQL Server licences are needed. A typical example of such a situation is the University of Vermont Business School (2006). Here WSS was used successfully by both staff and students as part of a much larger trial of a collection of Microsoft software including Office.

In October 2005, Blackboard and Microsoft (2005) issued a press statement indicating that they had started a project to link up the Blackboard Community Portal with SharePoint Portal server. Very soon after this announcement came the news that the Blackboard Learning for Microsoft .NET (2005) had been released which would indicate that the SharePoint/Blackboard integration when running on Windows would be made more straightforward.

The integration work is still progressing and looks likely to join the Web parts model in SharePoint with the Blackboard building block architecture. An obvious mechanism to use is the Web services for Remote Portlets protocol proposed by the OASIS Technical Committee (OASIS WSRP, 2006). The first author eagerly awaits the outcomes of this project as it will be an intelligent marriage of SharePoint and Blackboard, and will allow the author's expertise to

be leveraged into the future. Bond University eventually decided to implement Blackboard 7.0 from May 2006.

Probably the most exciting development will be the introduction of a major new design for SharePoint in the forthcoming release of Office 2007 due in early 2007. Not only will SharePoint become the heart of the Office System with even tighter Office integration, it will be partnered by the new Groove Server, which will bring full-featured shared workspaces to Office users. The many additions will make SharePoint even more attractive as an LMS, blogs, RSS feeds, and wiki pages being just a few of the examples.

CONCLUSION

Over the SPA trial, six coordinating teaching staff created nine teaching sites that were used throughout one full semester. About 15 other staff members, administrative and teaching, were allocated SPA accounts, and they experimented with SharePoint for short periods. Apart from student teaching, SPA was used for course planning and preparing educational content as well as some administrative tasks such as the quality audit.

The majority of the teaching staff reported their satisfaction with SharePoint although they were not able to carry out student surveys. It should be remembered that the use of SharePoint as an LMS reported here is very specifically as a supplement to face-to-face teaching, which is the main delivery paradigm. At Bond SharePoint was trialled against existing static Web sites. SharePoint proved to be more collaborative, interactive, and easy to populate with content. To this extent, the SPA trial was successful.

Surprisingly considering the corporate origins of SharePoint the administrative experiment with the quality audit was not a success. The intention was to present documents about the audit on a Web site, and then to survey all staff to determine that they had accessed the documents. The need to allocate new accounts for up to 400 staff was not a barrier. Despite the ease of creating the survey itself, the inability to incorporate the survey Web part into another page satisfactorily was the eventual stumbling block. Web part features became the limiting factor.

There is considerably less doubt of the usefulness of SharePoint as a collaborative intranet for designing courses by teams of teaching staff (Dain, 2003). SharePoint is also valuable for forging a community of practice and expertise as described in McFerrin, Tewson, and Wallis (2003). Such a collaborative environment encourages the sharing of ideas and exemplars.

It should not be forgotten that the major defect in SharePoint as regards its use as an LMS lies in the lack of a centralized grade book and compelling features for online testing, the submission of assessment and marking. The authors did show that a relatively small development effort is needed to start to add customized assessment features using InfoPath forms and Word smart documents. Further development of more powerful assessment Web parts would further improve SharePoint's standing as a complete LMS.

ACKNOWLEDGMENT

The authors would like to thank the other members of the SPA Management Group, Jay Forder (law) and Peter Stewart (vice-chancellors office), for their help and support during the trial period. Thanks are also due to the deans of the faculties of business, humanities, law and information technology for funding the trial. Robert Bannerman of Microsoft initiated the Spotlight on Office competition and was very supportive of our effort. Significant programming support for the Bond University Research Pane was provided by Ole Rynning, a Bond Masters student in information technology, and the authors are appreciative of his work.

REFERENCES

Barros, A., Herring, C., Hildebrandt, J., & Rees, M. (2000). *Generating command and control systems in an hour: The Microsoft way.* Paper presented at the 5th International Command and Control Research and Technology Symposium, Canberra, Australia.

Blackboard Academic Suite. (2006). Retrieved 26 March 2006, from http://www.blackboard.com/

Blackboard and Microsoft Cooperate to Integrate Administrative and Academic Collaboration Products. (2005). Retrieved 31 March 2006, from http://www.blackboard.com/company/press/release.aspx?id=769016)

Blackboard Learning System for Microsoft .NET Framework Unveiled. (2005). Retrieved 31 March 2006, from http://www.blackboard.com/company/press/release.aspx?id=633054

Dain, M. (2003). Instant Intranets with Microsoft Sharepoint team services. *World Conference on Educational Multimedia, Hypermedia, and Telecommunications 2003* (Vol. 1, pp. 631-634).

Goldberg, M. (2004). Keynote Address. The *10th Australian World Wide Web Conference*. Retrieved March 27, 2006, from http://ausWeb.scu.edu.au/aw04/papers/edited/goldberg/

Herring, C., & Rees, M. (2001). *Internet-based collaborative software development using Microsoft tools.* Paper presented at the 5th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida.

Herring, C., Rees, M., Loch, A., & Rhodes, B. (1998, 20-21 September). *Microsoft first contact: The Borg Experiment.* Paper presented at the Oz-Group'98 Australian Workshop on Computer Support for Collaboration, University of Queensland, Brisbane, Australia.

Kennedy, D. (2005). *Challenges in evaluating Hong Kong students' perceptions of Moodle.* Paper presented at ASCILITE 2005, Brisbane, Australia, (pp. 327-336).

McFerrin, K., Tewson, V., & Wallis, D. (2003). *Professional development for in-service and pre-service teachers.* Paper presented at the World Conference on E-Learning in Corp., Govt., Health., & Higher Ed.

Microsoft SharePoint Products and Technologies. (2006). Retrieved 27 March 2006, from http://www.microsoft.com/sharepoint/

Moodle Course Management System. (2006). Retrieved 29 March 2006, from http://moodle.org/.

OASIS Web Services for Remote Portlets (WSRP) Technical Committee. (2006). Retrieved 31 March 2006, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

Sakai: Collaboration and Learning Environment for Education (2006). Retrieved 27 March 2006, from http://www.sakaiproject.org/

University of Vermont Gives Business Students Real-World Technology Experience. (2006). Retrieved 31 March 2006, from http://members.microsoft.com/CustomerEvidence/Search/EvidenceDetails.aspx?EvidenceID=4515&LanguageID=1

Virtual Learning Environment. (2005). Retrieved 26 April 2005, 2005, from http://en.wikipedia.org/wiki/Virtual_learning_environment

WebCT. (2006). *A BlackBoard Company.* Retrieved March 26, 2006, from http://www.webct.com/

KEY TERMS

FAQ: Frequently asked questions.

LMS: Learning management system.

SPA: SharePoint alliance trial.

SPS: SharePoint portal server.

Web Part: Portlets supported by SharePoint.

WSS: Windows SharePoint services.

This work was previously published in Handbook of Research on Instructional Systems and Technology, edited by T. Kidd and H. Song, pp. 326-342, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 2.9
Development of Machine Learning Software for High Frequency Trading in Financial Markets

Andrei Hryshko
University of Queensland, Australia

Tom Downs
University of Queensland, Australia

ABSTRACT

Foreign exchange trading has emerged in recent times as a significant activity in many countries. As with most forms of trading, the activity is influenced by many random parameters, so that the creation of a system that effectively emulates the trading process will be very helpful. This chapter presents a novel trading system using Machine Learning methods of Genetic Algorithms and Reinforcement Learning. The system emulates trader behavior on the Foreign Exchange market and finds the most profitable trading strategy.

INTRODUCTION

In spite of many years of debate between economists and financiers, the question of whether financial markets are predictable remains open. Numerous tests with financial data have been conducted by researchers but these have tended to support both sides of the issue. In our view, the best evidence of predictability of financial markets would be the development of a strategy or an algorithm which is capable of consistently gaining a profit from the financial market. In this chapter we demonstrate that machine learning techniques are capable of performing this task.

The use of machine learning and optimization methods in finance has become a fairly common practice amongst financiers and researchers. With the continuous deregulation and increasing volatility of financial markets, competition in the financial industry is getting stronger and new techniques are being developed to provide efficient trading for financial institutions and the public.

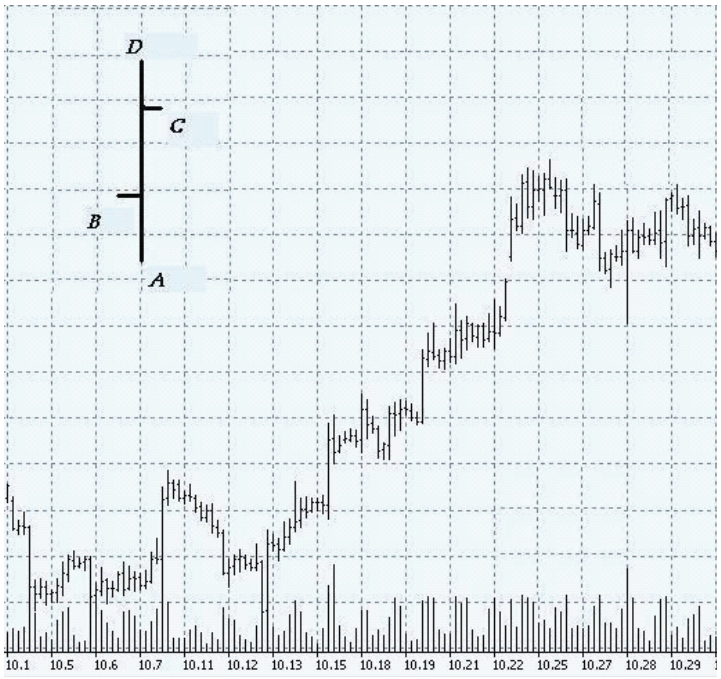
At the present time, millions of people trade in financial markets and even more wish to become involved. The main problems they face are how to trade and how to develop a profitable strategy. Usually it takes several years to become a successful trader and sometimes success remains elusive.

Trading usually takes place through a broker who provides software for the trader to buy and sell assets on a financial market. This software provides information to the trader such as current and past share prices, exchange rate, market indicators, etc. Based on this data, a trader can decide when to sell and when to buy a particular stock or currency. Choosing these actions in order to maximize profit is a difficult task, not just for beginners, but also for experienced traders. The market is constantly changing so that different rules and concepts apply in different situations. It is not uncommon that a trader's strategy that works well at a given time performs poorly two hours later. Hence the trader has to determine the times at which a strategy should be changed and to identify the changes that should be made. Another problem for the trader is that different strategies are successful for different financial markets. Thus, a strategy should be tailored to a particular situation in a particular market.

To deal with these problems, the issue of market information analysis needs to be addressed, not only theoretically but also practically. In this chapter, we describe our investigations into this issue and how they can be used to develop a software system capable of operating in the manner of a human trader.

Some related studies have previously been carried out, but the question of how to combine

Figure 1. EUR/USD prices and volumes for October, 2004



theoretical investigations with practical trading requires further attention. Existing methods (at least those in the open literature) are examined and are considered not capable of generating significant profits and therefore cannot be applied to online trading.

THE FOREIGN EXCHANGE MARKET

The Foreign Exchange (FX) Market is an interbank market that was created in 1971 when international trade began using floating rather than fixed exchange rates. The FX Market was an integration of deregulated domestic stock markets in leading countries. Since the time of its creation, currency rates have been determined based on supply and demand with regard to each currency (Carew & Slatyer, 1989). Figure 1 displays a sequence of four-hour prices and volumes for the Euro/US dollar market during October 2004.

Each element of the upper curve has the structure shown at the upper left of the figure. In this structure, points A and D indicate the lowest and highest rates achieved in a four-hour period, and points B and C indicate the opening and closing prices, respectively, for the four-hour period. The histogram-like plot along the bottom of the figure indicates the volume of transactions over each four-hour period.

It is our view that the FX market is a good basis for testing the efficiency of the prediction system because:

1. Historical data on exchange rates of currencies is readily available.
2. The FX market is easily accessible. It is open to individual investors and doesn't require large deposits.
3. The FX market provides freedom to open and close positions of any size at any time at the current market rate.
4. The FX market is open 24 hours a day, five days a week. This means that at each moment of time every time zone (London, New York, Tokyo, Hong Kong, Sydney) has dealers who can quote prices for currencies.
5. A trader can define a period of time for his position on the FX market. This means that his position is kept open until that time has elapsed.

STATISTICAL APPROACH TO TECHNICAL ANALYSIS

There are two basic approaches to market analysis: fundamental analysis and technical analysis.

Fundamental analysis focuses on the economic forces of supply and demand and determines whether prices move higher, lower or stay almost the same. It examines all relevant factors affecting the price in order to determine the intrinsic value of the market. Intrinsic value means worth based on the law of supply and demand. Technical analysis is the practice of trying to forecast market prices by examining trading patterns and comparing the shape of current charts to those from the past.

Both of these approaches to market forecasting try to solve the same problem, that is, to determine in what direction prices will move. The difference between fundamental and technical analyses is that the first one explores the causes of movements on the market and the second one explores the effect of these movements.

Fundamental analysis is generally concerned with longer-term trends than technical analysis. So given that most traders on the FX market are intra-day traders they use just technical analysis and do not take into consideration the fundamental method. The main instrument employed by technical analysts is the set of available indicators, which helps traders to discover trends, reversals of trends and other fluctuations.

An indicator makes use of a set of mathematical formulae that may, for instance, be derived from past prices or from other market data such as trade volumes. Different indicators play different roles in the analysis. Some indicators work better when the market has a strong trend, and some when the market is neutral. The Machine Learning system described here makes use of ten commonly used indicators.

The simplest indicator we use is the Moving Average (MA) that shows the average value of prices during a defined period of time. The *n*-day MA is calculated using the simple formula:

$$MA = \frac{P_1 + ... + P_n}{n}$$

where P_i is the price *i*-1 days previously.

Buy and sell signals are generated according to the behavior of moving averages in the short and longer term. A buy signal is produced when the short average crosses above the longer one. When the short average moves below, a sell signal is generated. This technique is called the *double crossover method.*

The 10 commonly used indicators that we have employed in this work are detailed in Appendix 1.

METHODOLOGY

Genetic Algorithms

Genetic Algorithms (GA) were introduced by Holland (1975) as a general model of adaptive processes and have been widely applied as an optimization technique. They were inspired by natural genetic processes and employ a population of "chromosomes" (represented by binary strings) that are analogous to the DNA of a natural organism. Unlike traditional optimization methods that require well-defined mathematical representations of the objective, GA can solve optimization problems that are far less well described mathematically. This makes them very useful for trading models, which cannot easily be expressed in terms of mathematical formulas and functions.

In the original GA formulation, a population of possible solutions is encoded as a set of bit strings (known as parameter strings), each of the same fixed length. The fitness of each string in the population is estimated and the basic GA operators are then applied as described below. This provides a second generation population whose average fitness is greater than that of the initial population. The GA operators are now applied to the second generation population and the process is repeated, generation after generation, until some stopping criterion is met. The string with maximum fitness in the final population is then selected as the solution.

In applying a GA to FX trading, each string represents a possible solution for the trader — adopt a short, long or neutral position.

Decisions made by the trader are based upon the values of a set of market indicators. These values can be incorporated into the GA bit strings as binary variables. For instance, in the case of the double-crossover method mentioned in the previous section, the indicator generates a sell signal when the short average moves below the longer one. The indicator corresponding to this signal is called "MASell" and it takes on the binary value 1 when the condition for the sell signal is met. It has the value 0 otherwise.

An example of a rule employing this and two other indicators (described in the Appendix) is shown in Table 1. This rule instructs the trader to adopt a short position (i.e., sell) and is encoded as a bit string in the table. The rule states "IF MASell = 1 OR (MomentumSell =1 AND StochasticBuy = 0) THEN adopt a short position." Thus the connectives in the table are the Boolean operators AND and OR which have binary values 1 and 0 respectively.

Note that the instruction to sell in this rule is encoded as a zero at the right-hand end of the

Table 1. A sell rule and its associated bit string

MASell	Connective	MomSel	Connective	StochBuy	Sell
1	0	1	1	0	0

Table 2. An example of an exit

RSI Buy	Connect	Mom Sell	Connect	LW Sell	Connect	PO Sell
1	0	1	0	1	1	0

bit string. Rules that instruct the trader to adopt a long position (i.e., to buy) have a "1" in this position.

Rules for adopting a short or long position are called *entry* rules because they instruct the trader to participate actively in the market. There are also exit rules under which the trader returns to a neutral position. An example is shown in Table 2. Note that there is no explicit binary value for the instruction to exit. No such value is necessary because the exit action is always the same — that is, to return to a neutral position.

The rule in Table 2 rule states "IF RSIBuy = 1 OR (MomentumSell = 1 OR (Larry WilliamsSell = 1 AND Price OscillatorSell = 0) THEN adopt a neutral position."

The rules and their binary strings are obviously significantly longer than these, when a large number of indicators have to be coded.

The rules in Tables 1 and 2 together can be considered as a *strategy*. This strategy states that if the trader enters under the rule in Table 1, the rule for exiting is the one in Table 2. Our objective is to use machine learning methods to determine the best possible strategy for given market conditions.

In a typical implementation, a population of 150 rules of each type (entry and exit) is generated randomly. Then out of these 300 rules we randomly combine 150 pairs consisting of one entry rule and one exit rule. This therefore gives us 150 trading strategies. These strategies are ranked according to their profitability and are then stochastically chosen to participate in the creation of a new population. Those strategies with greater profitability (or *fitness*) are more likely to be selected to participate. The process of generating a new population employs rules that are analogous to processes involved in natural reproduction. The exchange of genetic material is reflected in the *crossover* operation which combines a pair of rules to form two "children" by swapping sub-strings. A form of *mutation* is also occasionally applied in order to broaden the mix of bit strings. This is implemented by simply inverting bit values. A low probability is used for this so that bit values are only infrequently inverted.

Crossover and mutation are applied only to rules of the same type (entry or exit). If rules derived in this way are unique (differing from all other rules in the population), they are used to replace low-ranked rules. This is done in such a way that the number of rules in the population remains constant. The process is continued until a stopping criterion is met upon which the best pair of rules (the best strategy) is chosen to be the output of the genetic algorithm. The best strategy is the one that gives maximum profitability.

Reinforcement Learning

Another Machine Learning framework that is helpful in the implementation of a financial market model is reinforcement learning. The general reinforcement learning problem addresses the following: an agent must explore its environment

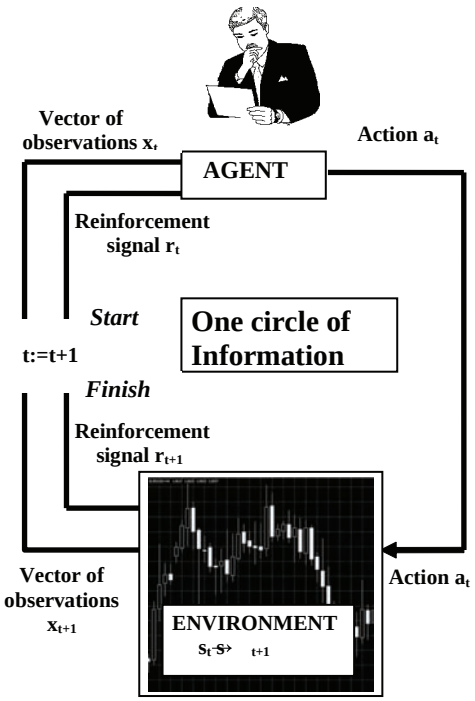
and make decisions in different situations based on incomplete knowledge about this environment. The only feedback that the agent receives from the environment is a scalar reinforcement signal which is positive if its actions are beneficial and negative otherwise. The objective of the agent is to choose its actions so as to increase the long-term sum of the reinforcement signals (Kaelbling & Littman, 1996). Besides the reinforcement signal the agent also receives information on the current state of the environment (in the form of a vector of observations). On the FX market a trader (machine or human) has insufficient knowledge about the environment to choose the times at which buy and sell decisions should be made in order to maximize profit. The only information available is the gain (positive or negative) generated by trading decisions and this provides the reinforcement signal that drives our system.

Figure 2 illustrates the principles of reinforcement learning (RL). At time *t* the agent receives inputs $\boldsymbol{x}_t$ (a vector of observations) and r_t (a reinforcement signal) and based on these it chooses an action, a_t. The action a_t changes the state of the environment from s_t to s_{t+1}. The process then repeats.

For our software implementation, we have developed an RL-engine based on the Q-learning algorithm proposed by Watkins (1989) for partially observable Markov decision processes. The Q-learning algorithm, which can be used online, was developed for the optimization of a strategy based upon experience gained from the unknown environment.

In general, the Q-learning algorithm works as follows. The value Q(*s*, *a*) is defined to be the expected discounted sum of future reinforcement signals when action *a* is taken in state *s* and an optimal policy is then followed. The state *s* belongs to *S*, the discrete set of states of the environment and the action *a* belongs to the set *A* of possible agent actions. Once we have the Q(*s*, *a*) values, the optimal action from any state is the one with the highest Q-value. This means that we obtain

Figure 2. Reinforcement learning cycle



the policy by executing the action with the highest Q-value. At the first step we initialize $Q_0(s_0, a_0)$ by arbitrary numbers and improved estimates of the Q-values are then obtained from incoming signals using the following procedure:

1. From the current state s_t, select an action a_t. This takes us to the next state s_{t+1} and provides the reinforcement signal $r_{t+1.}$
2. Update $Q_t(s_t, a_t)$ based on this experience:

$$Q_{t+1}(s_t, a_t) := Q_t(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right) \quad (1)$$

   where α ($0 < \alpha \leq 1$) is a learning-rate parameter and $0 <$ ãð < 1 is the discount factor used to place more emphasis on reinforcement signals that are received earlier.
3. Go to 1.

Note that we have to store each value $Q(s,a)$ for all $s \in S$ and $a \in A$. They are stored in a table

called a Q-table. An illustration of Q-learning algorithm is given in Appendix 2.

The objective of the agent is to find the optimal policy $\pi(s) \in A$ for each state of the environment to maximize the long-run total reward. The Q-learning algorithm uses optimal Q-values $Q^*(s_t, a_t)$ for states s and actions a. The optimal Q-value function satisfies Bellman's optimality equation:

$$Q^*(s_t, a_t) := \sum_{s_{t+1}} P(s_t, a_t, s_{t+1}) \left[R(s_t, a_t, s_{t+1}) + \gamma \max_{a'} Q^*(s_{t+1}, a') \right] \quad (2)$$

where $P(s_t, a_t, s_{t+1})$ is the probability of a transition from state s_t to s_{t+1} with action a_t is taken.

$R(s_t, a_t, s_{t+1})$ is an immediate reward obtained from taking action a_t when the environment state changes s_t to s_{t+1}.

$\gamma (0 \leq \gamma \leq 1)$ is a discount factor to weight future rewards.

Given the optimal Q-values $Q^*(s,a)$ it is possible to choose the best action:

$$a^* = \arg\max_a (Q^*(s, a))$$

A major advantage of using Q-learning is that there is no need to know the transitive probabilities $P(s_t, a_t, s_{t+1})$. The algorithm can find the $Q^*(s,a)$ in a recursive manner. The Q-values are adjusted according to equation (1).

If equation (1) is repeatedly applied for each pair (s_t, a_t) and the learning rate α is gradually reduced toward 0 over time, then $Q(s,a)$ converges with probability 1 to $Q^*(s,a)$.

A HYBRID TRADING SYSTEM

Our hybrid system involves a combination of the two techniques of machine learning described in the previous section.

Combining the Two Techniques

It is important to realize that, because of the vast number of combinations of indicator values and connectives, the GA is unable to search the whole space of strategies to find the optimum. To see this, note that if we have m indicators per rule and N indicators in total, the number of possible rules is $P(N, m)*2^{m-1}$, where $P(N, m)$ is the number of permutations of N objects taken m at a time. In our system, the ten indicators we use are applied to both buying and selling, giving a total of 20 indicators in all. Our average rule length is eight indicators, so the above formula gives the approximate number of possible rules as 6.5×10^{11}.

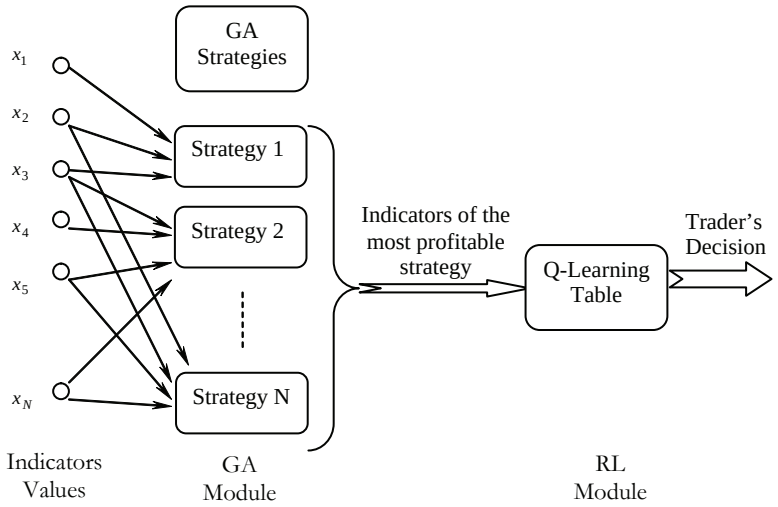
To illustrate our procedure a little further, suppose that the strategy given by Table 1 has been selected by the GA as the most profitable one. (Note that this could only occur in a market where prices are falling.) Because of the combinatorial complexity, the GA will, with very high probability, have only considered one set of instantiations for the indicators and connectives making up this rule (i.e., the ones in Tables 1 and 2). And this is where reinforcement learning comes in.

The fact that the GA identified this strategy as a profitable one shows that the indicators used in the strategy are capable of making useful market predictions. Because of this, it is worthwhile to consider the other possible instantiations of the rule values in the strategy, and this is the role of the Q-learning algorithm. Figure 3 shows the basic structure of the system. Once we have the set of the most useful indicators, they can be used to represent states of the environment and in this way we can take into account all possible combinations of the indicators.

Consider the theoretical situation where the GA module has identified the two indicators RSIBuy and CCISell as useful predictors. They provide the following set of states of the environment

$$\begin{cases} s_1 = \text{RSIBuy} = 1, \text{CCISell} = 1 \\ s_2 = \text{RSIBuy} = 1, \text{CCISell} = 0 \\ s_3 = \text{RSIBuy} = 0, \text{CCISell} = 1 \\ s_4 = \text{RSIBuy} = 0, \text{CCISell} = 0 \end{cases}$$

Figure 3. The hybrid system



At each moment of time the trader has to make a decision whether take a short, long or neutral position. Thus, the set of actions is {buy, neutral, sell}.

This set of indicators and actions gives a Q-table of the form in Table 3. The entries in this table contain Q-values that are calculated and updated using the method explained in the Reinforcement Learning section. A detailed example is given in Appendix 2.

Since a trader on the FX Market tries to maximize profit, the reinforcement signal r_t is set to the difference between portfolio values at times *t* "1 and *t*. For instance, if at time *t* "1 the trader took a long position, and then it was found that the portfolio at time *t* has a lower value than at time *t* "1, the reinforcement signal would be negative. This provides the basic mechanism for adopting long and short positions. But the third position, the neutral position, must also be properly dealt with. This requires the allocation of a threshold to avoid spurious shifts from the neutral position caused by minor market fluctuations. The major effect of this is to avoid the cost of insignificant transactions.

Table 3. Q-learning table consisting of 3 actions and 2N states where N — number of indicators

	RSIBuy=1 CCISell=1	RSIBuy=1 CCISell=0	RSIBuy=0 CCISell=1	RSIBuy=0 CCISell=0
Buy				
Neutral				
Sell				

Related Methods

Previous work with similar objectives includes Dunis, Gavridis, Harris, Leong and Nacaskul (1998) who used a genetic algorithm (GA) to optimize indicator parameters. Their trading model was based on two indicator — *viz.* the Momentum Oscillator and the Relative Strength Index (both are described in the Appendix). Their data covered only a relatively brief trading period. Many experiments were carried out and their best results were 7.1% of the annualized return for the period 20/3/1996 – 20/5/1996. This is approximately equal to the interest rate at that time and so the gains were not particularly impressive.

Yao and Tan (2000) used Neural Networks to perform technical forecasting on the FX. The only technical indicators they used were moving averages of different orders and these were employed as inputs to the neural network. The best results

achieved were 8.4% annualised return trading the USD/CHF exchange rate.

Dempster, Payne, Romahi and Thompson (2001) used a method based on an ensemble of indicators. They compared genetic algorithms and reinforcement learning to a simple linear program characterizing a Markov decision process and a heuristic. The best result achieved for trading over a 15-minute interval was around a 5% return.

Dempster and Romahi (2002) introduced a hybrid evolutionary reinforcement learning approach to constrain inputs to the RL system. The GA here aims to choose an optimal subset of indicators and then feeds them to the RL module. The annualised return gained from the implementation of this approach varied from 5% to 15% at a 15-minute trading frequency.

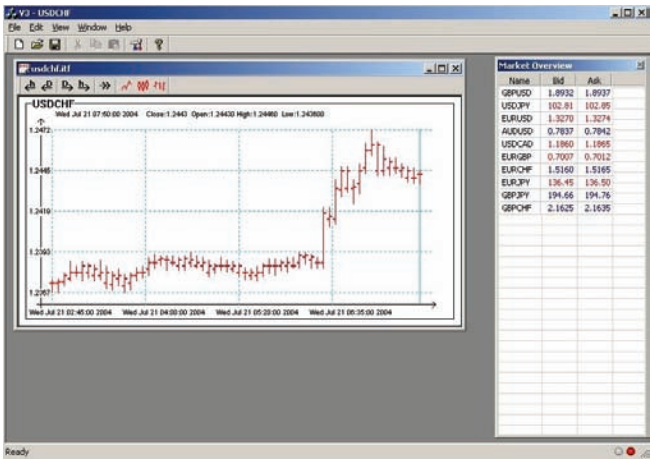
Moody and Saffell (2001) proposed the use of *recurrent* reinforcement learning to optimize risk-adjusted investment returns. Their system is based on price series returns rather than technical indicators. The recurrent RL trading system achieves an annualised 15% return on the USD/GBP exchange rate and they concluded that the recurrent RL system is superior to Q-Learning.

Note that although these results appear excellent, one must be careful in extrapolating them to real trading. Note also that Q-learning on its own is very susceptible to over-fitting but, as will become clear below, it can perform very well in a hybrid system.

THE SOFTWARE

The human trader makes use of online data to take a short, neutral or long position. Usually traders start with a set of concepts based upon indicators and then turn those concepts into a set of rules. The rule creating process requires a subjective choice to be made of which indicators to rely on and further subjectivity is needed in order to define rules for interpretation of the indicator signals. The trader then has to program the rules to create software for technical analysis. This is a difficult task, even for experienced traders. The market is changing all the time so that different rules and concepts work in different situations. Hence a trader has to determine the times at which the program should be changed, and obviously program modifications cannot be made online. Another problem for the trader is that different indicators are successful on different financial markets. For example, indicators that are profitable on one set of exchange rates could lead to catastrophic losses on another. Therefore a trader's strategy must be tailored to a particular situation. A software system that is able to adapt

Figure 4. Software for trading on the FX market



automatically to changing conditions avoids most of these problems and therefore has the potential to outperform a human in online trading. Our hybrid approach provides such an automated adaptive system and Figure 4 provides a snapshot of our system in operation.

When a trader uses our system, the software automatically connects to the broker's server, downloads data from the server and analyses the situation in the market. When using the "online trading" mode the trader does not require any knowledge of the state of the market " the software system automatically sells and buys assets and follows the market whilst updating system parameters. In the "off-line trading" mode, the trader can himself (or herself) place orders to sell or buy based upon analysis and advice provided by the system. When the software connects to the server at the first time, initial learning based on historical data occurs and then one of the modes can be selected.

The engine of the software consists of a genetic algorithm module and a reinforcement learning module based on Q-learning as described above. This system draws upon available information to determine the optimum strategy for the trader. Unlike the human trader, it is capable of working online and around the clock so its parameters are updated continuously over time to achieve the highest returns. Our system makes its decisions and predicts the future market using a combination of different market models. It recognizes the state of the market by simultaneously examining signals from each market indicator (rather than examining indicator signals one–by-one).

The fitness evaluation, crossover and mutation mechanisms are repeated until the fitness function cannot be improved any longer or a maximum number of iterations is reached. The fitness function is considered maximized if the average performance of the most profitable ten pairs of rules does not change more than 3% over several iterations. When the GA module is finished we feed the indicators from the most profitable strategy to the RL module.

Our method of choosing the indicators to feed to the RL module is an improvement on the one in Dempster & Romahi (2002) where the RL algorithm is itself employed to determine which of the indicators have the greatest fitness. This is computationally much more demanding than our method in which the fitness function is simply calculated in terms of the Sharpe ratio (Sharpe, 1966).

MAIN RESULTS

The data employed for testing our system is the intra-day Foreign Exchange (FX) rate EUR/USD. This currency pair is highly liquid since it is traded by a large number of market participants in all time zones. The data we employ relate to the period from 02 June 2002 to 31 December 2002 with a five-minute frequency 'off-line trading' mode and were obtained from the CQG Data Factory (www.cqg.com). They consist of 43,700 intra-day records with each daily record containing seven data fields. Figure 5 illustrates a sample of this data.

The first row of entries in the table in Figure 5 indicates that in the five-minute period commencing 1600 on 02 June 2002 the rate was 1 EURO = 0.9238 USD and at 1605 it was 1 EURO = 0.9331 USD. The highest and lowest rates in this period were 0.9335 USD and 0.9222 USD respectively. There were 41 trades between 1600 and 1605. A typical plot of the EUR/USD exchange rate that would provide a data entry identical to the first row of Figure 5 is shown in Figure 6.

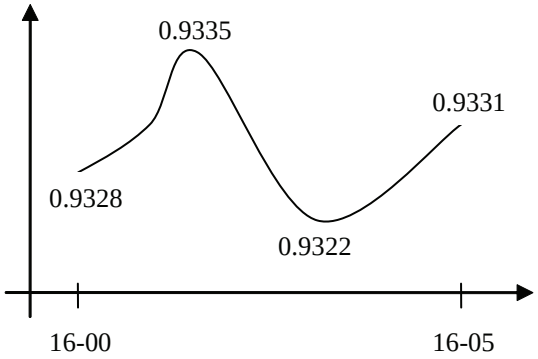
Using historical data our system learns to implement online trading. New training data is provided every five minutes and using this, the system learns to take a position - buy, sell or neutral.

Note that the software is able to work in any financial market where generally accepted market

Figure 5. Samples of 5 minute EUR/USD data extracted from 2 June 2002

Date	Time	Open	High	Low	Close	#Ticks
20020602	1600	9328	9335	9322	9331	41
20020602	1605	9324	9334	9320	9330	30
20020602	1610	9326	9334	9320	9333	32
20020602	1615	9323	9334	9320	9330	46

Figure 6. Illustration of the first row of Figure 5



rules are applicable. Transaction cost here is 2 pips per trade where a pip is the minimum unit of currency movement in the FX market. The initial trading capital was 10,000 Euros. Following training on 2.5 months of data the system achieved a profitability of about 6% on 3.5 months of test data. The annualised return achieved was therefore about 20%, which is clearly superior to the results quoted above for other approaches.

CONCLUDING REMARKS

A hybrid GA-RL system has been described that is aimed at optimizing trading strategies in the FX market. The system was trained and tested on historical data and was shown to be capable of achieving moderate gains over the tested period. Based on this system, real-time software has been designed that is capable of replacing a human trader. There are still some important features that are to be designed in future versions of the system. Some of these include stop-losses, different contract sizes, and the possibility of trading through different brokers simultaneously.

REFERENCES

Carew, E., & Slatyer, W. (1989). *Forex: The techniques of foreign exchange*. Sydney: Allen and Unwin.

Dempster, M., & Romahi, Y. (2002). Intraday FX trading: An evolutionary reinforcement learning approach. In H. Yin et al. (Eds.), *Intelligent Data Engineering and Automated Learning: Proceedings of the IDEAL 2002 International Conference* (pp. 347-358). Berlin: Springer Verlag.

Dempster, M., Payne, T., Romahi, Y., & Thompson, G. (2001). Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks, 4*(12), 744-754.

Dunis, C., Gavridis, M., Harris, A., Leong, S., & Nacaskul, P. (1998). An application of genetic algorithms to high frequency trading models: A case study. In C. Dunis & B. Zhou (Eds.), *Non-linear modelling of high frequency financial time series* (pp. 247-278). New York: Wiley.

Holland, J. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan.

Kaelbling, L., & Littman, M. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237-285.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks, 4*(12), 875-889.

Murphy, J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications.* Englewood Cliffs, NJ: Prentice Hall.

Sharpe, W. (1966). Mutual fund performance. *Journal of Business, 39*, 119-138.

Watkins, C. (1989). *Learning with delayed rewards.* PhD thesis. Cambridge University, UK.

Watkins, C., & Dayan, P. (1992). Technical note. Q-learning. *Machine Learning, 8*, 279-292.

Yao, J., & Tan, C. (2000). A case study on using neural networks to perform technical forecasting of Forex. *Neurocomputing, 34*, 79-98.

APPENDIX 1: THE INDICATORS

Besides the Moving Average (MA) indicator described in the Statistical Approach to Technical Analysis section, our system employs nine other indicators as described below. Here we add a little technical detail to the description of the MA indicator before detailing the other nine. All ten indicators provide the two signals: buy and sell. We consider standard indicator parameters that are usually used in the calculation by traders as advised in Murphy (1999).

Moving Average (MA)
The nine-day and 40-day moving averages can be written

$$MA_9(n) = \frac{1}{9}\sum_{i=0}^{9} C(n-i)$$

$$MA_{40}(n) = \frac{1}{40}\sum_{i=0}^{40} C(n-i)$$

where $C(n)$ is the latest closing price.
MABuy(n) = 1 if ($MA_9(n\text{-}1) < MA_{40}(n\text{-}1)$) AND ($MA_9(n) > MA_{40}(n)$).
MASell(n) = 1 if ($MA_9(n\text{-}1) > MA_{40}(n\text{-}1)$) AND ($MA_9(n) < MA_{40}(n)$).

Moving Average Convergence/Divergence (MACD)
The MACD is one of the most popular indicators. Two lines are used in its calculation. The MACD line is the difference between two exponentially smoothed moving averages of closing prices and responds very quickly to trend movements. The signal line is the exponentially smoothed average of the MACD line and it responds more slowly to trend movements. To calculate the MACD we need to compute 12 and 26 period Exponential Moving Averages (EMA).

$$EMA_{12}(n) = \frac{2}{12+1}C(n) + (1-\frac{2}{12+1})EMA_{12}(n-1)$$
$$EMA_{26}(n) = \frac{2}{26+1}C(n) + (1-\frac{2}{26+1})EMA_{26}(n-1)$$
$$MACD(n) = EMA_{12}(n) - EMA_{26}(n)$$
$$EMA_{12}(1) = EMA_{26}(1) = C(1)$$

The Signal Line is given by the nine-day exponentially smoothed average of the MACD line.

$$SignalLine(n) = \frac{2}{9+1}MACD(n) + (1-\frac{2}{9+1})SignalLine(n-1)\ SignalLine(1) = MACD(1)$$

Signals:
MACDBuy = 1 if (MACD(n-1) < SignalLine(n-1)) AND (MACD(n) > SignalLine(n))
MACDSell = 1 if (MACD(n-1) > SignalLine(n-1)) AND (MACD(n) < SignalLine(n))

The Stochastic (Slow Stochastic)

This is based on the observation that as prices go up, closing prices tend to be closer to the upper end of the price range and vice versa when trend goes down the closing price tends to be near the lower end of the range. The Stochastic consists of two lines: the %K line and the %D line.

$$\%K(n) = 100 * \frac{C(n) - L14(n)}{H14(n) - L14(n)}$$

where
L14(n) – the lowest low for the last 14 periods
L14(n) = min (Low(n), Low(n -1), …, Low(n - 13))
H14(n) – the highest high for the same 14 periods
H14(n) = max (High(n), High(n - 1), …, High(n - 13))
where Low and High are respective components of price bar in Figure 5.
%D is a three-period moving average of the %K

$$\%D(n) = \frac{1}{3}\sum_{i=0}^{2} \%K(n-i)$$

%Dslow is a three-period moving average of the %D

$$\%Dslow(n) = \frac{1}{3}\sum_{i=0}^{2} \%D(n-i)$$

Signals:
StochasticBuy(n) = 1 if (%D(n) < 20) AND (%Dslow(n) < 20) AND (%D(n - 1) < %Dslow(n - 1)) AND (%D(n) > %Dslow(n)).
StochasticSell(n) = 1 if (%D(n) > 80) AND (%Dslow(n) > 80) AND (%D(n - 1) > %Dslow(n - 1)) AND (%D(n) < %Dslow(n)).

Relative Strength Index (RSI)

This is considered to be a very powerful and popular indicator among traders. It is used to identify overbought and oversold market conditions.

$$RSI(n) = 100 - \frac{100}{1 + RS(n)}$$

$$RS(n) = \frac{\left[(Average\ Gain(n-1)) * 13 + Current\ Gain(n)\right]/14}{\left[(Average\ Loss(n-1)) * 13 + Current\ Loss(n)\right]/14}$$

where
Current Gain(n) = max (C(n) – C(n - 1), 0)
Current Gain(n) = max (C(n - 1) – C(n), 0)

$$Average\ Gain(n) = \frac{1}{14}\sum_{i=0}^{13} \max(C(n-i) - C(n-i-1), 0)$$

$$Average\ Loss(n) = \frac{1}{14}\sum_{i=0}^{13} \max(C(n-i-1) - C(n-i), 0)$$

Signals:
RSIBuy(n) = 1 if (RSI(n - 1) < 30) AND (RSI(n) > 30))
RSISell(n) = 1 if (RSI(n - 1) > 70) AND (RSI(n) < 70))

Commodity Channel Index (CCI)
This was designed to identify cyclical turns in exchange rates movements.
Most traders use CCI as an overbought/oversold oscillator. CCI is based upon the comparison the current price with a moving average over a selected time frame.

$$TypicalPrice(n) = \frac{C(n) + High(n) + Low(n)}{3}$$

$$SMATP(n) = \frac{1}{20}\sum_{i=0}^{19} TypicalPrice(n-i)$$

$$MeanDeviation(n) = \frac{\sum_{i=0}^{19} abs(SMATP(n) - TypicalPrice(n-i))}{20}$$

$$CCI(n) = \frac{TypicalPrice(n) - SMATP(n)}{0.015 * MeanDeviation(n)}$$

Signals:
CCIBuy(n) = 1 if (CCI(n - 1) < 100) AND (CCI(n) > 100))
CCISell(n) = 1 if (CCI(n - 1) > -100) AND (CCI (n) < -100))

Momentum Oscillator
Momentum measures velocity of price changes as opposed to the actual price level.

$$Momentum(n) = C(n) - C(n-10)$$

Signals:
MomentumBuy(n) = 1 if (Momentum(n - 1) < 0) AND (Momentum(n) > 0)
MomentumSell(n) = 1 if (Momentum(n - 1) > 0) AND (Momentum(n) < 0)

Price Oscillator
This is based upon the difference between two moving averages. The moving averages can be exponential, weighted or simple. Here, we consider the exponential moving average. Averages are calculated based on closing prices.

$$PO(n) = \frac{EMA_{10}(n) - EMA_{20}(n)}{EMA_{20}(n)}$$

where EMA_k is calculated using the same principle as for MACD calculation.
Signals:
POBuy(n) = 1 if (PO(n - 1) < 0) AND (PO(n) > 0)
POSell(n) = 1 if (PO(n - 1) > 0) AND (PO(n) < 0)

Larry Williams
This is an indicator that works similarly to the Stochastic Indicator and is especially popular for identifying overbought and oversold markets.

$$LW(n) = -100 * \frac{H14(n) - C(n)}{H14(n) - L14(n)}$$

L14(n) = min (Low(n), Low(n - 1), …, Low(n - 13)
H14(n) = max (High(n), High(n - 1), …, High(n - 13)).

Signals:
LWBuy(n) = 1 if (LW(n - 1) < -80) AND (LW(n) > -80)
LWSell(n) = 1 if (LW(n - 1) > -20) AND (LW(n) < -20)

Bollinger Bands
This is based on two trading bands placed around a moving average. Upper and lower bands are three standard deviations above and below the moving average.

$$UpperBand(n) = Average\,\mathrm{Pr}\,ice(n) + 3 * StDev(n)$$

$$LowerBand(n) = Average\,\mathrm{Pr}\,ice(n) - 3 * StDev(n)$$

$$AveragePrice(n) = \frac{1}{20}\sum_{i=0}^{19} C(n-i)$$

$$StDev(n) = \sqrt{\frac{\sum_{i=0}^{19}\left(C(n-i) - AveragePrice(20)\right)^2}{20}}$$

BBandBuy(n) = 1 if (C(n - 1) < LowerBand(n - 1)) AND (C(n) > LowerBand(n))
BBandSell(n) = 1 if (C(n - 1) > UpperBand(n - 1)) AND (C(n) < UpperLowerBand(n))

On Balance Volume
OBV can be used either to confirm the price trend or notify about a price trend reversal.

$$OBV(n) = OBV(n-1) + \frac{C(n) - C(n-1)}{abs(C(n) - C(n-1))} * V(n)$$

where V(n) is #Ticks component of the price bar in Figure 5.

The direction of OBV is more important than the amplitude. The OBV line should follow in the same direction as the price trend otherwise there is a notification about a possible reversal.

OBVBuy(n) = 1 if (OBV(n - 3) > OBV(n - 2)) AND (OBV(n - 2) < OBV(n - 1)) AND (OBV(n - 1) < OBV(n)) AND (C(n - 3) > C(n - 2)) AND (C(n - 2) > C(n - 1)) AND (C(n - 1) > C(n))
OBVSell(n) = 1 if (OBV(n - 3) < OBV(n - 2)) AND (OBV(n - 2) > OBV(n - 1)) AND (OBV(n - 1) > OBV(n)) AND (C(n - 3) < C(n - 2)) AND (C(n - 2) < C(n - 1)) AND (C(n - 1) < C(n))

APPENDIX 2: ILLUSTRATION OF Q-LEARNING

In Figure 7 we consider a real but simplified situation where a trader has two indicators for a financial market and has to make a decision based on the indicator signals. In this simplified situation, we assume that each indicator can advise either to buy or sell only. That is, we ignore the possibility of indicators advising that a neutral position be adopted. Since we have two indicators that can have two values each, we have in total four states of the environment. In any state of the environment a trader can make two decisions - either to buy a security or to sell it. Thus, writing the set of actions available in state i as A(state i), we have A(state i) = {buy, sell}for i = 1,2,3,4. After taking an action the trader makes a transition from one state of the environment to another according to a set of probabilities determined by the market situation.

The state of the environment is determined by the values of the indicators and, whatever the state, the trader must decide whether to buy or sell. To understand the meaning of Figure 7, consider the arrow at the upper-left of the figure. The rectangle attached to this arrow contains the following: "Sell, R.S. = 20, Pr = 12%." What this means is that this particular transition from state 1 to state 2 will take place if the trader, in state 1, decides to sell and that this decision will result in a transition to state 2 with a 12% probability, upon which the trader will receive a reinforcement signal of 20. Note that the decision to sell while in state 1 could also result in a transition to state 4, with probability 88%. Note also that the probabilities of these two transitions add to 100%, indicating that there is zero probability of a decision to sell when in state 1 causing a transition to state 3.

The values of the reinforcement signals and the transition probabilities in Figure 7 have been chosen arbitrarily for the purposes of this illustration. In practice, these values would change due to the non-static nature of the market, but we keep them fixed here so that the key features of Q-learning can be clearly understood. If we did not do this, the origin of some of the numbers in the example would become unclear unless we included frequent updates to Figure 7.

Suppose that the initial values of $Q_0(s, a) = 0$ for all s, a, discount rate $\gamma = 0.85$ and learning rate $\alpha = 0.15$. Also assume that the following sequence has taken place: (state 1, sell) → (state 3, buy) → (state 2, sell) → (state 3, sell) → (state 4, buy) → (state 3, buy)→ (state 2, buy) → (state 1, buy) → (state 2, sell) → (state 3, …) → … For this state-action sequence, equation (1) gives the following sequence of Q-value updates:

1) (state 1, sell, 20, state 3)

$$Q_1(state\,1,\ sell) = Q_0(state\,1,\ sell) + 0.15\left[20 + 0.85\max_a Q_0(state\,3,\ a) - Q_0(state\,1,\ sell)\right] =$$
$$0 + 0.15[20 + 0.85\max\{0,0\} - 0] = 0 + 0.15 * 20 = 3$$

2) (state 3, buy, 70, state 2)

$$Q_2(state\,3,\ buy) = Q_1(state\,3,\ buy) + 0.15\left[70 + 0.85\max_a Q_1(state\,2,\ a) - Q_1(state\,3,\ buy)\right] =$$
$$0 + 0.15[70 + 0.85\max\{0,0\} - 0] = 0 + 0.15 * 70 = 10.5$$

3) (state 2, sell, -30, state 3)

$$Q_3(state\,2,\ sell) = Q_2(state\,2,\ sell) + 0.15\left[-30 + 0.85\max_a Q_2(state\,3,\ a) - Q_2(state\,2,\ buy)\right] =$$
$$0 + 0.15\left[-30 + 0.85\max\{10.5, 0\} - 0\right] = 0 + 0.15 * (-30 + 0.85 * 10.5) = -3.16$$

4) (state 3, sell, 10, state 4)

$$Q_4(state\,3,\ sell) = Q_3(state\,3,\ sell) + 0.15\left[10 + 0.85\max_a Q_3(state\,4,\ a) - Q_3(state\,3,\ sell)\right] =$$
$$0 + 0.15\left[10 + 0.85\max\{0, 0\} - 0\right] = 0 + 0.15 * 10 = 1.5$$

So after the first four steps of the state-action sequence the Q-table looks like: (see Table 4)

Now consider the remaining values in the sequence:
(state 4, buy) → (state 3, buy) → (state 2, buy) → (state 1, buy) → (state 2, sell) → (state 3, …) → …

5) (state 4, buy, 30, state 3)

$$Q_5(state\,4,\ buy) = Q_4(state\,3,\ buy) + 0.15\left[30 + 0.85\max_a Q_4(state\,3,\ a) - Q_4(state\,4,\ buy)\right] =$$
$$0 + 0.15\left[30 + 0.85\max\{10.5, 1.5\} - 0\right] = 0 + 0.15 * (30 + 0.85 * 10.5) = 5.84$$

6) (state 3, buy, 70, state 2)

$$Q_6(state\,3,\ buy) = Q_5(state\,3,\ buy) + 0.15\left[70 + 0.85\max_a Q_5(state\,2,\ a) - Q_5(state\,3,\ buy)\right] =$$
$$10.5 + 0.15\left[70 + 0.85\max\{0, -3.16\} - 10.5\right] = 10.5 + 0.15(70 + 0.85 * 0 - 10.5) = 19.43$$

7) (state 2, buy, 50, state 1)

$$Q_7(state\,2,\ buy) = Q_6(state\,2,\ buy) + 0.15\left[50 + 0.85\max_a Q_6(state\,1,\ a) - Q_6(state\,2,\ buy)\right] =$$
$$0 + 0.15\left[50 + 0.85\max\{3, 0\} - 0\right] = 0 + 0.15 * (50 + 0.85 * 3) = 7.88$$

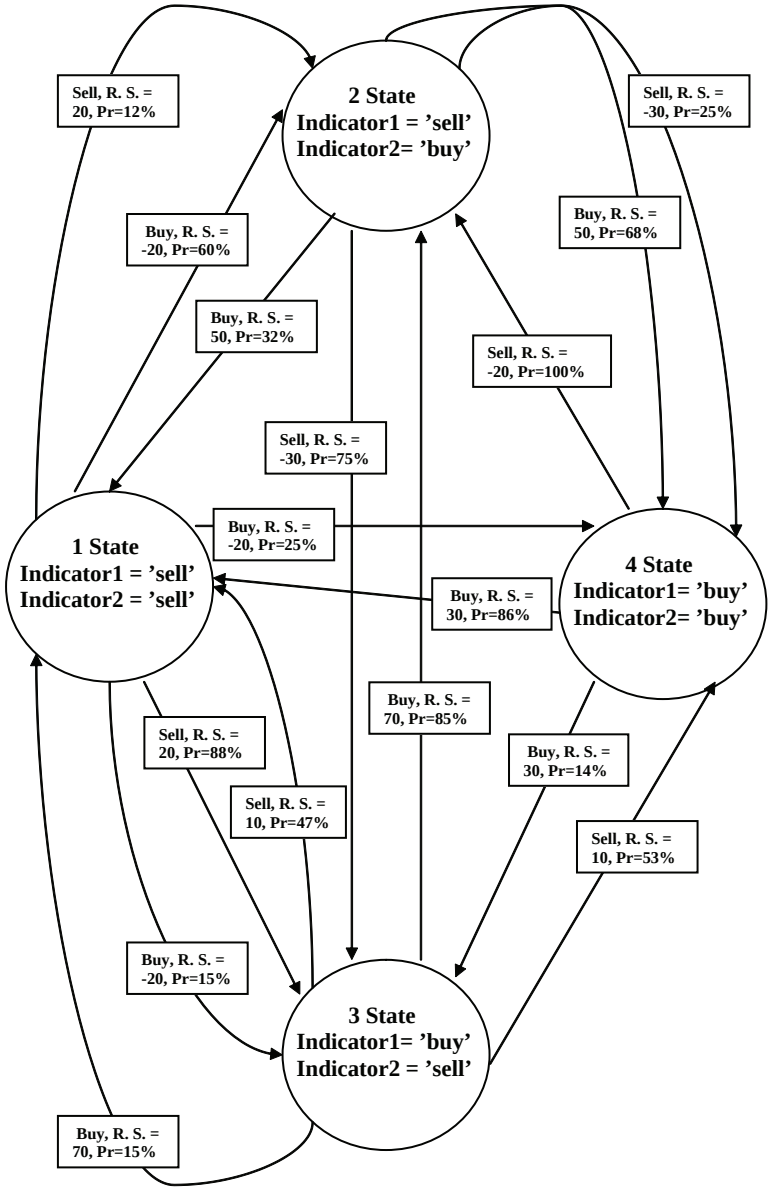
8) (state 1, buy, -20, state 2)

$$Q_8(state\,1,\ buy) = Q_7(state\,1,\ buy) + 0.15\left[-20 + 0.85\max_a Q_7(state\,2,\ a) - Q_7(state\,1,\ buy)\right] =$$
$$3 + 0.15\left[-20 + 0.85\max\{7.88, -3.16\} - 3\right] = 3 + 0.15 * (-20 + 0.85 * 7.88 - 3) = 0.55$$

Table 4. Q-learning table after 4 steps

$Q_4(s, a)$	Buy	Sell
state 1	3	0
state 2	0	-3.16
state 3	10.5	1.5
state 4	0	0

Figure 7. Example of Q-Learning



9) (state 2, sell, -30, state 3)

$$Q_9(state\,2,\ sell) = Q_8(state\,2,\ sell) + 0.15\left[-30 + 0.85 \max_a Q_8(state\,3,\ a) - Q_8(state\,2,\ sell)\right] =$$

$$-3.16 + 0.15\left[-30 + 0.85 \max\{19.43, 1.5\} - (-3.16)\right] = -3.16 + 0.15 * (-30 + 0.85 * 19.43 + 3.16)$$

$$= -4.71$$

After nine steps of the state-action sequence the Q-table looks like (see Table 5).

Table 5. Q-learning table after 9 steps

$Q_9(s, a)$	Buy	Sell
state 1	0.55	0
state 2	7.88	-4.71
state 3	19.43	1.5
state 4	5.84	0

And the optimal policy after nine steps is given by equation

$$\pi^*(s) = \arg\max_a Q_9(s, a).$$

Thus, at every state of the environment the trader will be able to determine the appropriate action. It can be shown (Watkins & Dayan, 1992) that the Q-table converges with probability 1 to the optimal set of Q-values.

This work was previously published in Business Applications and Computational Intelligence, edited by K. Voges; N. Pope, pp. 406-430, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 2.10
Architecture of an Information System for Personal Financial Planning

Oliver Braun
Saarland University, Germany

Günter Schmidt
Saarland University, Germany

ABSTRACT

We present a reliable application architecture and a corresponding system architecture of a system for personal financial planning. The application architecture is related to the business requirements, and the system architecture is related to information technology. We will present an analysis model as part of the application architecture, showing the granularity of an industry model. An intrinsic part of the proposed system architecture is the usage of Web technologies.

INTRODUCTION

The architecture of an information system influences just about everything in the process of developing an information system. The *system architecture* gives the rules and regulations by which the information system has to be constructed. The architecture's key role is to define the set of constraints placed on the design and implementation teams when they transform the requirements and analysis models into the executable system. These constraints contain all the significant design decisions and the rationale behind them.

The mission and concern of this chapter is to present a reliable *application architecture* and a corresponding *system architecture* of a system for *personal financial planning.* The application architecture is related to the business requirements, and the *system architecture* is related to information technology. We base the discussion of the business requirements on the application architecture, LISA, where four views on models are defined in Schmidt (1999):

1. The granularity of the model differing between industry model, enterprise model, and detailed model
2. The elements of the model differing between data, function, and coordination
3. The life cycle of modelling differing between analysis, design, and implementation
4. The purpose of modelling differing between problem description and problem solution

We will present an *analysis model* as part of the application architecture showing the granularity of an industry model. It contains *data*, *function*, and *coordination models* related to the purpose of modelling. The language we use to develop the analysis model is the Unified Modeling Language (UML). An intrinsic part of the proposed system architecture is the usage of Web technologies, as the global Internet and the World Wide Web are the primary enabling technologies for delivering customized decision support. We refer to the reference model as a combination of the analysis model and the system architecture. We believe that combining analysis model and system architecture could enhance the usability of the reference model.

We understand *personal financial planning* as the process of meeting life goals through the management of finances (Certified Financial Planner's (CFP) Board of Standards, 2005). Our reference model fulfils two kinds of purposes: First, the analysis model is a conceptual model that can serve financial planners as a decision support tool for the analysis of requirements. Second, system developers can map the analysis model to the system architecture at the design stage of system development. Furthermore, the reference model serves as a capture of existing knowledge in the field of IT-supported personal financial planning. The model also addresses interoperability assessments by the concept of platform-independent usage of personal financial planning tools.

STATE OF THE ART

Personal Financial Planning

The field of personal financial planning is well supplied with a lot of textbooks, among them: Böckhoff and Stracke (2003), Keown (2003), Nissenbaum, Raasch, and Ratner (2004), Schmidt (2006), and Woerheide (2002), to name just a few. Most books can be used as guides to handle personal financial problems, for example, maximise wealth, achieve various financial goals, determine emergency savings, maximise retirement plan contributions, and so forth. There are also papers in journals ranging from the popular press to academic journals. Braun and Kramer (2004) give a review on software for personal financial planning in German-speaking countries.

We will start our discussion with some definitions related to the world of personal financial planning. Certified Financial Planner's (CFP) Board of Standards defines *personal financial planning* as follows:

Definition 1: *Financial planning* is the process of meeting your life goals through the proper management of your finances. Life goals can include buying a home, saving for your child's education, or planning for retirement. Financial planning provides direction and meaning to your financial decisions. It allows you to understand how each financial decision you make affects other areas of your finances. For example, buying a particular investment product might help you pay off your mortgage faster, or it might delay your retirement significantly. By viewing each financial decision as part of a whole, you can consider its short and long-term effects on your life goals. You can also adapt more easily to life changes and feel more secure that your goals are on track. (Certified Financial Planner's (CFP) Board of Standards, 2005)

The draft ISO/DIS 22 222-1 of the Technical Committee ISO/TC 222, Personal Financial Planning, of the International Organization for Standardization, defines *personal financial planning* as follows:

Definition 2: *Personal financial planning* is an interactive process designed to enable a consumer/client to achieve their personal financial goals. (ISO/TC 222, 2004)

In the same draft, *Personal Financial Planner, consumer, client,* and *financial goals* are defined as follows:

Definition 3: A *Personal Financial Planner* is an individual practitioner who provides financial planning services to clients and meets all competence, ethics, and experience requirements contained in this standard. A *consumer* is an individual or a group of individuals, such as a family, who have shared financial interests. A *client* of a Personal Financial Planner is an individual who has accepted the terms of engagement by entering into a contract of services. A *financial goal* is a quantifiable outcome aimed to be achieved at some future point in time or over a period of time. (ISO/TC 222, 2004)

Certified Financial Planner's (CFP) Board of Standards and the Technical Committee ISO/TC 222, Personal Financial Planning, of the International Organization for Standardization define the *personal financial planning process* as follows:

Definition 4: The *personal financial planning process* shall include, but is not limited to, six steps that can be repeated throughout the client and financial planner relationship. The client can decide to end the process before having passed all the steps. The process involves gathering relevant financial information, setting life goals, examining your current financial status, and coming up with a strategy or plan for how you can meet your goals given your current situation and future plans. The financial planning process consists of the following six steps:

1. **Establishing and defining the client-planner relationship:** The financial planner should clearly explain or document the services to be provided to the client, and define both his and his client's responsibilities. The planner should explain fully how he will be paid and by whom. The client and the planner should agree on how long the professional relationship should last and on how decisions will be made.
2. **Gathering client data and determining goals and expectations:** The financial planner should ask for information about the client's financial situation. The client and the planner should mutually define the client's personal and financial goals, understand the client's time frame for results, and discuss, if relevant, how the client feel about risk. The financial planner should gather all the necessary documents before giving the advice that the client needs.
3. **Analyzing and evaluating the client's financial status:** The financial planner should analyze the client's information to assess the client's current situation and determine what the client must do to meet his goals. Depending on what services the client has asked for, this could include analyzing the client's assets, liabilities, and cash flow, current insurance coverage, investments, or tax strategies.
4. **Developing and presenting financial planning recommendations and/or alternatives:** The financial planner should offer financial planning recommendations that address the client's goals, based on the information which the client provides. The planner should go over the recommendations with the client to help the client understand them so that the client can make informed

decisions. The planner should also listen to the client's concerns and revise the recommendations as appropriate.

5. **Implementing the financial planning recommendations:** The client and the planner should agree on how the recommendations will be carried out. The planner may carry out the recommendations or serve as the client's "coach," coordinating the whole process with the client and other professionals such as attorneys or stockbrokers.
6. **Monitoring the financial planning recommendations:** The client and the planner should agree on who will monitor the client's progress towards his goals. If the planner is in charge of the process, he should report to the client periodically to review his situation and adjust the recommendations, if needed, as the client's life changes.

Systems, Models, Architectures

Models are simplifications of reality. Models exist at various levels of abstraction. A level of abstraction indicates how far removed from the reality a model is. High levels of abstraction represent the most simplified models. Low levels of abstraction have close correspondence with the system that they are modelling. An exact 1:1 correspondence is no longer a model, but rather a transformation of one system to another. *Systems* are defined as follows:

Definition 5: A *system* is a collection of connected units organized to accomplish a purpose. A system can be described by one or more models, possibly from different viewpoints. The complete model describes the whole system. (Rumbaugh, Jacobson, & Booch, 2005, p. 635)

The personal financial planning process can be supported by financial *Decision Support Systems* (DSS). In general, a Decision Support System can be described as follows:

Definition 6: A *Decision Support System* (DSS) is an interactive, flexible, and adaptable computer-based information system, specially developed for supporting the solution of a non-structured management problem for improved decision-making. It utilises data, provides an easy-to-use interface, and allows for the decision-maker's own insights. (Turban, 1995)

Continuous interaction between system and decision-makers is important (Briggs, Nunamaker, & Sprague, 1997). Interactive decision-making has been accepted as the most appropriate way to obtain the correct preferences of decision-makers (Mathieu & Gibson, 1993; Mukherjee, 1994). If this interaction is to be supported by a Web-based system, then there is a need to manage the related techniques (or models), to support the data needs, and to develop an interface between the users and the system. Advances in Web technologies and the emergence of the e-business have strongly influenced the design and implementation of financial DSS. As a result, improvement in global accessibility in terms of integration and share of information means that obtaining data from the Internet has become more convenient. Growing demand in fast and accurate information sharing increases the need in Web-based financial DSS (Chou, 1998; Dong, Deng, & Wang, 2002; Dong, Du, Wang, Chen, & Deng, 2004; Fan, Stallaert, & Whinston, 1999, 2000; Hirschey, Richardson, & Scholz, 2000; Li, 1998; Mehdi & Mohammad, 1998).

The application of DSS to some specific financial planning problems is described in several articles. In Schmidt and Lahl (1991), a DSS based on expert system technology is discussed. The focus of the system is portfolio selection. Samaras, Matsatsinis, and Zopounidis (2005) focus on portfolio selection. They propose a DSS which includes three techniques of investment analysis: Fundamental Analysis, Technical Analysis, and Market Psychology. Dong, Du, Wang, Chen, and Deng (2004) also focus on portfolio selection,

and report on the implementation of a Web-based DSS for Chinese financial markets. Zahedi and Palma-dos-Reis (1999) describe a personalised intelligent financial DSS. Zopounidis, Doumpos, and Matsatsinis (1997) give a survey on the use of knowledge-based DSS in financial management. The basic characteristic of such systems is the integration of expert systems technology with models and methods used in the decision support framework. They survey some systems applied to financial analysis, portfolio management, loan analysis, credit granting, assessment of credit risk, and assessment of corporate performance and viability. These systems provide the following features:

- They support all stages of the decision-making process, that is, structuring the problem, selecting alternative solutions, and implementing the decision.
- They respond to the needs and the cognitive style of different decision-makers based on individual preferences.
- They incorporate a knowledge base to help the decision-maker to understand the results of the mathematical models.
- They ensure objectiveness and completeness of the results by comparing expert estimations and results from mathematical models.
- Time and costs of the decision-making process is significantly reduced, while the quality of the decisions is increased.

Special emphasis on Web-based decision support is given in Bhargava, Power, and Sun (in press). Even though the above efforts in developing DSS have been successful to solve some specific financial planning problems, these do not lead to a general framework for solving any financial planning problem. There is no DSS which covers the whole process of financial planning. *Models* can be defined as follows:

Definition 7: A *model* is a semantically-complete description of a system. (Rumbaugh, Jacobson, & Booch, 2005, p. 461)

According to that definition, a model is an abstraction of a system from a particular viewpoint. It describes the system or entity at the chosen level of precision and viewpoint. Different models provide more or less independent viewpoints that can be manipulated separately. A model may comprise a containment hierarchy of packages in which the top-level package corresponds to the entire system. The contents of a model are the transitive closures of its containment (ownership) relationships from top-level packages to model elements. A model may also include relevant parts of the system's environment, represented, for example, by actors and their interfaces. In particular, the relationship of the environment to the system elements may be modelled. The primary aim of modelling is the reduction of complexity of the real world in order to simplify the construction process of information systems (Frank, 1999, p. 605). Mišić and Zhao (1999) define *reference models* as follows:

Definition 8: A *reference model* describes a standard decomposition of a known problem domain into a collection of interrelated parts, or components, that cooperatively solve the problem. It also describes the manner in which the components interact in order to provide the required functions. A reference model is a conceptual framework for describing system architectures, thus providing a high-level specification for a class of systems. (Mišić & Zhao, 1999)

Reference modelling can be divided into two groups (Fettke & Loos, 2003): Investigation of methodological aspects (e.g., Lang, Taumann, & Bodendorf, 1996; Marshall, 1999; Remme, 1997; Schütte, 1998), and construction of concrete reference models (e.g., Becker & Schütte, 2004; Fowler, 1997; Hay, 1996; Scheer, 1994). To the

best of our knowledge, there is no reference model for personal financial planning. Our reference model for personal financial planning consists of an analysis model for the system analysis of a personal financial planning system and of corresponding system architecture of an information system for personal financial planning.

Analysis Model

Surveys of reference models at the analysis stage of system development (i.e., conceptual models) are given in various papers, for example, Scholz-Reiter (1990), Marent (1995), Mertens, Holzer, and Ludwig (1996), Fettke and Loos (2003), and Mišić and Zhao (1999). For the construction of models, languages are needed. We will use the *Unified Modeling Language* (UML) for our analysis model.

Definition 9: The *Unified Modeling Language* (UML) is a general-purpose modelling language that is used to specify, visualize, construct, and document the artefacts of a software system. (Rumbaugh, Jacobson, & Booch, 2005, p. 3)

UML was developed by the Rational Software Corporation to unify the best features of earlier methods and notations. The UML models can be categorised into three groups:

- **State models** (that describe the static data structures)
- **Behaviour models** (that describe object collaborations)
- **State change models** (that describe the allowed states for the system over time)

UML also contains a few architectural constructs that allow modularising the system for iterative and incremental development. We will use UML 2.0 (Rumbaugh, Jacobson, & Booch, 2005) for our analysis model.

System Architecture

In general, *architecture* can be defined as follows:

Definition 10: An *architecture* is the organizational structure of a system, including its decomposition into parts, their connectivity, interaction mechanisms, and the guiding principles that inform the design of a system. (Rumbaugh, Jacobson, & Booch, 2005, p. 170)

Definition 11: The term *system architecture* denotes the description of the structure of a computer-based information system. The structures in question comprise software components, the externally-visible properties of those components, and the relationships among them. (Bass, Clements, & Kazman, 1998)

IBM (Youngs, Redmond-Pyle, Spaas, & Kahan, 1999) has put forth an *Architecture Description Standard* (ADS), which defines two major aspects: functional and operational. The IBM ADS leverages UML to help express these two aspects of the architecture. This standard is intended to support harvesting and reuse of reference architectures.

For a comprehensive overview of architectures, languages, methods, and techniques for analysing, modelling, and constructing information systems in organisations, we refer to the Handbook on Architectures of Information Systems (Bernus, Mertins, & Schmidt, 2005)

The reference architecture we use to derive our *system architecture* is a service-oriented architecture (SOA) proposed by the World Wide Web Consortium (W3C) (Booth, Haas, McCabe, Newcomer, Champion, Ferris, et al., 2005) for building Web-based information systems. According to the Gartner group, by 2007, service-oriented architectures will be the dominant strategy (more than 65%) of developing information systems. The architecture we propose supports

modular deployment of both device-specific user interfaces through multi-channel delivery of services and new or adapted operational processes and strategies. Service-oriented architectures refer to an application software topology which separates business logic from user interaction logic represented in one or multiple software components (services), exposed to access via well-defined formal interfaces. Each service provides its functionality to the rest of the system as a well-defined interface described in a formal mark-up language, and the communication between services is both platform- and language-independent. Thus, modularity and re-usability are ensured, enabling several different configurations and achieving multiple business goals. The first service-oriented architecture relates to the use of DCOM or Object Request Brokers (ORBs) based on the CORBA specification.

A *service* is a function that is well-defined, self-contained, and does not depend on the context or state of other services. Services are what you connect together using Web Services. A service is the endpoint of a connection. Also, a service has some type of underlying computer system that supports the connection that is offered. The combination of services, both internal and external to an organisation, makes up a service-oriented architecture. The technology of Web services is the most likely connection technology of service-oriented architectures. Web services essentially use XML to create a robust connection.

Web Services technology is currently the most promising methodology of developing Web information systems. Web Services allow companies to reduce the cost of doing e-business, to deploy solutions faster, and to open up new opportunities. The key to reach these goals is a common program-to-program communication model, built on existing and emerging standards such as HTTP, Extensible Markup Language (XML - see also Wüstner, Buxmann, & Braun, 2005), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). However, the real Business-to-Business interoperability between different organisations requires more than the aforementioned standards. It requires long-lived, secure, transactional conversations between Web Services of different organisations. To this end, a number of standards are underway (e.g., Web Services Conversation Language (WSCL), and the Business Process Execution Language for Web Services (BPEL4WS)). With respect to the description of services' supported and required characteristics, the WS-Policy Framework is under development by IBM, Microsoft, SAP, and other leading companies in the area. On the other hand, no other standard languages and technologies have been proposed for composing services and modelling transactional behaviour of complex service compositions. Such proposals include the Unified Transaction Modeling Language (UTML) and the SWORD toolkit.

Requirements for a Reference Model for Personal Financial Planning

Our reference model for personal financial planning should fulfil the following requirements concerning the *input*, *data processing*, and *output*: The input has to be collected in a complete (that is, adequate to the purpose) way. The *input* data have to be collected from the client and/or from institutions that are involved in this process. Such institutions may be Saving Banks, Mortgage Banks, Life Insurance Companies, Car Insurance Companies, Providers of Stock Market Quotations, Pension Funds, and so forth. This stage is time-consuming, and the quality of the financial plan is dependent on the completeness and correctness of the collected input data. *Data processing* (i.e., data evaluation and data analysis) have the task to process the input data in a correct, individual, and networked way. Correct means that the results have to be precise and error-free according to accepted methods

of financial planning. Individual means that the concrete situation with all its facets has to be centred in the planning, and that it is forbidden to make generalisations. Networked means: All of its data's effects and interdependencies have been considered and structured into a financial plan. The *output* has to be coherent and clear.

Analysis Model

The main task of an analysis model for personal financial planning is to help financial planners to consult individuals to properly manage their personal finances. Proper personal financial planning is so important because many individuals lack a working knowledge of financial concepts and do not have the tools they need to make decisions most advantageous to their economic well-being. For example, the Federal Reserve Board's Division of Consumer and Community Affairs (2002) stated that "financial literacy deficiencies can affect an individual's or family's day-to-day money management and ability to save for long-term goals such as buying a home, seeking higher education, or financial retirement. Ineffective money management can also result in behaviours that make consumers vulnerable to severe financial crises".

Furthermore, the analysis model should fulfil the following requirements:

- **Maintainability** (the analysis model should be understandable and alterable)
- **Adaptability** (the analysis model should be adaptable to specific user requirements)
- **Efficiency** (the analysis model should solve the *personal financial planning* problem as fast and with the least effort possible)
- **Standards-oriented** (the analysis model should be oriented at ISO / DIN /CFP- board standards; it should fulfil, for example, the CFP financial planning practice standards (Certified Financial Planner's (CFP) Board of Standards, 2005)

System Architecture

An architecturally-significant requirement is a system requirement that has a profound impact on the development of the rest of the system. Architecture gives the rules and regulations by which the software has to be constructed. The architect has the sole responsibility for defining and communicating the system's architecture. The architecture's key role is to define the set of constraints placed on the design and implementation teams when they transform the requirements and analysis models into the executable system. These constraints contain all the significant design decisions and the rationale behind them. Defining an architecture is an activity of identifying what is architecturally-significant and expressing it in the proper context and viewpoint.

The system architecture for an IT-based personal financial planning system should fulfil the following requirements:

- **Maintainability** (the architecture should be understandable and alterable)
- **Adaptability** (the architecture should be adaptable to specific user requirements)
- **Efficiency** (the architecture should deliver a framework that allows a Personal Financial Planner to solve *personal financial planning* problems as fast and with the least effort possible, for example, by automatic data gathering)
- **Standard-oriented** (the architecture should be oriented at well-established standards, for example, service-oriented architectures)
- **Reliability and robustness** (the architecture should support a reliable and robust IT-system)
- **Portability** (the architecture should support a platform-independent usage of the personal financial planning system)

In the following sections, we describe a reference model for personal financial planning which

meets the requirements specified in this section. The whole reference model is elaborated in more detail by Braun (2006b).

ARCHITECTURE OF A RELIABLE WEB APPLICATION FOR PERSONAL FINANCIAL PLANNING

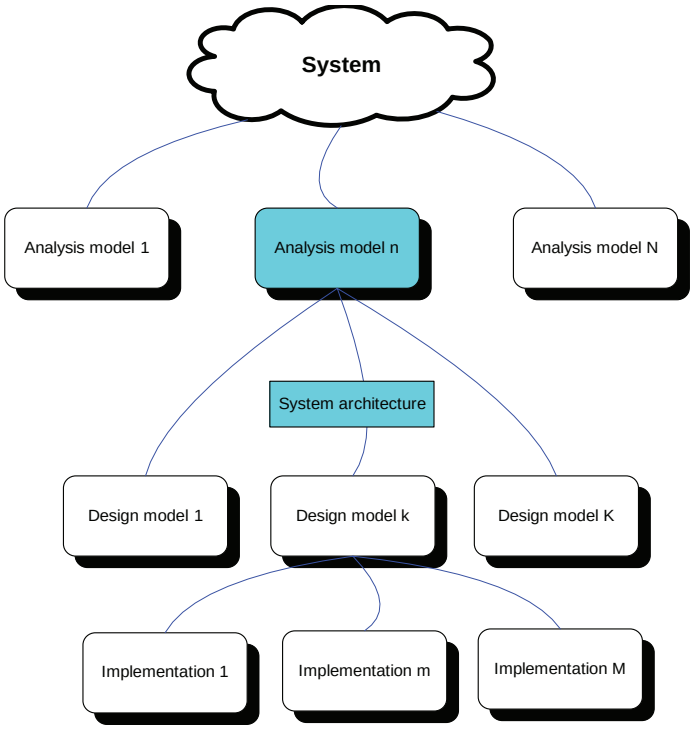
Our architecture for personal financial planning consists of two parts (see Figure 1):

1. **Analysis model** as part of the application architecture, which can help financial planners to do personal financial planning; and
2. **System architecture**, which can help system developers to develop logical models at the design stage of system development.

Our reference model for personal financial planning has two purposes: First, the analysis model can help financial planners to do personal financial planning in a systematic way. Second, at the design stage of system development, system developers can take the analysis model produced during system analysis and apply the system architecture to it. An intrinsic part of our reference model at the architecture level is the usage of Web technologies, as Web technologies have already and will strongly influence the design and implementation of financial information systems in general.

Design starts with the analysis model and the software architecture document as the major inputs (q.a. here and in the following Conallen, 2002). Design is where the abstraction of the business takes its first step into the reality of software. At design level, the analysis model is refined such that it can be implemented with the components that obey the rules of the architecture. As with analysis, design activities revolve around the class and interaction diagrams. Classes become more defined, with fully qualified properties. As this happens, the level of abstraction of the class

Figure 1. Analysis model and system architecture



shifts from analysis to design. Additional classes, mostly helper and implementation classes, are often added during design. In the end, the resulting design model is something that can be mapped directly into code. This is the link between the abstractions of the business and the realities of software.

Analysis and design activities help transform the requirements of the system into a design that can be realized in software. Analysis begins with the use case model, the use cases and their scenarios, and the functional requirements of the system that are not included in the use cases. The analysis model is made up of classes and collaborations of classes that exhibit the dynamic behaviours detailed in the use cases and requirements.

The analysis model and the design model are often the same artifact. As the model evolves, its elements change levels of abstraction from analysis to detailed design. Analysis-level classes represent objects in the business domain. Analysis focuses on the functional requirements of the system, ignoring the architectural constraints of the system. Use case analysis comprises those activities that take the use cases and functional requirements to produce an analysis model of the system. Analysis focuses on the functional requirements of the system, so the fact that some or all of the system will be implemented with Web technologies is beside the point. Unless the functional requirements state the use of a specific technology, references to architectural elements should be avoided. The analysis model is made up of classes and collaborations of classes that exhibit the dynamic behaviours detailed in the use cases and the requirements. The model represents the structure of the proposed system at a level of abstraction beyond the physical implementation of the system. The classes typically represent objects in the business domain, or problem space. The level of abstraction is such that the analysis model could be applied equally to any architecture. Important processes and objects in the problem space are identified, named, and categorised during analysis. Analysis focuses on the functional requirements of the system, ignoring the architectural constraints of the system. The emphasis is on ensuring that all functional requirements, as expressed by the use cases and other documents, are realised somewhere in the system. Ideally, each use case is linked to the classes and packages that realise them. This link is important in establishing the traceability between requirements and use cases and the classes that will realise them. The analysis model is an input for the design model and can be evolved into the design model.

SYSTEM ARCHITECTURE

Architecture influences just about everything in the process of developing software. Architecture gives the rules and regulations by which the software has to be constructed. Figure 2 shows the general architecture of our prototype FiXplan (IT-based Personal Financial Planning), and the data flow between the *User Interfaces*, the *Application Layer*, and the *Sources* (Braun & Schmidt, 2005).

Basically, the system contains the following components:

- **Clients:** Used by Personal Financial Planners or clients to access our main financial planning system and our Web Services; User Interfaces may be Web browsers such as the Internet Explorer or Mozilla, or applications such as Microsoft Excel.
- **Server:** As described in detail in the analysis model; the Web Services Toolbox provides useful tools for personal financial planning.
- **Sources:** Used as a repository for the tools of the personal finance framework.

The three-tier architecture of FiXplan supports modular deployment of both device-specific user

interfaces through multi-channel delivery of services and new or adapted operational processes and strategies. All connectivity interfaces are based on standard specifications.

Reliable Messaging

One of the first attempts to deliver guaranteed and assured delivery for Web Services was Reliable HTTP (HTTPR) from IBM. HTTPR aimed to provide reliable message delivery for Web Services, providing a “send and forget” facility through a guaranteed messaging model. It provided guaranteed delivery because it automatically retried requests if a link or a server was down. However, "OASIS WS-Reliability and WS-ReliableMessaging proposed by IBM, BEA, TIBCO, and Microsoft (submitted to the W3C) are the new competing protocols for reliable messaging" (Watters, 2005, p.24).

At the client side, the Web browser and PHP-scripts handle the user interface and the presentation logic. At the server side, the Web server gets all *http* requests from the Web user and propagates the requests to the application server, which implements the business logic of all the services for personal financial planning. At the database server side, transactional or historical data of the day-to-day operations are stored in the system database by RDBMS. The application server sends the query to the database server and gets the result set. It prepares the response after a series of processes and returns to the Web server to be propagated back to the client side. A registered user can log in and pick an existing session or create a new session of his preference, such as a level of user protection, access rights, and level of complexity. Based on the three-tier structure, FiXplan can run on both Internet and intranet environments. The user can, for example, use a Web browser to access a Web server through HTML language and HTTP protocol. The kernel of the system is placed on the web server. FiXplan uses a collection of Web Services. These services communicate with each other. The communication can involve either simple data passing, or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Figure 3 illustrates a basic *service-oriented architecture*. It shows a service consumer below, sending a service request message to a service provider above. The service provider returns a response message to the service consumer. The

Figure 2. General three-tier architecture of FiXplan

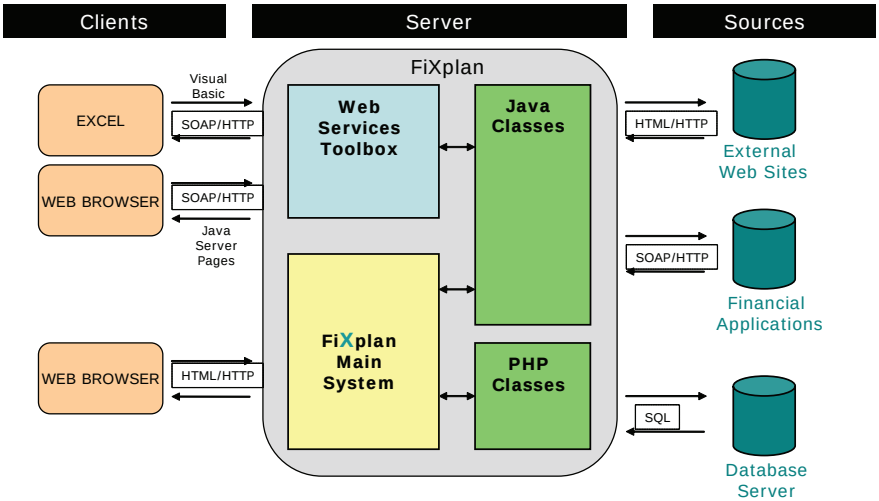
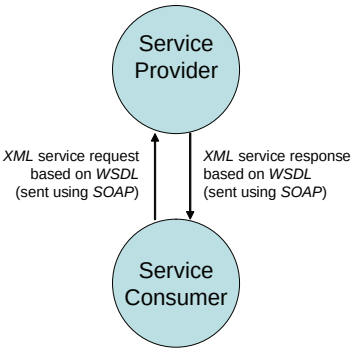


Figure 3. Basic service oriented architecture



request and subsequent response connections are defined in some way that is understandable to both the service consumer and the service provider. A service provider can also be a service consumer.

All the messages shown in the above figure are sent using SOAP. SOAP generally uses HTTP, but other means of connection such as Simple Mail Transfer Protocol (SMTP) may be used. HTTP is the familiar connection we all use for the Internet. In fact, it is the pervasiveness of HTTP connections that will help drive the adoption of Web services. SOAP can be used to exchange complete documents or to call a remote procedure. SOAP provides the envelope for sending Web Services messages over the intranet/Internet. The envelope contains two parts: an optional header providing information on authentication, encoding of data, or how a recipient of a SOAP message should process the message; and the body that contains the message. These messages can be defined using the WSDL specification. WSDL uses XML to define messages. XML has a tagged message format. Both the service provider and service consumer use these tags. In fact, the service provider could send the data shown at the bottom of this figure in any order. The service consumer uses the tags and not the order of the data to get the data values.

FiXplan is both Web Service Provider and Web Service Consumer, as shown in Figure 4.

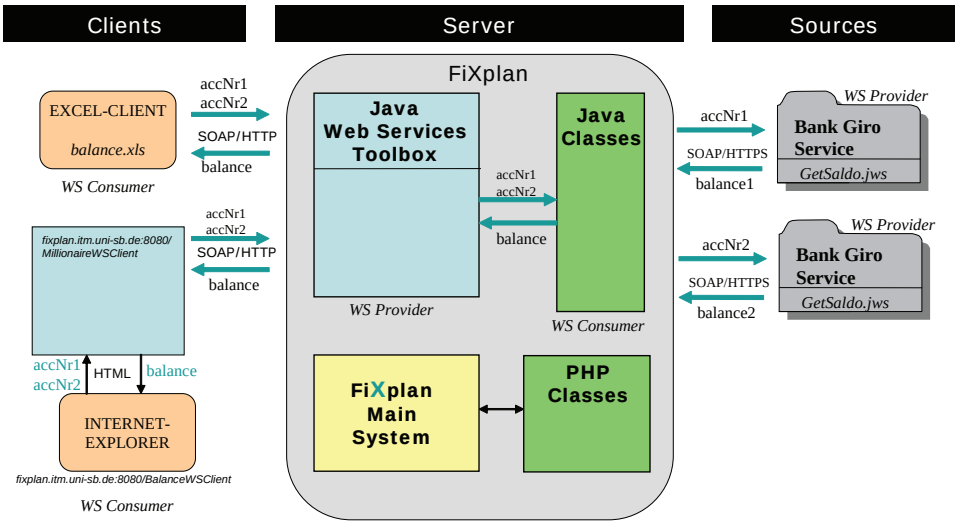
In this example, a browser (e.g., INTERNET EXPLORER) requests a static Web page with a form from a Web server (e.g., TOMCAT). After submitting the form, the Web page containing the desired results will be dynamically generated using Java Server Pages (JSP). In order to build up the Web page from JSP, the JSP calls a special Java method in a proxy class. As a result of this call, a SOAP message is sent to the corresponding Web service of the JAVA WEB SERVICES TOOLBOX. The Web service extracts the input data (in our example *accNr1, accNr2*) needed to compute the output data (in our example *balance*) from the received SOAP message. Because of the fact that our Web services are written in Java, looking for an answer means calling one of the methods of our JAVA CLASSES.

The design allows the Web service to be a consumer of another Web service: One of the Java classes called by the Web service to calculate the output data (in our example *balance*) is a proxy class which sends SOAP messages to external Web services (in our example a BANK GIRO SERVICES) and receives the results from these Web services (in our example *balance1* and *balance2*). In our example, this Java class computes also the sum of *balance1* and *balance2*.

The output data of the Java method (in our example *balance*) is packed in a new SOAP message and sent back. On the client side, the searched result is extracted from the SOAP message and given to the JSP by the proxy class. Finally, the JSP code includes the result in the Web page which is sent back to the browser.

If you have an Excel client, you have an Excel sheet instead of a Web page. When you want to calculate a result, you call a Visual Basic function which corresponds to the JSP in the example above. The Visual Basic function uses a Microsoft object which does the same as the proxy class: sending a SOAP message to the server and receiving the answer. The result is given to the Visual Basic function which writes it in the Excel sheet.

Figure 4. Service provider and service consumer



ANALYSIS MODEL AS PART OF THE APPLICATION ARCHITECTURE

The *analysis model* of our reference model for *personal financial planning* consists of the following parts: *Use Cases* build the dynamic view of the system, and *class diagrams* build the structural view of the system. Both are combined in the behavioural view of the system with *activity diagrams* and *sequence diagrams.*

Dynamic View on the System: Use Cases

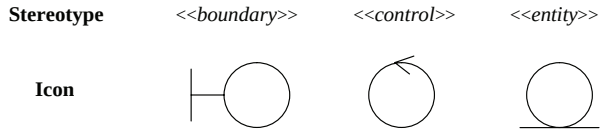
Use cases, *use case models*, and *use case diagrams* are defined as follows:

Definition 12: A *use case* is the specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside objects to provide a service of value. A use case is a coherent unit of functionality provided by a classifier (a system, subsystem, or class) as manifested by sequences of messages exchanged among the system and one or more outside users (represented as actors), together with actions performed by the system. The purpose of a use case is to define a piece of behaviour of a classifier (including a subsystem or the entire system), without revealing the internal structure of the classifier. A *use case model* is a model that describes the functional requirements of a system or other classifier in terms of use cases. A *use case diagram is a diagram* that shows the relationships among actors and use cases within a system. (Rumbaugh, Jacobson, & Booch, 2005, pp. 668-677)

Structural View on the System: Class Diagrams

The hierarchy of the dynamic view of the system (use cases) may provide a start but usually falls short when defining the *structural view* of the system (classes). The reason is that it is likely that certain objects participate in many use cases and logically cannot be assigned to a single use case package. At the highest level, the packages are often the same, but at the lower levels of the hierarchy, there are often better ways to divide the packages. Analysis identifies a preliminary mapping of required behaviour onto structural

Figure 5. Structural elements



elements, or classes, in the system. At the analysis level, it is convenient to categorise all discovered objects into one of three stereotyped classes: «*boundary*», «*control*», and «*entity*», as suggested by Jacobson, Christerson, Jonsson, and Overgaard (1992); see Figure 5.

Definition 13: *Boundary classes* connect users of the system to the system. *Control classes* map to the processes that deliver the functionality of the system. *Entity classes* represent the persistent things of the system, such as the database. (Jacobson, Christerson, Jonsson, & Overgaard, 1992)

Entities are shared and have a life cycle outside any one use of the system, whereas control instances typically are created, executed, and destroyed with each invocation. Entity properties are mostly attributes, although operations that organise an entity's properties can also often be found on them. Controllers do not specify attributes. Initially, their operations are based on verb phrases in the use case specification and tend to represent functionality that the user might invoke. In general, *class diagrams* can be defined as follows:

Definition 14: A *class diagram* is a graphic representation of the static view that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. (Rumbaugh, Jacobson & Booch, 2005, p. 217)

Use Case Realisation: Sequence and Activity Diagrams

Construction of behavioural diagrams that form the model's use case realisations help to discover and elaborate the analysis classes. A use case realisation is an expression of a use case's flow of events in terms of objects in the system. In UML, this event flow is expressed with *sequence diagrams.*

Definition 15: A *sequence diagram* is a diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in an interaction and the sequences of messages exchanged. (Rumbaugh, Jacobson & Booch, 2005, p. 585)

A sequence diagram can realise a use case, describing the context in which the invocation of the use case executes. The sequence diagram describes the objects that are created, executed, and destroyed in order to execute the functionality described in the use case. It does not include object relationships. Sequence diagrams provide a critical link of traceability between the scenarios of the use cases and the structure of the classes. These diagrams can express the flow in a use case scenario in terms of the classes that will implement them.

An additional diagram that is useful for mapping analysis classes into the use case model is the *activity diagram*.

Definition 16: An *activity diagram* is a diagram that shows the decomposition of an activity into

its constituents. (Rumbaugh, Jacobson, & Booch, 2005, p. 157)

Activities are executions of behaviour. Activities are behavioural specifications that describe the sequential and concurrent steps of a computational procedure. Activities map well to *controller classes*, where each controller executes one or more activities. We create one activity diagram per use case.

Example Models

The basic analysis *use case* model (see Figure 6) is oriented to the Certified Financial Planner's (CFP) Board of Standards and ISO/TC 222 definition of the *personal financial planning* process (see Definition 4).

In the following, we will describe the subsystem *Core Personal Financial Planning* in detail. *Core Personal Financial Planning* includes three use cases, *determining the client's financial status*, *determining a feasible to-be concept*, and *determining planning steps*, that are elaborated in more detail in the following sections.

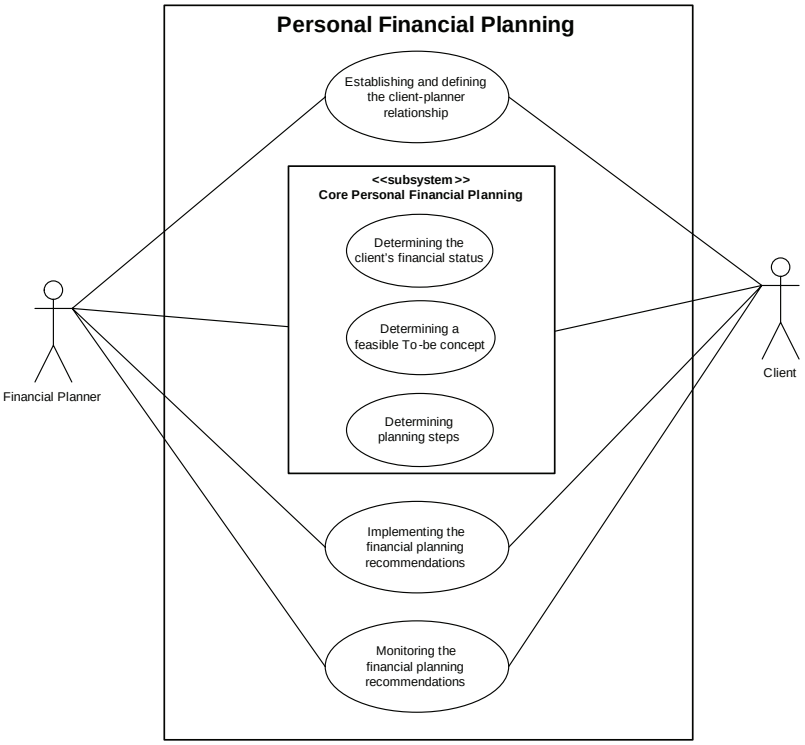
Determining the Client's Financial Status

Determining the client's financial status means answering the question "*How is the current situation defined?*" We will pick the following scenarios as the main success scenarios.

Determining the client's financial status includes:

1. **Data gathering:** The Financial Planner asks the client for information about the client's financial situation. This includes information about *assets*, *liabilities*, *income*, and *expenses*.
2. **Data evaluation:** The Financial Planner uses two financial statements: *balance sheet* and *income statement*.

Figure 6. Basic use case model



3. **Data analysis:** The Financial Planner analyses the results from balance sheet and income statement.

He can investigate, for example, the state of the provisions and pension plan, risk management, tax charges, return on investments, and other financial ratios.

In the following, the scenario is described in more detail:

- **Data gathering—assets and liabilities:** The first category of assets is *cash*, which refers to money in accounts such as checking accounts, savings accounts, money market accounts, money market mutual funds, and certificates of deposit with a maturity of less than one year. The second category of assets is fixed assets, fixed-principal assets, fixed-return assets, or debt instruments. This category includes investments that represent a loan made by the client to someone else. Fixed-return assets are primarily corporate and government bonds, but may also be mortgages held, mutual funds that invest primarily in bonds, fixed annuities, and any other monies owed to the client. The third category of assets is equity investments. Examples of equity investments are common stock, mutual funds that hold mostly common stock, variable annuities, and partnerships. The fourth category of assets is the value of pensions and other retirement accounts. The last category of assets is usually personal or tangible assets, also known as real assets. This category includes assets bought primarily for comforts, like home, car, furniture, clothing, jewellery, and any other personal articles with a resale value. *Liabilities* are debts that a client currently owes. Liabilities may be charge account balances, personal loans, auto loans, home mortgages, alimony, child support, and life insurance policy loans. In general, assets are gathered at their market value, and debts are gathered at the amount a client would owe if he were to pay the debt off today.
- **Data gathering—income and expenses:** *Income* includes wages and salaries, income from investments such as bonds, stocks, certificates of deposit, and rental property, and distributions from retirement accounts. *Expenses* may be costs for housing, food, and transportation. Expenses can be grouped by similarity; for example, one can choose insurance premiums as a general category, and then include life insurance, auto insurance, health insurance, homeowner's or renter's insurance, disability insurance, and any other insurance premiums one pays in this general category. The acquisition of personal assets (buying a car) is also treated as an expense.
- **Data evaluation—balance sheet:** A *balance sheet* has three sections. The first section is a statement of *assets* (things that one owns). The second section is a statement of *liabilities* (a list of one's debts). The third section is called *net worth*. It represents the difference between one's assets and one's liabilities. Businesses regularly construct balance sheets following a rigorous set of accounting rules. Personal balance sheets are constructed much less formally, although they should follow some of the same general rules incorporated into business balance sheets. The assets on a personal balance sheet may be placed in any order. The most common practice is to list assets in descending order of *liquidity*. Liquidity refers to an asset's ability to be converted to cash quickly with little or no loss in value. Liabilities are normally listed in order of increasing maturity. The *net worth* on a balance sheet is the difference between the total value of assets owned and the total amount of liabilities owed. Net worth is the measure of a person's financial wealth.

- **Data evaluation—income statement:** Where the personal financial balance sheet is like a snapshot of your net worth at a particular point in time, an *income statement* is a statement of income and expenses over a period of time. The time frame is usually the past year. The first section of an income statement is the *income*. The second section is a statement of *expenses*. Expenses are usually listed in order of magnitude, with the basic necessities being listed first. The third section is a summary that represents the *contribution to savings*. Contribution to savings is the difference between the total of income and the total of expenses. A positive contribution to savings would normally show up in the purchase of investments, a reduction in debt, or some combination of the two. A negative contribution to savings shows up in the liquidation of assets, an increase in debt, or some combination of the two.
- **Data analysis:** The *net worth* number is significant in and of itself, since the larger the net worth, the better off a person is financially; however, the balance sheet and the income statement contain even more useful information that a financial planner should consider. The traditional approach for analysing financial statements is the use of *financial ratios*. Financial ratios combine numbers from the financial statements to measure a specific aspect of the personal financial situation of a client. Financial ratios are used:

  1. To measure changes in the quality of the financial situation over time
  2. To measure the absolute quality of the current financial situation
  3. As guidelines for how to improve the financial situation

  We will group the ratios into the following eight categories: liquidity, solvency, savings, asset allocation, inflation protection, tax burden, housing expenses, and insolvency/credit (DeVaney, Greninger, & Achacoso, 1994; Greninger, Hampton, & Kitt, 1994). For example, the Solvency Ratio is defined as:

  Solvency Ratio = Total Assets / Total Liabilities

  and the Savings Rate is defined as:

  Savings Rate = Contribution to Savings / Income

Determining a Feasible to-be Concept

The use case scenario is as follows.

Determining a feasible to-be concept includes:

1. **Data gathering and feasibility check:** The financial planner asks for information about the client's future financial situation. This includes information about the client's *future incomes and expenses* and the client's assumption on the *performance* of his assets. Future incomes and expenses are determined by *life-cycle scenarios.*
   The financial planner asks the client for his *requirements* on a *feasible* to-be concept.
   A to-be concept is feasible if and only if all requirements are fulfilled at every point in time.
2. **Data evaluation:** The financial planner uses two financial statements: (planned) *balance sheets* and (planned) *income statements.*
3. **Data analysis:** The financial planner analyses the results from balance sheets and income statements. He can investigate, for example, the state of the foresight and pension plan, risk management, tax charges, return on investments, and other financial ratios.

Data evaluation and data analysis have to be done in a very similar way as in determining the client's financial status. In this section, we will concentrate our discussion at the first step: *Data gathering.*

Future incomes and expenses are determined by *financial goals*. Financial goals are not just monetary goals, although such goals are obvious and appropriate. Goals can be short-term, intermediate-term, and long-term.

Short-term goals refer to the next twelve months. What does a client want his personal balance sheet to look like in twelve months? What would he like to do during the coming year: Where would he like to spend his vacation? How much would he like to spend on the vacation? What assets, such as a new computer or a new car, would he like to acquire?

Intermediate goals are usually more financial in nature and less specific in terms of activities and acquisitions. As one thinks further into the future, one cares more about having the financial ability to do things and less about the details of what one will do. Most intermediate goals would likely include more than just wealth targets. For example, a client may target a date for buying his first home. Intermediate goals often address where one wants to be in 5 or 20 years.

For most people, *long-term goals* include their date of retirement and their accumulated wealth at retirement. What sort of personal balance sheet does a client want to have at retirement?

The *goals* of a system for personal financial planning consist in helping a personal financial planner to handle the financial situation of a client in a way that the client can maximise his/her quality of life by using money the best way for him/herself. Goals are about the management, protection, and ultimate use of wealth.

There is a strong propensity to associate wealth or standard of living with quality of life. Wealth is normally defined as assets owned less any outstanding debts. Standard of living is measured in terms of annual income. For some people, wealth or standard of living may be the correct measure of their quality of life. For many, that is not the case. First, many people give away substantial amounts of wealth during their lifetimes (places of worship, charities, non-profit organisations such as alma maters). People give gifts because they derive pleasure from it or because they have a sense of obligation. Second, many people hold jobs noted for low pay (religious professions such as priests) and earn low pay compared with what their level of educational training could earn them in other professions. A lot of people pass up lucrative corporate incomes in order to be self-employed. Some people seek jobs with no pay. Full-time, stay-at-home parents are examples of people who accept unpaid jobs based on the benefits they perceive from managing the household. Obviously, many people regularly make choices that are not wealth- or income-maximising, and these choices are perfectly rational and important to their lives. Proper financial planning seeks only to help people make the most of their wealth and income given the non-pecuniary choices they make in their lives, not just maximise their wealth and income.

Life-cycle scenarios (Braun, 2006a) are, for example, applying for a personal loan, purchasing a car, purchasing and financing a home, auto insurance, homeowner insurance, health and disability insurance, life insurance, retirement planning, and estate planning.

Requirements to a feasible to-be concept as introduced in Braun (2006a) are primarily requirements to balance sheets such as the composition of the entire portfolio, a cash reserve, and so forth. According to the financial goals of a company, a client may have the following two main requirements to a feasible to-be concept: assurance of liquidity (i.e., to always have enough money to cover his consumer spending) and prevention of personal insolvency (i.e., to avoid the inability to pay).

Assurance of liquidity (liquidity management) means that one has to prepare for anticipated cash

shortages in any future month by ensuring that enough liquid assets to cover the deficiency are available. Some of the more liquid assets include a checking account, a savings account, a money market deposit account, and money market funds. The more funds are maintained in these types of assets, the more liquidity a person will have to cover cash shortages. Even if one has not sufficient liquid assets, one can cover a cash deficiency by obtaining short-term financing (such as using a credit card). If adequate liquidity is maintained, one will not need to borrow every time one needs money. In this way, a person can avoid major financial problems and therefore be more likely to achieve financial goals.

Prevention of personal insolvency can be carried out by purchasing insurance. Property and casualty insurance insure assets (such as car and home), health insurance covers health expenses, and disability insurance provides financial support in case of disability. Life insurance provides the family members or other named beneficiaries with financial support if they lose their breadwinner. Thus, insurance protects against events that could reduce income or wealth. Retirement planning ensures that one will have sufficient funds for the old-age period. Key retirement planning decisions involve choosing a retirement plan, determining how much to contribute, and allocating the contributions.

If the *feasibility check* fails, personal financial planners can apply their knowledge and experience to balance the financial situation. They can take actions which adjust to the financial structure by changing requirements to a feasible to-be concept, by changing future expenses (such as expenses for a car), or by alternative financing expenses (for example, by enhancement of revenues). These actions may be necessary to ensure that the main financial goals are fulfilled.

Measures to Take

Financial goals of a client determine the planning steps as gathered in the section "Determining a Feasible To-Be Concept" and activities to achieve a feasible to-be concept.

UML Diagrams

In this section, we will elaborate the use case, *Determining the client's financial status: Data evaluation*, as an example of the UML diagram.

The *class diagram* in Figure 7 shows a first-pass analysis model for the use case, *Determining the client's financial status: Data evaluation*. The most important elements are the class names, the types, and the relationships.

This basic class diagram can be elaborated with the operations and attributes as described in

Figure 7. Basic use case analysis model elements

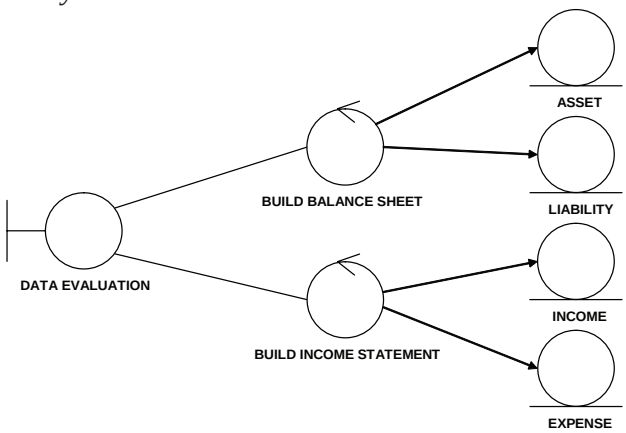


Figure 8. Elaborated analysis model elements

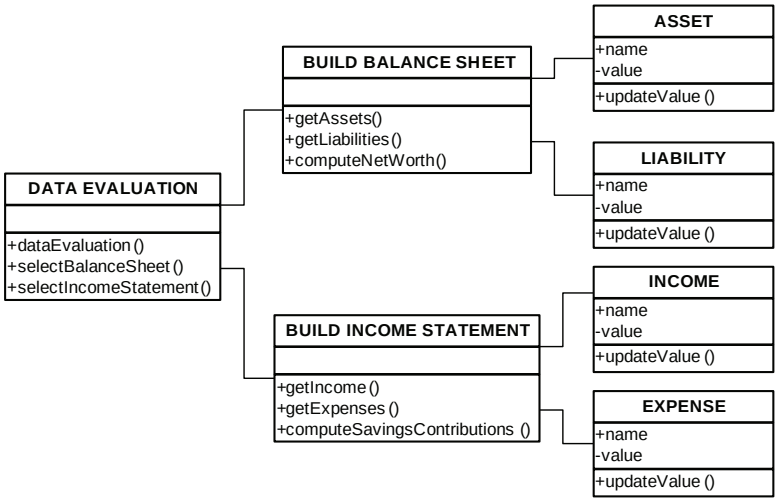


Figure 9. Basic sequence diagram

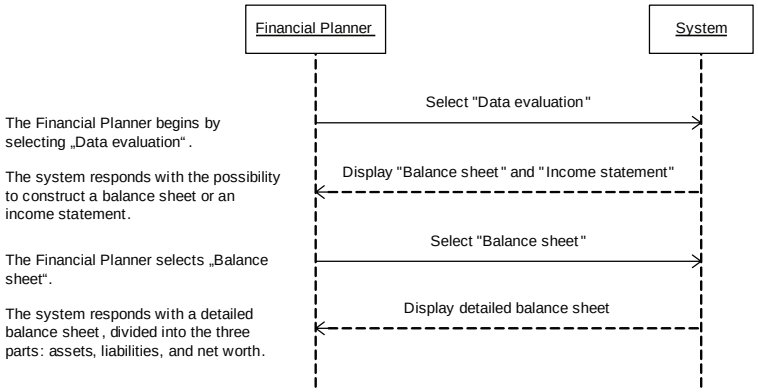


Figure 10. Sequence diagram elaborated with analysis objects

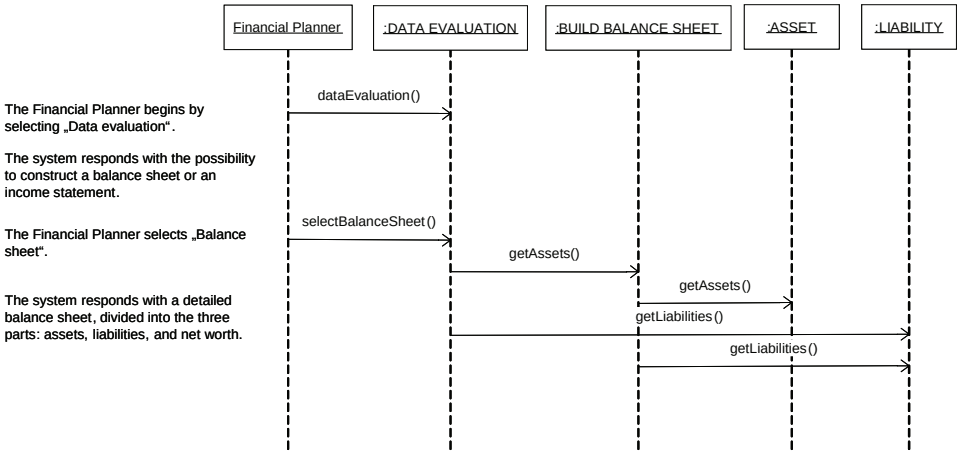
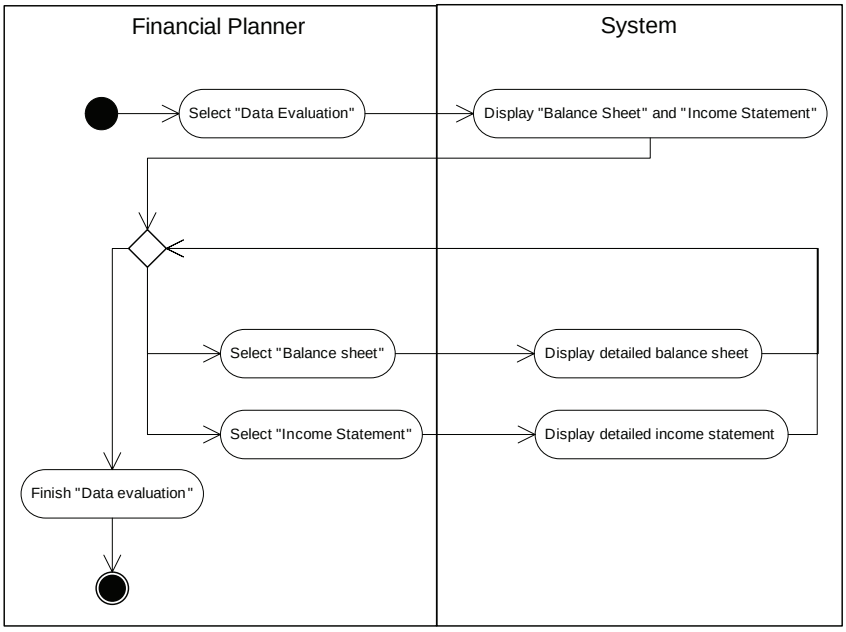


Figure 11. Basic activity diagram



the use case specification to an elaborated class diagram (Figure 8).

Figure 9 shows a simple *sequence diagram* that indicates a basic flow of the use case, *Determining the client's financial status: Data evaluation.* Messages corresponding to operations on the classes are drawn to correspond to the main narrative text alongside the scenario.

Figure 10 shows the same use case scenario, *Determining the client's financial status: Data evaluation,* as depicted in the sequence diagram in Figure 9, with *System* expressed in terms of the analysis classes in Figure 7.

Figure 11 shows the basic *activity diagram* for the use case, *Determining the client's financial status: Data evaluation,* elaborated with classes from Figure 7.

CONCLUSION AND FUTURE TRENDS

We gave an architecture for a reliable *personal financial planning* system consisting of an *system architecture* and a corresponding *analysis model* (conceptual model) as part of the application architecture. An intrinsic part of our *system architecture* is the usage of Web technologies. Our model for personal financial planning fulfills two kinds of purposes: First, the *analysis model* is a conceptual model that can help financial planners to do their job. Second, at the design stage of system development, system developers can take the *analysis model* and apply the *system architecture* to them. The model combines business concepts of *personal financial planning* with technical concepts from information technology. We evaluated the model by building an IT system FiXplan (IT-based personal financial planning) based on the model. We showed that our model fulfills *maintainability, adaptability, efficiency, reliability,* and *portability* requirements. Furthermore, it is based on ISO and CFP Board *standards.*

Further research is needed to enlarge the model with respect to an automatisation of the to-be concept feasibility check. Here, classical *mathematical programming models* as well as *fuzzy*

models may deliver a solution to that problem. While in the case of classical models the vague data is replaced by average data, fuzzy models offer the opportunity to model subjective imaginations of the client or the personal financial planner as precisely as they will be able to describe it. Thus the risk of applying a wrong model of the reality, and selecting wrong solutions which do not reflect the real problem, could be reduced.

REFERENCES

Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice. The SEI series in software engineering.* Reading, MA: Addison Wesley.

Becker, J., & Schütte, R. (2004). *Handelsinformationssysteme*. Frankfurt am Main: Redline Wirtschaft.

Bernus, P., Mertins, K., & Schmidt, G. (2005). *Handbook on architectures of information systems.* Berlin: Springer.

Bhargava, H. K., Power, D. J., & Sun, D. (in press). Progress in Web-based decision support technologies. *Decision Support Systems.* (Corrected Proof), 2005.

Böckhoff, M., & Stracke, G. (2003). *Der Finanzplaner.* Heidelberg: Sauer. 2 Auflage.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. et al. (Eds.). *Web services architecture, W3C.* Retrieved August 5, 2005, from http://www.w3.org/TR/ws-arch/

Braun, O. (2006a). Lebensereignis- und Präferenzorientierte Persönliche Finanzplanung—Ein Referenzmodell für das Personal Financial Planning. In G. Schmidt (Ed.), *Neue Entwicklungen im Financial Planning,* Hochschulschriften des Instituts für wirtschaftsinformatik. Hochschule Liechtenstein.

Braun, O. (2006b). *Web-basierte Informationssysteme in betriebswirtschaftlichen Anwendungsumgebungen am Beispiel IT-gestützter Persönlicher Finanzplanung.* Habilitationsschrift. Saarland University, Saarbrücken, in preparation.

Braun, O., & Kramer, S. (2004). Vergleichende Untersuchung von Tools zur Privaten Finanzplanung. In S. Geberl, S. Weinmann, & D. F. Wiesner (Eds.), *Impulse aus der Wirtschaftsinformatik* (pp. 119-133). Heidelberg: Physica.

Braun, O., & Schmidt, G. (2005). A service oriented architecture for personal financial planning. In M. Khosrow-Pour (Ed.), In *Proceedings of the 2005 IRMA International Conference, Managing Modern Organizations with Information Technology* (pp. 1032-1034). Hershey, PA: Idea Group Publishing.

Briggs, R. O., Nunamaker, J. F., & Sprague, R. H. (1997). 1001 Unanswered research questions in GSS. *Journal of Management Information Systems, 14,* 3-21.

Certified Financial Planner's (CFP) Board of Standards, Inc. (2005). Retrieved August 14, 2005, from http://www.cfp.net

Chou, S. C. T. (1998). Migrating to the Web: A Web financial information system server. *Decision Support Systems, 23,* 29-40.

Conallen, J. (2002). *Building Web applications with UML (2nd ed.).* Boston: Pearson Education.

DeVaney, S., Greninger, S. A., & Achacoso, J. A. (1994). The Usefulness of financial ratios as predictors of household insolvency: Two perspectives. *Financial Counseling and Planning, 5,* 5-24.

Dong, J. C., Deng, S. Y., & Wang, Y. (2002). Portfolio selection based on the Internet. *Systems Engineering: Theory and Practice, 12,* 73-80.

Dong, J. C., Du, H. S., Wang, S., Chen, K., & Deng, X. (2004). A framework of Web-based decision support systems for portfolio selection

with OLAP and PVM. *Decision Support Systems, 37,* 367-376.

Fan, M., Stallaert, J., & Whinston, A. B. (1999). Implementing a financial market using Java and Web-based distributed computing. *IEEE Computer, 32,* 64-70.

Fan, M., Stallaert, J., & Whinston, A. B. (2000). The Internet and the future of financial markets. *Communications of the ACM, 43,* 82-88.

Federal Reserve Board Division of Consumer and Community Affairs (2002). *Financial literacy: An overview of practice, research, and policy.* Retrieved on October 24, 2006 from http://www.federalreserve.gov/pubs/bulletin/2002/1102lead.pdf

Fettke, P., & Loos, P. (2003). Multiperspective evaluation of reference models—towards a framework. In M. A. Jeusfeld & Ó. Pastor (Eds.), *ER 2003 Workshops* (LNCS 2814, pp. 80-91). Heidelberg: Springer.

Fowler, M. (1997). *Analysis patterns: Reusable object models.* Menlo Park, CA: Addison-Wesley.

Frank, U. (1999). Conceptual modelling as the core of the information systems discipline—Perspectives and epistemological challenges. In W. D. Haseman & D. L. Nazareth (Eds.), *Proceedings of the 5th Americas Conference on Information Systems (AMCIS 1999)* (pp. 695-697). Milwaukee, WI.

Greninger, S. A., Hampton, V. L., & Kitt, K. A. (1994). Ratios and benchmarks for measuring the financial well-being of families and individuals. *Financial Counselling and Planning, 5,* 5-24.

Hay, D. C. (1996). *Data model patterns—conventions of thought.* New York: Dorset House Publishing.

Hirschey, M., Richardson, V. J., & Scholz, S. (2000). Stock price effects of Internet buy-sell recommendations: The motley fool case. *The Financial Review, 35,* 147-174.

International Organization for Standardization (ISO/TC 222). (2004). *ISO/DIS 22 222-1 of the Technical Committee ISO/TC 222, Personal financial planning.*

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-oriented software engineering: A use case driven approach.* Boston: Addison-Wesley.

Keown, A. J. (2003). *Personal finance: Turning money into wealth*. Upper Saddle River, NJ: Prentice Hall.

Lang, K., Taumann, W., & Bodendorf, F. (1996). Business process reengineering with reusable reference process building blocks. In B. Scholz-Reiter & E. Stickel (Eds.), *Business process modelling* (pp. 264-290). Berlin: Springer

Li, Z. M. (1998). Internet/Intranet technology and its development. *Transaction of Computer and Communication, 8,* 73-78.

Marent, C. (1995). Branchenspezifische Referenzmodelle für betriebswirtschaftliche IV-Anwendungsbereiche. *Wirtschaftsinformatik, 37*(3), 303-313.

Marshall, C. (1999). *Enterprise modeling with UML: Designing successful software through business analysis.* Reading, MA: Addison-Wesley.

Mathieu, R. G., & Gibson, J. E. (1993). A methodology for large scale R&D planning based on cluster analysis. *IEEE Transactions on Engineering Management, 30,* 283-291.

Mehdi, R. Z., & Mohammad, R. S. (1998). A Web-based information system for stock selection and evaluation. In *Proceedings of the International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems,* p. 81,Washington D.C.: IEEE Computer Society.

Mertens, P., Holzer, J., & Ludwig, P. (1996). *Individual- and Standardsoftware: tertium datur?*

Betriebswirtschaftliche Anwendungsarchitekturen mit branchen- und betriebstypischen Zuschnitt (FORWISS-Rep. FR-1996-004). Erlangen, München, Passau.

Mišić, V. B., & Zhao, J. L. (1999, May). Reference models for electronic commerce. In *Proceedings of the 9th Hong Kong Computer Society Database Conference —Database and Electronic Commerce*, Hong Kong (pp. 199-209).

Mukherjee, K. (1994). Application of an interactive method for MOLIP in project selection decision: A case from Indian coal mining industry. *International Journal of Production Economics, 36,* 203-211.

Nissenbaum, M., Raasch, B. J., & Ratner, C. (2004). *Ernst & Young's personal financial planning guide (5th ed.).* John Wiley & Sons.

Remme, M. (1997). *Konstruktion von Geschäftsprozessen—Ein modellgestützter Ansatz durch Montage generischer Prozesspartikel.* Wiesbaden: Gabler.

Rumbaugh, J., Jacobson, I., & Booch, G. (2005). *The unified modelling language reference manual* (2nd ed.). Boston: Addison-Wesley.

Samaras, G. D., Matsatsinis, N. F., & Zopounidis, C. (2005). Towards an intelligent decision support system for portfolio management. *Foundations of Computing and Decision Sciences, 2,* 141-162.

Scheer, A. W. (1994). *Business process engineering—reference models for industrial companies* (2nd ed.). Berlin: Springer.

Schmidt, G. (1999). *Informationsmanagement.* Heidelberg: Springer.

Schmidt, G. (2006). *Persönliche Finanzplanung - Modelle und Methoden des Financial Planning.* Heidelberg: Springer.

Schmidt, G., & Lahl, B. (1991). Integration von Expertensystem- und konventioneller Software am Beispiel der Aktienportfoliozusammenstellung. *Wirtschaftsinformatik, 33,* 123-130.

Scholz-Reiter, B. (1990). *CIM—Informations— und Kommunikationssysteme.* München, Wien: Oldenbourg.

Schütte, R. (1998). *Grundsätze ordnungsmäßiger Referenzmodellierung—Konstruktion konfigurations- und anpassungsorientierter Modelle.* Reihe neue betriebswirtschaftliche Forschung. Wiesbaden.

Turban, E. (1995). *Decision support and expert systems* (4th ed.). Englewood Cliffs, NJ: Prentice-Hall.

Watters, P. A. (2005). *Web services in finance.* Berlin: Springer.

Woerheide, W. (2002). *Core concepts of personal finance.* New York: John Wiley and Sons.

Wüstner, E., Buxmann, P., & Braun, O. (2005). The extensible markup language and its use in the field of EDI. In P. Bernus, K. Mertins, & G. Schmidt (Eds.), *Handbook on architectures of information systems* (2nd ed., pp. 391-419). Heidelberg: Springer.

Youngs, R., Redmond-Pyle, D., Spaas, P., & Kahan, E. (1999). A standard for architecture description. *IBM Systems Journal 38*. Retrieved on October 16, 2006 from http://www.research.ibm.com/journal/sj/381/youngs.html

Zahedi, F., & Palma-dos-Reis, A. (1999). Designing personalized intelligent financial decision support systems. *Decision Support Systems, 26,* 31-47.

Zopounidis, C., Doumpos, M., & Matsatsinis, N. F. (1997). On the use of knowledge-based decision support systems in financial management: A survey. *Decision Support Systems, 20,* 259-277.

This work was previously published in Architecture of Reliable Web Applications Software, edited by M. Radaideh; H. Al-Ameed, pp. 83-111 , copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.11
Educational Theory Into Practice Software (ETIPS)

Sara Dexter
University of Virginia, USA

ABSTRACT

The ETIPS software is a Web-based learning environment that delivers cases that allow educators to practice instructional decision making. Here I recount its development but mainly emphasize the two key concepts that were central to our design process. The first was the Conceptual Assessment Framework, an evidentiary reasoning and design perspective that helped us to focus on which key attributes to build into the software and cases. The second concept is described as extreme programming, which is an iterative approach to software programming based upon user stories and rapid prototyping. The story of developing the ETIPS cases illustrates the need to know very clearly what the point is of the educational experience you are creating and to design software where form follows function.

INTRODUCTION

In this chapter, I recount the important aspects of the creation of the ETIPS software and its cases but mainly emphasize the two key concepts that were central to our design process. The first was the *Conceptual Assessment Framework*, developed by Mislevy, Steinberg, Almond, Haertel, and Penuel (2001); this framework helped us to focus on which key attributes to build into the software and cases. The second concept is described as extreme programming (Beck & Fowler, 2000), which is an iterative approach to software programming based upon user stories and rapid prototyping.

The story of developing the ETIPS cases illustrates the need to know very clearly what the point is of the educational experience you are creating and to design software where form follows function. The first generation of ETIPS

cases was created with existing case-authoring software; halfway through this four-year project our team realized that this software constrained the sort of learning experience we wanted the cases to provide. During the project's third year we began to create from scratch software for a second generation of cases and an interface that brought to fruition our case-based pedagogical approach. We used the *Conceptual Assessment Framework* (Mislevy et al., 2001) to guide the development and refinement of each user story for the software; this helped us to connect form to function in the second generation of software and to recognize how our case-based pedagogy could be used with other topic areas as well. Thus, a side benefit of using these conceptual approaches was that we increased our product's sustainability through broader user bases, potential co-authoring partnerships, and licensing.

EDUCATIONAL PURPOSE OF THE CASES

The purpose of the ETIPS project was to create teacher education cases that were learning exercises about educational technology integration and implementation. The primary audience for our cases was pre-service teacher education classes on either educational technology or pedagogical methods. Key premises upon which we based the software for our second generation of cases were that teaching is decision making—and decision making is a process that can be taught and requires practice in order to learn—and that instructional decisions are guided by schemas, or mental models.

The cases allowed students studying to be teachers to practice making instructional decisions about educational technology use in classrooms and schools using the Educational Technology Integration and Implementation Principles as a schema, or the basis of a schema, for those decisions. By providing instructors nine virtual yet realistic schools among which to choose to set these decision-making exercises in it allowed them to give their students multiple practice opportunities to see how these principles can guide instructional decision making about technology integration and implementation in a variety of school contexts.

The six principles summarize what research suggests are the conditions that should be present in order for educational technology integration and implementation to be effective (Dexter, 2002). The first three educational technology principles focus on integration, meaning teachers' instructional decision-making process when considering the use of educational technology resources in their classrooms. Discussion of these principles develops the premise that a teacher must act as an instructional designer and plan for the use of the technology to support student learning.

- **Principle 1:** Learning outcomes drive the selection of technology.
- **Principle 2:** Technology use provides added value to teaching and learning.
- **Principle 3:** Technology assists in the assessment of the learning outcomes.

The last three educational technology principles focus on the implementation of technology at the school level—that is, how a school setting can create a supportive context that provides teachers with the necessary access to technology, technical and instructional support, and a positive climate for professional collaboration about educational technology tools.

- **Principle 4:** Ready access to supported, managed hardware/software resources is provided.
- **Principle 5:** Professional development is targeted at successful technology integration.

- **Principle 6:** Professional community enhances technology integration and implementation.

UNIQUE FEATURES OF THE ETIPS CASE METHOD

ETIPS stands for Educational Theory into Practice Software. It is a Web-based learning environment in which students complete cases that are set in a K-12 school and focus on an educational theory. Unlike text-based cases, which are read in a linear fashion and emphasize the multiplicity of perspectives inherent in an event that is often told in chronological fashion, cases in ETIPS present learners with a scenario in which they need to make an instructional decision, and require them to select which information they think they will need to make that decision. The case is an opportunity to practice reasoning with a guiding theory that relates to the case topic and to develop an understanding of how the different school contexts in which the cases are set might influence how that theory is applied in practice.

This case approach emphasizes learners' metacognition—their thinking about their thinking—through a software feature called a PlanMap. The PlanMap asks students to check off what information they think they will need to make the decision posed in the scenario (see Figure 1); if they return to their PlanMap during the case, they will see that these choices are noted with a checkmark (see Figure 2). As students look at information in the case—and their choices are not limited to only what they checked while planning their search—the software records what they access and uses a different icon to record it on their PlanMap; in addition, experts' recommendations of which key items should be considered are indicated with yellow highlighting. Thus, the PlanMap provides feedback to the learners on their planned and actual progress as well as an in-progress check of their approach as compared to experts'.

Figure 1. The PlanMap page view initially explains to students the purpose of the PlanMap, and asks them to click in the box next to each category of case information they think will be necessary to access in order to complete the case

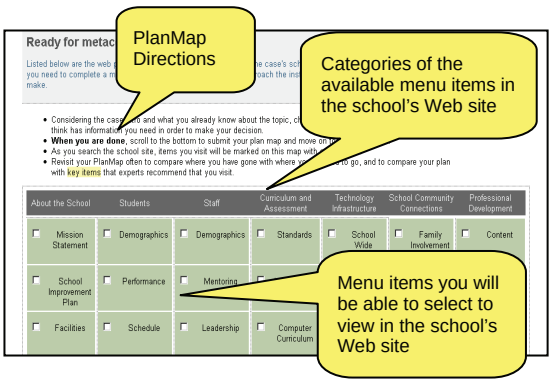


Figure 2. This close up of a PlanMap from a case in progress shows how checkmarks indicate items the user planned to visit; green arrows indicate what case information the user has viewed so far, and yellow highlighting shows which pieces of information experts deem as key for making a decision such as the case requires

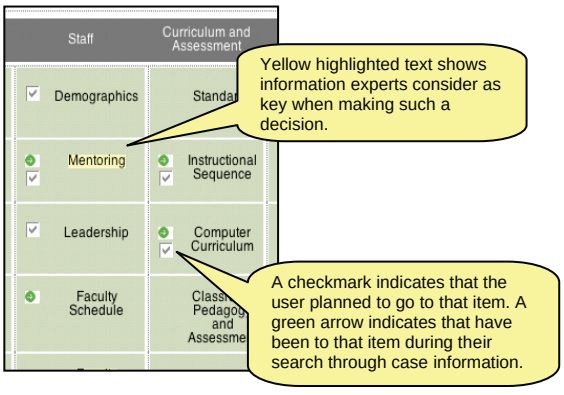
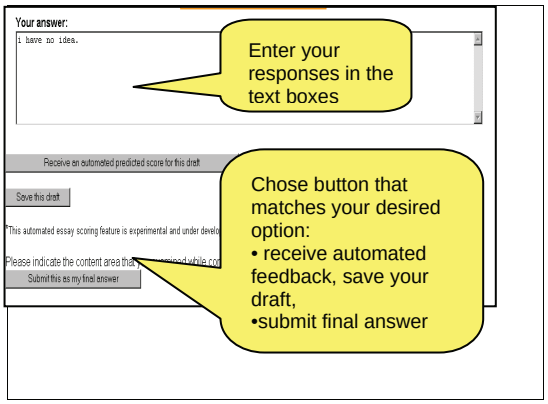


Figure 3. On the submit answer page, students can chose to save their work as a draft, use the automated essay scoring feature, or submit their response to their instructor as a final answer

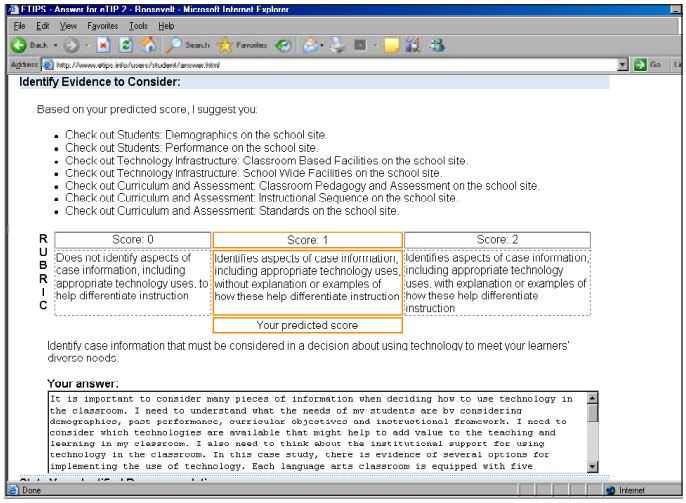


Another formative assessment tool in ETIPS is automated essay scoring, which students can use to get feedback on their decision; this feedback is in the form of a predicted score of their short answer responses against a rubric before they submit it to their instructor for a final grade (see Figure 3). The automated essay scoring engine software compares the student's response to other essays, which were scored by humans against the same rubric, predicts which score a student is most likely to receive, and presents it and the scoring criteria to them. In addition, if students have not yet been to the case information items that experts consider to be key, it suggests they review that information.

ETIPS PROJECT TEAM MEMBERS

The ETIPS project was funded by a grant from the US Department of Education through its Preparing Tomorrow's Teachers to Use Technology (PT3) initiative. A majority of the grants were made to colleges of education; consortiums were encouraged, and awardees were required to include a project evaluator to collect data for performance feedback. The initial ETIPS team represented a strategic alliance among the project lead, which was a college of education at a major research university, a software development lab based at a similar type of institution, and a non-profit organization with extensive experience developing software and delivering professional

Figure 4. The automated essay scoring engine shows the student their predicted score for that answer, the rubric criteria, and suggests any of the case's information items they should also take into consideration



development. Toward the end of the second year of the project, when the project director decided to create from scratch the software that was the basis for the second generation of cases, the software development lab agreed to bow out of the partnership.

At the lead university the team included the half-time project director, who had educational technology and teacher education expertise, a full-time project coordinator and two half-time graduate assistants who wrote case content and supported our test-bed of faculty members and their students, and a half-time project evaluator. At the non-profit organization there were two part-time collaborators who brought extensive assessment experience to the project as well as a software programming company with whom they had successfully worked and that we hired on an hourly basis.

PROJECT CHALLENGES

The major challenge that the project faced was articulating clearly the pedagogical purpose of the case-based learning experience we wanted to provide to learners. Because at the outset of the project we began to create cases using the software developed previously by the university-based software development lab in our consortium we initially assumed the learning experience functions inherent in that software. While aspects of the resulting learning experience worked well for this audience and these topics, other aspects were not well suited to practicing instructional decision making.

The first generation of cases was based upon the case approach pioneered by the IMMEX Software Development Lab at UCLA (Stevens, 1991). The IMMEX approach posed a problem to the learner and presented him or her with a menu-driven approach to selecting information necessary in order to solve it. The problem had a right answer, which—after having looked through sufficient case information—the user would select from a multiple-choice format. IMMEX software allowed case designers to elect to limit how much information the users could select through a points system. This rewarded learners who understood the correct problem solving approach and could recognize what information to seek out and how to interpret it because they could use the least number of points to determine the correct answer choice. The software tracked their information choices and graphically portrayed the sequence of their choices and the time spent at each using IMMEX's proprietary SearchPath Map. The SearchPath Map was designed to show learners' strategic responses to solving the problem. Each labeled icon on a SearchPath Map represents one menu item; the lines connecting the icons illustrate the path a student took in trying to solve the challenge presented in the case. Lines lead from the upper left-hand corner of an icon to the middle of the icon of the next menu item that was accessed. In this way, the SearchPath Map also illustrates the order in which students visited menu items.

The map on the left represents that the student explored many menu items and thus made a rather complete search of the problem space. The performance of the student at right shows only two general areas of the problem space were explored, indicating they do not have a firm grasp of the concepts underlying the problem that was presented

The IMMEX developers applied neural network analysis to determine the most common problem-solving approaches used by students who selected correct and incorrect answers, and to infer the likely student misconceptions that would lead to their taking such paths (Stevens, Johnson, & Soller, 2005; Stevens, Wand, & Lopo, 1996).

At the outset of the ETIPS project it appeared that the major change to IMMEX that would be required to create cases about technology integration and implementation for pre-service teachers would be to allow for short answer responses.

Adding a text box for responses was easy, but soon our test-bed faculty members wanted online modules for scoring essays, recording the scores, and then reporting them to students. We also saw that the instructional decision making exercise at the core of the ETIPS cases did not have a right or wrong answer and that scoring was a key point of the analysis of IMMEX cases. Further, students' searches through case information did not result in predictable, categorizable sequences that the SearchPath maps could help illustrate. Consequently, our test-bed instructors reported that they often did not use the maps with students, which reduced the case experience's potential for helping students become aware of their schema for their instructional decision making about technology integration.

User performances and faculty feedback made clear to the project leadership team which aspects of the emerging vision of the ETIPS case-based pedagogy were not compatible with the IMMEX software's features, and so we redirected a significant portion of the project's budget away from this partner and to the programming company associated with the non-profit organization. This mid-course redesign was thus a challenge to the project's budget, timeline, and consortium partner relationships.

In retrospect, the initial design of the ETIPS cases in the IMMEX software served as a fast way to create a prototype and proof of concept for an online case-based pedagogy that was focused on student reasoning and decision making. The downside to this rapid start-up was that by using an existing software to author the cases it let us skip over what we now would argue is an essential step of creating educational games or simulations: articulating what you want to learn about the student's knowledge and skills, and determining what tasks in the game or simulation will elicit that information.

The data we collected through the evaluation component of the project reinforced the project director's impressions that there were discontinuities between our users' intended and actual experiences with the cases. The evaluation process enabled us to attend in a rigorous way to our test-bed faculty members' experiences with the cases as an instructional resource and whether or not their students learned what we wanted them to learn about the case's use.

At the time project leaders decided to create our own ETIPS software some project leadership team members had been reading about assessment design, namely the Conceptual Assessment Framework developed by Robert Mislevy and his colleagues. Applying this framework forced us to articulate what we wanted the cases to help the user learn, and what task we would create in order to elicit that knowledge and skills, and then how to interpret and report their performance data. Through this process we determined how our software could be thought of as a general case-based pedagogy where the virtual schools could serve as the settings for cases on a variety of topics. We then began to think of ETIPS as standing for Educational Theory into Practice Software and that the Educational Technology Integration and Implementation Principles, for which the acronym was originally coined, would be just one topic about which we would offer cases. We enlisted two other organizations, chosen because of their large member bases and project leaders' connections to them, as co-authors of cases about urban teaching and digital equity.

At the outset of designing the software that would anchor the second generation of cases, the software programming company the project leaders hired (Green River; see http://greenriver.org), asked us to read *Extreme Programming* (Beck & Fowler, 2000) and to take that approach to our development work together. From the project leadership team's perspective as a client, this is a user-centered design process that is articulated through descriptions of desired functionalities called user stories that are then checked out

through usability testing and revised accordingly. Thus it was congruent with our on-going evaluation process.

Table 1 recounts the timeline of the key steps during the development of the ETIPS software that anchored the second generation of the cases.

The overall budget allocated for the software development, as well as the domain name and hosting costs, over this two-year process was approximately $450,000. The personnel costs for the project director and coordinator and graduate students at the lead institution and the team members at the non-profit organization were approximately $640,000. Additional costs were incurred for travel to project meetings and for dissemination and co-authoring work that was undertaken. All the personnel costs given are for the entire life of the grant; during the first two years this went mostly for case topic conceptualization and case authoring work and in the latter two years it was for test-bed member implementation support, co-authoring, and dissemination efforts. The portion

Table 1. Timeline for key development steps of ETIPS

Timeline	Key steps
2003	Project Year Two
April	Software development planning meeting with Green River
May	Consultations with Mislevy on Conceptual Assessment Framework Test-bed faculty meeting at end of Year 1 of generation 1 cases' use
June	Programming commences
July	Review of key elements in ETIPS engine with project leadership team
August	Alpha testing with students and faculty, usability expert's review
September	Programming meeting and beta testing with one test-bed faculty member's students
October	Planning for more responsive feedback to students on their case performance User interface planning for incorporating cases on multiple topics Co-authoring of cases on new topics begins with selected organizations Graphic re-design of interfaces for public part of site, post log-in, and cases
November	Programmer's meeting to develop PlanMap feature Systematic data collection on students' reaction to automated essay scoring
December	Begin discussion with publisher about bundling cases with book
2004	Project Year Three
January	New Web site interfaces deployed PlanMap deployed
March	Usability testing of new interfaces with faculty and students
May	Section 508 compliance review
July	Revise navigation scheme per users' and 508 report input
September	Implement revisions Design better feedback on answer to students per results from student data
October	Begin controlled experiments with students to determine impact of a student's use of the automated essay scorer on his or her essay's quality
2005	Project Year Four
February	Begin controlled experiments with students to determine impact of a student's use of the PlanMap on his or her essay's quality
June	Improve performance of automated essay scorer

of the budget allocated for the evaluation work was approximately $210,000. In addition, the direct cost for the training and travel to project meetings for the test-bed faculty members' was approximately $80,000.

LESSONS TO PASS ALONG TO OTHERS

Games and simulations produced for educational purposes in effect serve as a sort of formative, and perhaps summative, assessment. That is, they allow the user to practice doing something that is based upon some key premises that guide the operational rules of the game or simulation. It is likely that through his or her performance the software indicates, among other things, how well the user understands those key premises. From the outset, the ETIPS project leaders knew we wanted the cases to be an opportunity to practice instructional decision making while keeping an educational technology integration and implementation principle in mind. But when designing the specifics of the software—especially what feedback we could give to the learner during the case—Mislevy et al.'s (2001) Conceptual Assessment Framework helped us to work more efficiently, as well as with confidence since this framework draws upon contemporary learning and assessment theory.

The National Academy of Sciences report *How People Learn: Brain, Mind, Experience and School*, (Bransford, 1999) suggested that effective learning environments, among other things, are assessment-centered in the sense of providing multiple opportunities to make students' thinking visible so they can receive feedback and be given chances to revise and to learn about their own learning. The companion National Academy of Sciences report, *Knowing What Students Know: The Science and Design of Educational Assessment* (Pellegrino, Chudowsky, & Glaser, 2001), addresses principles of assessment design for learning situations aligned with the findings reported in *How People Learn*. According to Pellegrino et al. (2001), central to assessment is reasoning from evidence generated through a process consisting of three points: "a model of how students represent knowledge and develop competence in the subject domain, tasks or situations that allow one to observe students' performance, and an interpretation method for drawing inferences from the performance evidence thus obtained" (p. 2). Mislevy et al. (2001) incorporate these same three points into their Conceptual Assessment Framework by specifying it consists of a student model, task model, and an evidence model.

Almond, Steinberg, and Mislevy (2002) write that the student model is "the knowledge, skills, and abilities…to measure for each participant" (p. 35); the task model includes "the presentation material to be presented to the user…[and] a description of the work products that will be returned as a result of user interaction with the task" (p.24); and the evidence model acts as a bridge between the student and task models in that it describes how to analyze the evidence called for by the task model and so as to assess the student's understanding.

In writing about assessment designers, Mislevy et al. (2001) and Pellegrino et al. (2001) assert that networks, new media, and new methodologies have much to offer us in the way of support and enhancement of assessment but that technology has the potential to lure designers into creating complex tasks where substantial amounts of data are collected without any plan for analyzing how it can combine as an assessment of learner progress. To guard against this, Mislevy and colleagues (2001; Almond et al., 2002) promote using evidentiary reasoning and a design perspective during the development of instruction and assessment materials so that the focus remains on construct definition, forms of evidence in keeping with the construct, and the creation of tasks that would produce such evidence.

Extreme programming is an approach to programming developed by Beck and Fowler (2000) that describes how a team of software programmers works together with the client to efficiently create reliable code. Its creators describe it more as a set of values and disciplined approach than a strict set of steps. From our programmer's use of it and the project director's and other team leaders' experience designing ETIPS, this strategic approach was excellent for letting the design of a case-based pedagogy emerge. This included the focus on developing an overall metaphor to illuminate what you are creating and then expressing distinct aspects of functionality in user stories that are coded, reviewed by the client and users, and then revised as needed.

Extreme programming demands a lot of communication between the client and the programmers, often on a quick-turnaround basis. Further, to the degree the design process is driven by user testing, this requires advance planning for data collection, analysis, and reporting so that the code revisions can occur on a timeline that meets deadlines. Our team used a wiki to record our user stories and record notes from project meetings. The project director and programmers used a trouble ticket system to record the priority of the various user stories, recorded one per ticket, and their timelines and related communication. Working with the project evaluator, the project director and coordinator arranged for data collection from faculty and student users as needed. We also consulted experts on the usability of our site, including our compliance with disabled users' needs as specified Section 508 of the Rehabilitation Act.

Considering the Conceptual Assessment Framework during the design phase and following the approach advocated in *Extreme Programming* to bring that design into code can help creators of games and simulations work efficiently toward the educational purpose of their work. More importantly, it will encourage the development of assessment-centered learning environments, which we know will aid learning.

WHAT IS NEXT FOR OUR TEAM

At the time of this writing the project has about six months of funding left and so in addition to refining all features and user interfaces and support materials, we are focused on creating a revenue stream that will ensure the sustainability of our work. Over the last year we explored various models for generating income from the use of the cases. We have decided upon three related strategies.

The first is to consider the functionality of the software and the case-based pedagogy it supports as a separate product that can be marketed to developers of educational cases on various topic areas. Developers would pay to license the ETIPS engine, and any income could help to improve the software. This strategy involved setting up working arrangements with a company that is focused on marketing the ETIPS engine as well as other educational software.

A second strategy is that we will work with the membership networks and organizations that inspired the cases that we authored on additional topics. They, in turn, may promote the use of these cases to their members, who may then purchase access to the cases.

The third, and most promising, strategy pertains to just the ETIPS cases on technology integration and implementation. The project director and a long-time test-bed faculty member are writing a series of books on technology integration in secondary science, mathematics, social studies, and English language arts. The educational technology integration and implementation principles serve as an organizational framework for the books; in the chapters about each of the six principles the online ETIPS cases are presented as homework exercises that learners complete to practice applying the

main idea from that chapter. With this strategy the student who purchase the book is then very likely to purchase access to the cases, through an e-commerce functionality we have added. This model of students assuming responsibility for the costs of the educational materials associated with a class is familiar to students and faculty alike, and also allows our development team to leverage the publishing company's expertise in marketing, selling, and distributing the books, and their inherent relationship to some of the ETIPS cases.

REFERENCES

Almond, R. G., Steinberg, L. S., & Mislevy, R. J. (2002). Enhancing the design and delivery of assessment systems: A four-process architecture. *Journal of Technology, Learning, and Assessment*, 5. Retrieved from http://www.jtla.org

Beck, K., & Fowler, M. (2000). *Extreme programming explained.* Boston: Addison-Wesley.

Bransford, J. D., Brown, A. L., & Cocking, R. R. (1999). *How people learn: Brain, mind, experience, and school.* Committee on Developments in the Science of Learning with additional material from the Committee on Learning Research and Educational Practice, National Research Council. Washington, D.C.: National Academy Press. Retrieved May 24, 2002, from http://www.nap.edu/html/howpeople1/

Dexter, S. (2002). eTIPS-Educational technology integration and implementation principles. In P. Rodgers (Ed.), *Designing instruction for technology-enhanced learning* (pp.56-70). Hershey, PA: Idea Group Publishing.

Mislevy, R. J., Steinberg, L. S., Almond, R. G., Haertel, G. D., & Penuel, W. B. (2001). Leverage *points for improving educational assessment* (Tech. Rep. No. 534). Retrieved from http://cse.ucla.edu/CRESST/Summary/534.htm

Pellegrino, J., Chudowsky, N., & Glaser, R. (2001). *Knowing what students know.* Washington, DC: National Academy Press.

Stevens, R. H. (1991). Search path mapping: A versatile approach for visualizing problem-solving behavior. *Academic Medicine, 66*(9), S72-S75.

Stevens, R., Johnson, D., & Soller, A., (2005). Probabilities and predictions: Modeling the development of scientific problem solving skills. *Cell biology education*, *4*(1), 42-57.

Stevens, R., Wang, P., & Lopo, A. (1996). Artificial neural networks can distinguish novice and expert strategies during complex problem-solving. *JAMIA*, *3*(2), 131-138.

This work was previously published in Games and Simulations in Online Learning: Research and Development Frameworks, edited by D. Gibson, pp. 223-238, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 2.12
Engineering Reusable Learning Objects

Ed Morris
RMIT University, Australia

ABSTRACT

We adapt the object-oriented software engineering design methodology for software objects to engineering reusable learning objects. Our approach extends design principles for reusable learning objects. The resulting learning object class is a template from which individualised learning objects can be dynamically created for, or by, students. The properties of these classes refine learning object definitions and design guidelines. We adapt software object levels of cohesion to learning object classes. We demonstrate reusability increases when learning object lessons are built from learning objects, like maintainable software systems are built from software objects. We identify facilities for learning management systems to support object-oriented learning object lessons that are less predetermined in sequencing activities for each student. Our overall approach to the design of learning object lessons is independent of, and complementary to, instructional design theory underlying the learning object design process, and metadata standards adopted by the IEEE for learning object packaging.

PERSPECTIVE

We approach the shared aim to design reusable e-learning objects by adapting software engineering methodology, where it enhances the reusability of software objects. Our work (Morris, 2005) contributes software engineering techniques to the design and evaluation of learning objects with enhanced reusability.

We also provide a different perspective on the continuing pedagogical debate over granularity and context for optimal e-learning object reusability (Littlejohn, 2003) as we focus on the user interface and internal structure of a learning object to enhance its reusability.

OBJECTIVES

The objectives of this chapter are:

1. To explain how object-oriented software engineering design methodology can be applied to the design of a learning object to enhance its reusability. See the "Designing Learning Objects as Software Objects" subsection and the "Example Learning Object Classes" section.
2. To show how object-oriented software engineering extends and refines Boyle's (2002) design principles for authoring dynamic reusable learning objects by enabling individual learning objects to be dynamically created for or by students from a template learning object, which we call a learning object class. See the "Designing Learning Objects as Software Objects" subsection and the "Example Learning Object Classes" section.
3. To contribute toward a learning object lesson design methodology that will facilitate the design and implementation of larger scale lessons, courses, and educational programs. See the "Developing Learning Object Lessons as Software Systems" subsection and the "Object-Oriented Design Principles for Learning Objects" section.
4. To explain how the properties of a learning object class refine existing learning object definitions and design guidelines. See the "Criteria That Define Object-Oriented Learning Objects" subsection.
5. To show how reusability is further enhanced by standardising the interface of a learning object class to provide its learning activities as services that can be invoked by other learning objects. See the "Developing Learning Object Lessons as Software Systems" subsection and the "Example Learning Object Classes" section.
6. To explain how software object levels of cohesion can be applied to the design of a learning object class, such that the higher the level of cohesion, the more it is reusable. See the "Object-Oriented Design Principles for Learning Objects" section.
7. To identify facilities required in a learning management system to support learning object lessons that are less predetermined in their sequencing of activities for each student. See the "Support for Object-Oriented Learning Objects" section.
8. To explain how our object-oriented software engineering approach to the design of learning object lessons is independent of, and complementary to, (a) instructional design theory underlying the learning object design process, and (b) metadata standards adopted by the IEEE for learning object packaging. See the "Support for Object-Oriented Learning Objects" section.

INTRODUCTION

Early research and development of online learning materials did not focus on their reusability. For example, our previous research focussed on the cost effectiveness (Zuluaga, Morris, & Fernandez, 2002) and educational effectiveness (Morris & Zuluaga, 2003) of our online learning approach. This involved both online course development and online course delivery phases. We also addressed the deployment, management, and scalability of our online courses over a network of learning management system servers (Zuluaga & Morris, 2003).

Most of our early online courses (1999-2002) were developed for 100% online delivery, utilising mostly textual learning materials, plus (on average) four short multimedia supplements such as Java applets, Flash animations, voice overs, and video clips. During online delivery of such

a course we relied more on students interacting with staff via e-mail or chat than on building interactivity into the online course material during its development. We assumed these early generation online course materials would soon be replaced as multimedia and learning management systems matured and standardised. So we were not so much concerned with upgrading the original online materials over the years, repurposing them for new educational programs, or repackaging them for different media in the future. Nor were we particularly concerned with standardising the packaging of online course materials so that they could be reassembled with online materials from other institutions.

However, during this early period of research and development (R&D) it became increasingly clear that e-learning materials need to satisfy all the 'bilities': *interoperability* among different systems connected by the Internet, *accessibility* anytime from another location, *reusability* by other developers to save time and money, *discoverability* in repositories using metadata, *extensibility* of existing courses due to their modular construction, *affordability* due to reduced development costs, and *manageability* by allowing easy changes and updates to small chunks (Computer Education Management Association, 2001).

The concept of a unit of e-learning material or 'learning object' is central to these objectives. A range of definitions of *learning object* or 'instructional object' exist (Wiley, 2001). One that captures a common theme defines learning objects as "small but pedagogically complete segments of instructional content that can be assembled as needed to create larger units of instruction, such as lessons, modules and courses. Learning objects should be stand-alone, and be built upon a single learning objective, or a single concept" (Hamel & Ryan-Jones, 2002).

Boyle (2002) proposed learning object design principles synthesised from pedagogy and software engineering for authoring dynamic reusable learning objects. From pedagogy, a learning object should have a single learning objective. From software engineering, a learning object should do one thing and only one thing (strong cohesion). And a learning object should have minimal bindings to other learning objects (weak coupling).

Contents

In the second section of this chapter, we expand the above synthesis by applying object-oriented software engineering design methodology to the design of learning objects. We introduce the 'abstraction' of a learning object to enable a designer to produce a 'learning object class'. A *learning object class* is a template from which similar but individualised learning objects can be dynamically created during a lesson. (Object-oriented software engineering refers to similar 'objects' being 'instantiated' (created) from a 'class', which encapsulates their shared attributes and activities.) We introduce 'inheritance' so that a designer can evolve a 'child' learning object class from its 'parent' learning object class, extending and modifying its attributes and activities as desired. The instructional designer can use inheritance to reuse learning object classes or repurpose a lesson by extending and coupling inherently cohesive learning object classes. The instructional designer can use instantiation during a lesson to enable student interaction to determine the actual sequence of possible events in a lesson.

In the third section we illustrate the application of object-oriented software engineering design methodology to the design of two learning object classes. Broadly speaking, one is for the programming discipline, and the other is for psychology. The first is based on the Java programming language while-loop learning object of Boyle et al. (2003). The second is based on a conflict resolution learning object that explains Maddux's five styles of conflict resolution (Rathsack, 2001). These two learning object classes from different disciplines demonstrate the general applicability of our approach.

In the fourth section, we adapt to learning objects a scale for grading the cohesion of software objects. We show how to classify a learning object's level of cohesion and explain how each of the lower levels further reduces learning object reusability. This informs the design of learning objects, as we illustrate with the examples from the third section.

Finally, in the fifth section, we identify facilities required in a learning management system to support object-oriented learning object lessons. We point out that our object-oriented software engineering approach to the design of learning object lessons is independent of, and complementary to, the instructional design theory underlying the learning object design process, and the metadata standards adopted by the IEEE (LTSC, 2003) for learning object packaging.

APPLICATION OF OBJECT-ORIENTED SOFTWARE ENGINEERING TO LEARNING OBJECTS

Software engineering is concerned with the design and implementation of large scale, complex information processing systems that are robust, maintainable, modularly reusable, scalable, and extensible (Pfleeger, 2001). These properties overlap the 'bilities' required of learning objects. This observation underlies the application of software engineering design principles to the design of learning objects. Boyle introduced this approach with reference to coupling and cohesion principles for the design of learning objects (Boyle, 2002). We extend this approach by applying *object-oriented* software engineering design methodology to the design of learning objects. Object-oriented software engineering has evolved into a dominant 'branch' in the software engineering design methodology 'tree'.

In the "Designing Learning Objects as Software Objects" subsection we show how learning objects can be designed with essentially the same techniques used to design software objects. In the "Developing Learning Object Lessons as Software Systems" subsection we explain how flexible learning object lessons can be built from reusable learning objects in the same way that maintainable software systems are built from software objects with well-designed interfaces. Our application of object-oriented software engineering to the design of object-oriented learning object lessons leads us in the "Criteria That Define Object-Oriented Learning Objects" subsection to synthesise criteria that define a truly object-oriented learning object.

Designing Learning Objects As Software Objects

Object orientation is an approach to software development that organises both the problem and the solution as a collection of discrete 'objects' (Pfleeger, 2001). Each software object can be based on a physical or abstract object in the problem space. In the software system solution, the software objects collaborate to answer a user's requests.

- **Problem:** Model an employee-employer relationship. This could be a fundamental requirement of a software application to manage staff. Solution: We consider an employee software object below; the reader can similarly consider an employer software object later.

In the object-oriented approach to software design the nouns in the problem statement generally identify the software objects and their attributes. The 'has-a' relationship governs a software object and each of its attributes.

Let us say that an employee software object (at least) has a *name*, a social security (tax) *number*, and regular *pay*.

Other attributes of a software object can be discovered by asking "if I am an employee, what

should I know?" For instance the problem could indicate that an employment *history* is required.

In general the verbs in the problem statement identify the activities (behaviours, actions, responsibilities, operations) required of a software object.

An employee software object at the very least *works* and gets paid; the latter possibly comprising two activities: *receivePay* and *showPay*.

Other activities of a software object can be discovered by asking "if I am an employee, what should I be able to do?" For instance the problem could indicate that *reportWork* is also required.

The state of a software object at any time is given by the values of its attributes, as determined by the software object's activities. For instance showPay should show a higher pay value after receivePay provides a pay rise.

By analogy with a software object, we consider a learning object to have attributes and activities to deliver a single learning objective. Just as the users of a software system (solution) cause interactions between software objects to solve a problem, the students of a 'lesson' can cause interactions between learning objects to achieve the lesson's learning outcomes.

Adapting the above example of the object-oriented approach to software design, consider the overall learning outcome: understand the employee-employer relationship. A learning objective could be to know the responsibilities of employees in an employment hierarchy. We develop an employee learning object below in the same way that we developed an employee software object above.

Importantly, the attributes and activities of a learning object can be identified in the same manner as they were identified for a software object.

For the employee learning object we identify the exact same attributes and activities in the same manner as they were identified for the employee software object above.

By comparison with a software object, the activities of a learning object provide a form of explanation, rather than a computation.

For instance the receivePay activity in the employee learning object could explain that pay is in return for work.

In the object-oriented approach to software design a software object is an instance of a software object class. The process of abstraction enables software objects with shared attributes and activities to be defined as a single software object class. A *software object class* acts as a template from which individualised software objects are instantiated (created), as determined by the user's interactions with the software system.

As a user interacts with a software system that simulates the employee–employer relationship, let us say that employees, *bob*, *ted*, *carol*, and *alice* are instantiated. Each of these software objects has a distinct *name*, social security (tax) *number*, and *pay*. If bob's showPay activity is invoked, the pay value need not be that same as the other employees. If we define an extra *work* attribute and appropriate activities in the employee software object class, all these employee software objects could collaborate to perform their collective work.

Analogously, learning objects can be created as customised instances of a learning object class. A learning object class is not only a container of learning materials for a single learning objective (attributes), but also a container of operations defined on the materials (activities) that a student interacts with to attain the intended learning outcomes.

During a lesson, a student could create employees, *bob*, *ted*, *carol*, and *alice*. Each learning object has at least the distinct attributes identified above. These learning objects could collaborate to explain their *work*.

In general, a student initiates a lesson by interacting with a learning management interface that instantiates a learning object to service the

student's initial request. During the learning object lesson, other learning objects are likely to be instantiated from one or more learning object classes. The student interacts with these collaborating learning objects to attain their desired learning experience.

In the object-oriented approach to software design, abstraction promotes generality, and instantiation provides flexibility and individuality. We assert that the design of learning objects can similarly benefit from essentially the same approach.

The unified modeling language (UML) is a standard for diagrammatically depicting relationships between classes (software object classes), via class diagrams, and between software objects, via object diagrams (Priestley, 1996). Figure 1 shows the Employee software object class in UML and four (4) instance Employee software objects. The attributes and activities of the Employee class are also shown.

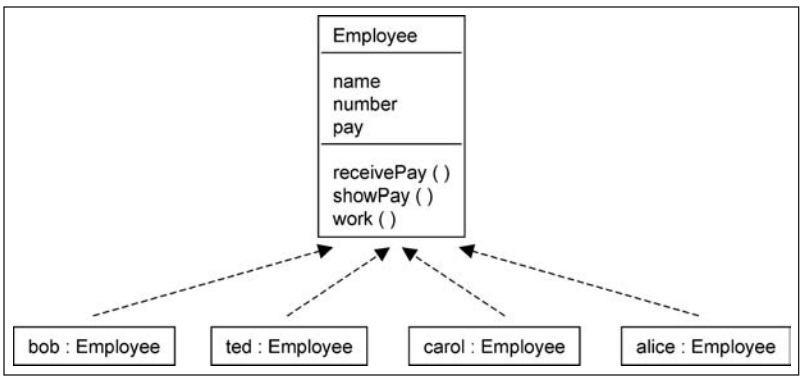
Figure 1 could equally show the Employee learning object class in UML and its four (4) instance Employee learning objects.

Access to the attributes and activities encapsulated by a software object is determined by its software object class. In general, attributes of one software object cannot be directly manipulated by another software object. Instead, the software object encapsulating the attribute in question performs the relevant activity in response to a request from another software object. For instance, an employee software object's pay cannot be directly accessed by other software objects; they can only request an employee software object to showPay or receivePay. Not all activities of a software object need be accessible to other software objects. The public activities supplied by a software object class define an interface that protects the private attributes and activities of its software objects. In effect, software objects request each others' 'services' via the public interface activities supplied by their software object classes. This allows the internals of a software object class to be modified by a programmer without affecting collaborating software objects, provided its interface remains unchanged; for example receivePay could incorporate a bonus without upsetting any software object that requests showPay.

Encapsulation can be equally applied to learning object classes. We assert that encapsulation enhances manageability by facilitating updates without requiring changes to collaborating learning object classes. Other 'bilities' such as reusability and affordability also benefit.

A software object class can be extended into a more specialised software object class without changing the original (parent) class. The 'child' software object class inherits the parent's attributes and activities while encapsulating extra

Figure 1. Class Employee and four instance objects



attributes and activities. The 'is-a' relationship governs a child as a specialised extension of its parent. The child can selectively modify its inherited characteristics too (called polymorphism). For example an Employee software object class could define employment *history* in terms of career achievements. A junior employee could reimplement its history in terms of final school courses and grades. Inheritance enhances reusability and adaptability of software object classes.

An *Employee* software object class can be extended to an *AirlineEmployee* software object class on the one hand, and a *HospitalEmployee* software object class on the other hand. An *AirlineEmployee* could add to its inheritance a knowledge of the *travelIndustry* and *airlinePolicies*. A *HospitalEmployee* could add to its inheritance a knowledge of *healthCareIssues* and *hospitalPolicies*. A *Nurse* software object class could further extend the *HospitalEmployee* with knowledge of *patientCare* and the activities to *takeBloodPressure* and *giveInjection*.

Figure 2. Class Employee and its 'child' classes

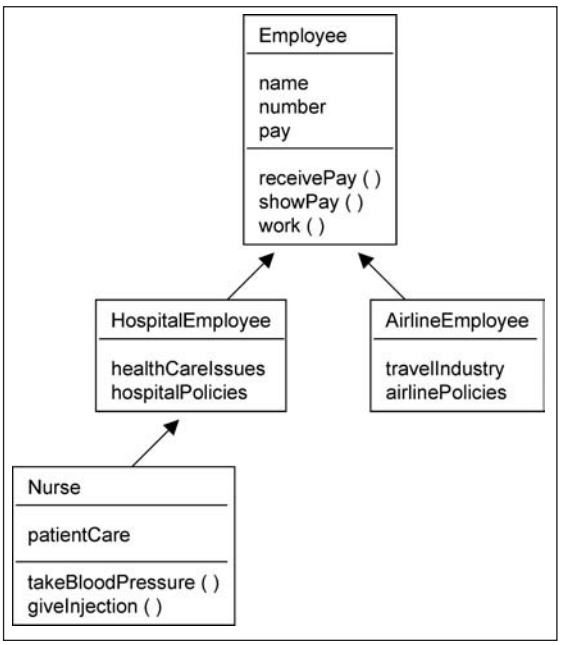


Figure 2 depicts the above inheritance hierarchy in a UML class diagram.

Figure 2 could equally depict an Employee learning object class inheritance hierarchy in a UML class diagram. We assert that inheritance can enhance reusability and extensibility of learning object classes.

Developing Learning Object Lessons as Software Systems

Software systems can be large and complex. Their design can comprise hundreds of software object classes and thousands of interactions of many kinds. Their implementation can amount to millions of lines of code. The software development life cycle (comprising requirements elicitation, analysis, specification, design, implementation, testing, and maintenance) can involve numerous teams of professionals of various kinds over many human years.

On the other hand, most learning objects are designed and implemented by one or two individuals, or a small team, over weeks or months, rather than years (Boyle et al., 2003; MIT OCW, 2005). No more than a handful of learning objects comprise a typical lesson. So there is at least an order of magnitude difference in the current scale of software systems design and learning object lesson design. However, software systems were originally far smaller. As the underlying hardware improved exponentially, it was still the advent of software engineering design methodologies that facilitated production of larger scale reliable software (Pfleeger, 2001). Hopefully, this chapter is a contribution toward a learning object lesson design methodology that will facilitate the design and implementation of larger scale lessons, courses, and educational programs.

An *interface* in object-oriented software engineering terms is the boundary around an object that defines which of its attributes and activities are accessible to other objects. An object's attributes remain private by default, but a public 'accessor'

activity can be defined in an object to return an attribute to any other object that requests it. An object may also define a public 'mutator' activity to enable an attribute to be altered at the request of other objects. Each interaction between two objects is in the form of a request and an answer. Data can be transferred in both directions—in and out. Such interactions in software systems are generally driven by the user(s). In general, objects are dynamically created to provide services in response to user requests.

This software object interaction model is also entirely applicable to a learning object lesson. An instructional designer can define the interface of a learning object class to provide certain learning activities as services. During a lesson, learning objects instantiated from several learning object classes can interact to provide a (student) user responsive learning experience.

The Java language also provides a special interface type that can be used to group a number of classes by insisting each class implements all activities specified by the interface. The grouped classes can be considered to have the same 'look and feel'. We think the Java interface type suggests a mechanism for combining learning object classes into learning object lessons. If a learning object class, L, implements interface I along with other learning object classes, all these learning object classes share a single look and feel. So a (student) user should experience an interactively integrated lesson. If the learning object class, L, also implements another interface, say J, then L can integrate with other learning object classes that implement interface J. This facilitates reuse of learning object class L in a new learning object lesson. Multiple interface types can act as 'wrappers' or 'skins' to provide different contexts for the one learning object class (and its learning objects) in different learning object lessons. This should aid the 'bilities', in particular, reusability and extensibility.

Criteria That Define Object-Oriented Learning Objects

Our above application of object-oriented software engineering design methodology to the design of learning object classes and learning object lessons leads us to synthesise the criteria that define a truly object-oriented learning object class. Our criteria (as follows) refine the learning object definition work of Wiley (2001) and others, and extend learning object design guidelines (Hamel & Ryan-Jones, 2002).

- Each learning object class has attributes and activities that meet a single well-defined learning objective and implement measurable learning outcomes in accordance with an instructional theory.
- Each learning object class and its attributes are identified by the nouns in its learning objective and learning outcomes. The verbs identify its activities.
- Each learning object class encapsulates learning activities that are stand-alone and achievable in a single sitting.
- Each learning object class's attributes and activities contribute packaging metadata (LTSC, 2003), to enhance the 'bilities' (see the "Determining Learning Object Cohesion" subsection).
- Each learning object class in general extends (specialises) its parent learning object class's attributes and activities. Inheritance enhances the 'bilities'.
- Each learning object class in general implements a Java-like interface type (see the "Developing Learning Object Lessons as Software Systems" subsection), which can also be implemented by other learning object classes. A single interface for multiple learning object classes provides the 'look and feel' for a learning object lesson. Implementing

multiple interfaces enables a learning object class to integrate into a different learning object lesson with other learning object classes. Learning object class interfaces enhance the 'bilities'.

Each learning object lesson involves student interaction to create learning objects on demand and to drive interactions between these learning objects to achieve the lesson's learning outcomes. The overall learning activity is necessarily interactive and in general involves (self-)assessment against the learning objectives.

EXAMPLE LEARNING OBJECT CLASSES

In this section we demonstrate our adaptation (see previous) of object-oriented software engineering design methodology to the design of learning object classes. The first example is based on the learning object developed by Boyle et al. (2003) for students to learn how while loops work in the Java programming language. The second example is based on a conflict resolution learning object that explains Maddux's five styles of conflict resolution (Rathsack, 2001).

We assert that our two learning object classes for different disciplines (programming and psychology) demonstrate the general applicability of our object-oriented software engineering approach to the design of learning object classes. Although we write each learning object class in Java, the design is our focus, and is language independent. In the "Cohesion of Example Learning Objects" subsection, we demonstrate the advantages of our design over the originals, as measured by the design principles of coupling and cohesion (elaborated in the "Learning Object Levels of Cohesion" subsection).

While Loop

The Java programming language while-loop learning object of Boyle et al. (2003) starts by showing the student how a program hammers a nail into wood: while (nail is not flush) hit the nail. Next, the Java code to move a car over a given distance is displayed, explained, and the student can animate the loop. The student can also step through the code, statement by statement. A second example shows a submarine submerging to a given depth. The code is displayed, explained, animated, and the student can step through. Then the student is asked to build the code from a given set of Java statements to move a horse a specified distance. Finally, the student is asked to spot errors in the code to move a lorry a given distance.

Our object-oriented software engineering design for a while-loop learning object class is shown (see Figure 3) as an incomplete class in Java. Class While has one attribute—the loop in question, represented as a string of characters. When an object of class While is instantiated, the loop can be initialised to an input string, or the default generic loop. Class While defines the following activities as operations on this loop—display, explain, animate, step_thru, build, and debug.

Figure 3. Class While

```
class While {

   String loop;

   While (String input) {
     // constructor given a loop
     loop = input
   }
   While ( ) {
     // default constructor with generic loop
     loop = "pre-action;" + "while (condition)" + "loop action;" + "post-action";
   }

   display ( );
   explain ( );
   animate ( );
   step_thru ( );
   build ( );
   debug ( );
}
```

The student interacts with a Java program that instantiates requisite While objects. For example, if the student follows the sequence intended in Boyle's learning object, the hammer object would be instantiated first. Its code would be displayed, explained, and animated. Next, the submarine object would be instantiated, its code displayed, explained, animated, and stepped through if desired. Next, the horse object would be instantiated, and the student would build its code. Finally, the lorry object would be instantiated, and the student would debug its code.

Note that another student, perhaps more advanced, could interact with class While to instantiate While objects (learning objects) in another sequence, bypassing some learning objects as desired.

Conflict Resolution

Rathsack's conflict resolution learning object explains Maddux's five styles for managing conflict. The learning object begins by stating that people can disagree, that this can be an opportunity for growth and learning, or it can be detrimental as conflict arises. The ability to manage conflict is important to succeed in one's career and life. The learning object then introduces Maddux's matrix depicting five styles for managing conflict: avoiding, accommodating, winning/losing, collaborating, and compromising. Each style is explained. Then the learning object explains that the matrix y-dimension shows increasing assertiveness and the x-dimension shows increasing cooperation, starting from zero at bottom left. This explains the location of each style in the matrix: winning/losing at top left, collaborating at top right, accommodating at bottom right, avoiding at bottom left, and compromise in the centre. The learning object then presents five conflict scenarios and asks the student to identify the style in use. Finally, the learning object explains that Maddux believes no one style is always best for all situations.

Figure 4. Class Conflict

```
class Conflict {

   String style;

   Conflict (String input) {
      // constructor given a style
      style = input
   }

   display ( );
   explain ( );
   animate ( );
   identify ( );
}
```

Our object-oriented software engineering design for a conflict resolution learning object class is shown (see Figure 4) as an incomplete class in Java. class Conflict has one attribute—the conflict style in question, represented as a string of characters. When an object of class Conflict is instantiated, the style is initialised to an input string. Class Conflict defines the following operations on this style—display, explain, animate, and identify.

The student interacts with a Java program that instantiates requisite Conflict objects. For example, if the student follows the sequence intended in Rathsack's learning object, the 'avoiding' object would be instantiated first. This style would be displayed on the matrix (in relation to the other four styles), explained, and animated on the matrix (in terms of the x and y dimensions). Next, the 'accommodating' object would be instantiated. Similarly, this style would be displayed on the matrix (in relation to the other four styles), explained, and animated on the matrix (in terms of the x and y dimensions). Next, the remaining three Conflict style objects would be instantiated. Finally, the Java program that instantiates Conflict objects would ask the student to identify the style in use in a conflict scenario. The Conflict objects could cooperate to randomise this test.

Note that another student, perhaps more advanced, could interact with class Conflict to instantiate Conflict objects (learning objects)

in another sequence, bypassing some learning objects if desired.

OBJECT-ORIENTED DESIGN PRINCIPLES FOR LEARNING OBJECTS

In a software system, *coupling* measures how cleanly objects are partitioned. And *cohesion* measures how closely activities in an object are related. Coupling and cohesion are interdependent measures—the less cohesive an object, the more likely it is coupled with other objects. The more coupled an object is with other objects, the harder it is to alter or upgrade the object in isolation, which lowers maintainability. A strongly coupled object is less reusable without significant maintenance (Pfleeger, 2001).

In our terminology, the 'objects' referred to above are software object classes, not individually instantiated software objects. Following Boyle (2002), we assert the above applies as much to learning object classes in a learning object lesson as to software object classes in a software system. Weak coupling between learning object classes in a learning object lesson promotes the 'bilities' in that the maintainability of software object classes is essentially the manageability of learning object classes. The more cohesive each learning object class is, the less coupling is required when learning object classes are reused in a new learning object lesson.

Stevens and Myers devised a table (Yourdon & Constantine, 1978) to classify the level of cohesion of a software module—in our terminology, a software object class. (Although their work predated object-oriented software engineering design methodology, it is readily accommodated.) In the "Learning Object Levels of Cohesion" subsection we adapt their seven-level scale to learning object classes. In the "Determining Learning Object Cohesion" subsection, we show how to classify the level of cohesion of a learning object class. We explain how each of the lower levels further reduces learning object class reusability. We assert that awareness of cohesion levels can improve the design of learning object classes, as we illustrate in the "Cohesion of Example Learning Objects" subsection with the examples from the "Example Learning Object Classes" section.

Learning Object Levels of Cohesion

In Table 1 we adapt a seven-level scale of cohesion for software object classes (Yourdon & Constantine, 1978) to learning object classes. We describe a learning object class at each level and provide an example. The strongest level of cohesion is called *functional*, and the weakest cohesion is called *coincidental*. Each level in Table 1 is less cohesive that the level above it.

The reusability issue for each level below functional cohesion is explained in Table 2. Note that the issue at a given level is often in addition to reusability issues at higher levels.

Determining Learning Object Cohesion

The description or name of a learning object class may suffice to determine its level of cohesion, as shown in Table 3.

We note that the presence of any of the above key words could be a valuable indicator for metadata tagging purposes, but further research is required to evaluate reliability.

In Figure 5, we adapt a decision tree (Page-Jones, 1998) that enables the level of cohesion of a learning object class to be more accurately determined by asking and answering a few questions, starting at top left.

If all the activities in a learning object class share more than one level of cohesion, the learning object class has the highest (strongest) of the shared levels of cohesion—according to the 'chains in parallel' rule. If the activities in a learning object class exhibit various levels of

Table 1. Learning object class cohesion levels

Cohesion level	Description	Example
Functional (strongest)	Each activity in a learning object class contributes to a single learning objective related task or learning outcome.	Learn to calculate net employee salary. This could be one of many tasks an accountant learns. It comprises: getting the gross salary, subtracting legal deductions, computing taxes. Every step contributes to the single purpose outcome of this learning object class.
Sequential	The outcome (output) of each activity in a learning object class is the input to the next activity in the learning object class.	Learn to paint a picture. This learning object class could comprise: sketching, painting outlines, coloring shapes, adding texture, signing, and dating. Each activity uses the result of the previous activity on the canvas. The picture may be complete, but learning to paint could still be a life-long objective, so the learning object class is not functionally complete.
Communicational	The activities in a learning object class share the same attributes, or inputs and outputs.	Learn to summarise, say, a chapter of a book. This learning object class could comprise: reading the chapter, highlighting headings in the chapter, listi;ng key words in the chapter, writing sentences that connect key words in the chapter. Each activity uses the chapter, but not necessarily the result of the previous activity.
Procedural	Control flows from one activity to the next in the learning object class, i.e., the activities are related solely by their order of execution, which is arbitrary. Data passing in and out of the learning object class are unrelated.	Learn to dissect, say, a fish or mouse. This learning object class could comprise: cleaning the bench, arranging implements, preparing the specimen, starting experimental notes, using scalpel, recording observations. Each activity leads to the next, but it does not necessarily use the result of any previous activity.
Temporal	The activities in a learning object class are related in time only, i.e., the activities are executed at about the same time.	Learn to study. This learning object class could comprise: turning off the radio and TV, collecting pen, paper, and textbook, working at one's desk, ignoring phone calls and other distractions, making notes, etc. All these activities occur during study time, but they need not occur in this exact order.
Coincidental (weakest)	The activities in a learning object class are unrelated by any of the above.	Learn to tidy up a room. This learning object class could comprise: disposing of litter, hanging clothes, finishing a snack, making the bed, vacuuming, etc. Not all these activities need be done (together). The activities are not logically related, nor connected by flow of execution or data.

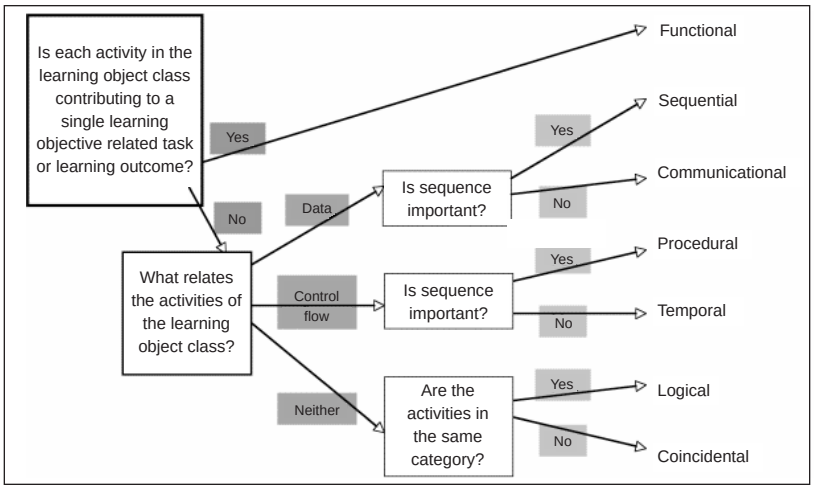
Table 2. Learning object class cohesion levels and reusability

Cohesion level	**Reusability issue**
Sequential	Not as reusable as a functional learning object class because the sequencing of its activities cannot be easily altered.
Communicational	Either the input or output coupling is generally broader than for the above levels of cohesion. Reuse often needs a subset of this coupling, hence redundant coupling; or a cut down version of the learning object class is created, which still duplicates functionality. A communicational learning object class can often be split into functional learning object classes.
Procedural	Intermediate or partial results are often passed into or out of a procedural learning object class, reducing reusability. It is tempting to combine distinct activities for 'efficiency' or 'convenience' further reducing reusability.
Temporal	Activities in a temporal learning object class tend to be related to activities in other learning object classes, increasing coupling. Activities in a temporal learning object class are often combined because they can occur together. But this compromises reusability in another situation where the activities can occur at different times.
Logical	Broad input coupling is required for a logical learning object class to select which activity to perform. The activities are typically combined because they share common parts. Reusability suffers.
Coincidental	A combination of inputs often determines the selected activity. As a result it can be hard to understand a coincidental learning object class unless its internal detail is examined. This reduces reusability.

Table 3. Determining learning object class cohesion by its name or description

Cohesion Level	**Name or Description**
Functional	Simple verb-object phrase.
Sequential	Commas often required.
Communicational	The word 'and' is often present.
Procedural	The word 'or' is often present, or words synonymous with repetition, e.g., 'while', 'until'.
Temporal	Time related words apparent, e.g., 'start', 'end', 'before', 'after'.
Logical	An 'umbrella' word is present, e.g., 'all', 'every', 'total'.
Coincidental	Description or name is meaningless, e.g., 'miscellaneous', 'X', 'Z-process'.

Figure 5. Cohesion decision tree



cohesion, the learning object class has the lowest (weakest) level of cohesion—according to the 'chains in series' rule.

Cohesion of Example Learning Objects

In this section we use Figure 5 to establish the cohesion of the learning object examples in the "Example Learning Object Classes" section. Similar analysis of other learning objects could establish their levels of cohesion, and hence their reusability.

When we address the first question (top left of Figure 5) for Boyle et al.'s Java while-loop learning object, the answer at first appears to be 'yes'—the While learning object appears functionally cohesive in that each activity contributes to understanding how while loops work, which is the learning object's learning objective. But on closer examination, the activities performed by the While learning object do not contribute to one and only one learning objective related task. The learning object performs two pairs of similar analysis activities in sequence (hammer and submarine, horse and lorry), followed by one synthesis activity. Data are not passed between the activities, so the sequence is program controlled (i.e., chosen by the instructional designer). So the learning object exhibits procedural cohesion at best. If we similarly address our object-oriented software engineering design for a while loop learning object class, shown in Figure 3, a learning object can be instantiated for each of hammer, submarine, horse, and lorry. The sequence is determined by the student (via input data), so at least the learning object class exhibits communicational cohesion, which is better than procedural cohesion.

The situation is similar for Rathsack's conflict resolution learning object and our Conflict learning object class shown in Figure 4. When we address the first question (top left of Figure 5) for Rathsack's learning object, the answer at first appears to be 'yes'—the learning object appears functionally cohesive in that each activity contributes to understanding Maddux's five styles for managing conflict, which is the learning object's learning objective. But after closer examination, the activities performed by the Conflict learning object do not contribute to one and only one learning objective related task. The learning object

performs five similar activities in sequence, one for each style. Data are not passed between the activities, so the sequence is program controlled (i.e., chosen by the instructional designer). So the learning object exhibits procedural cohesion at best. If we similarly address our object-oriented software engineering design for a Conflict learning object class, shown in Figure 4, a learning object can be instantiated for each of the five styles. The sequence is determined by the student (via input data), so at least the learning object class exhibits communicational cohesion, which is better than procedural cohesion.

Since each object-oriented software engineering designed learning object class exhibits stronger cohesion than the original learning object, our learning object class is likely to require weaker coupling with other learning object classes in a new learning object lesson, thereby enhancing its reusability.

SUPPORT FOR OBJECT-ORIENTED LEARNING OBJECTS

We henceforward refer to our above described object-oriented software engineering approach to designing learning objects and learning object lessons as 'object-oriented learning'. We claim benefits of object-oriented learning include the following, in addition to enhancing the 'bilities'. Dynamic instantiation of a learning object from its learning object class in response to a student's input/choice enables a learning object lesson to not only be more highly interactive but also far less predetermined in its sequence of activities for each student. Also the instructional designer need not build lessons as a predetermined sequence of learning object classes, as the student can be given more choices.

Below we investigate supports available for our object-oriented learning approach. We outline how object-oriented learning can be accommodated in a learning object development project from initial application of an instructional design theory to final implementation in a learning management system (e.g., Blackboard, WebCT, Moodle).

Object-Oriented Learning and Instructional Design Theory

It is reasonable that the design of learning objects and learning object lessons should be informed by an instructional design theory (Wiley, 2001). After these pedagogical choices are made, an instructional designer can apply our object-oriented learning approach, where the focus is on structuring the functionality provided by learning object classes. We assert that object-oriented learning is independent of, and complementary to, instructional design theory. Further research and development is needed to confirm this.

In fact we are presently using object-oriented learning to design a substantial learning object lesson, which is composed of several learning object classes. We intend to use UML during the design process in order to report on its usefulness as a tool for communicating the design of a learning object lesson. But the focus of our study will be on evaluating the effectiveness of our object-oriented learning approach on learning object class 'bilities'. We also expect to demonstrate most of the objectives of this chapter.

Object-Oriented Learning and Learning Management Systems

- Learning management systems like Blackboard, WebCT, and Moodle currently facilitate the development of e-learning courses, their collection in a repository, and their delivery to online students at anytime over any distance. Learning management systems are also starting to support standardised metadata tagging of learning objects to better facilitate the combination of learning objects into lessons, courses, and educational programs. Our object-oriented

learning approach contributes to learning object metadata tagging as explained in the "Determining Learning Object Cohesion" subsection. We contend that our approach is automatically accommodated within the pedagogically neutral standards adopted by the IEEE (LTSC, 2003) for learning object packaging.

- Learning management systems will also need to provide the 'programming language' that enables instructional designers using object-oriented learning and students to realise the full potential of the dynamics inherent in the design of learning object classes. Our use of Java in the "Example Learning Object Classes" section was not only to illustrate the application of object-oriented software engineering to the design of learning objects. Java can also be the programming language used in a learning management system to implement student-centered combination of learning object classes and the dynamic instantiation of their learning objects. However, the power of a general programming language environment such as Java is not necessary for this purpose. Indeed, further research is desirable to produce a complete yet simple programming tool for instructional designers to use across learning management systems. One avenue to explore is the programmability introduced into the computer aided instruction systems of the past (Gibbons & Richards, 2001). Fortunately, today's graphic user interfaces, more powerful computers, and faster connectivity will make the experience far more friendly for both today's instructional designers and students.

CONCLUSION

We have applied object-oriented software engineering design methodology to the design of learning objects. Our object-oriented learning approach extends and refines Boyle's (2002) design principles for authoring dynamic reusable learning objects as follows. The attributes and activities of a prospective learning object are first 'abstracted' into a learning object class. This facilitates dynamic instantiation of an individualised learning object for, or by, a student during a lesson; as we illustrated in the "Example Learning Object Classes" section with two example learning object classes for different disciplines. We introduced the unified modeling language to illustrate the learning object class design process (Figures 1 and 2). We showed how inheritance and polymorphism further enhance learning object class reusability in new lessons.

We explained how our object-oriented learning approach can benefit the design of lessons comprising several learning object classes. We identified a Java-like interface type as a useful mechanism to assist reuse and repurposing of learning object classes into new learning object lessons. This application of object-oriented software engineering design methodology led us to synthesise the criteria that define a truly object-oriented learning object class. Our criteria refine the learning object definition work of Wiley (2001) and others, and extend learning object design guidelines (Hamel & Ryan-Jones, 2002).

We adapted to learning object classes a scale for grading the cohesion of software modules (Yourdon & Constantine, 1978). We showed in the "Object-Oriented Design Principles for Learning Objects" section how to classify the level of cohesion of a learning object class, and described how each of the lower levels further reduces reusability. To illustrate we outlined two object-oriented software engineering designed learning object classes that exhibit stronger cohesion than the original learning objects. These learning object classes are likely to require weaker coupling with other learning object classes in a new learning object lesson, thereby enhancing reusability. Our adaption of cohesion levels to learning objects

further extends and refines Boyle's design principles for authoring dynamic reusable learning objects (Boyle, 2002).

We explained in the "Support for Object-Oriented learning objects" section how our object-oriented learning approach is independent of, and complementary to, (a) the instructional design theory underlying the learning object design process, and (b) the metadata standards adopted by the IEEE (LTSC, 2003) for learning object packaging. Pedagogical decisions can be made by the instructional designer before applying our approach to structuring the functionality of learning object classes. Our object-oriented learning approach is pedagogy neutral in this respect. Our approach also contributes to learning object metadata tagging by identifying attributes and activities for each learning object class.

We assert that our object-oriented learning approach expands and informs the options available to instructional designers for structuring and interfacing learning object. We believe our approach assists the systematic development of more complex, authentic lessons, composed of student-centered, dynamically created learning objects. We expect further contributions toward an object-oriented learning methodology will facilitate the design and implementation of larger scale lessons, courses and educational programs, composed of learning objects that increasingly exhibit the 'bilities'.

REFERENCES

Boyle, T. (2002). Design principles for authoring dynamic, reusable learning objects. In A. Williamson, C. Gunn, A. Young & T. Clear (Eds), *Winds of change in the sea of learning: Proceedings of the 19th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education* (pp. 57-64). Auckland, New Zealand: UNITEC Institute of Technology. Retrieved February 19, 2007, from http://www.ascilite.org.au/conferences/auckland02/proceedings/papers/028.pdf

Boyle, T., Chalk, P., Fisher, K, Jones, R., Bradley, C., Haynes, R., et al. (2003). *Example learning objects.* Retrieved February 19, 2007, from http://www.londonmet.ac.uk/ltri/learningobjects/examples.htm

Computer Education Management Association (2001). *Learning architecture learning objects overview.* Retrieved February 19, 2007, from http://learnativity.com/lalo.html

Gibbons, A., & Richards, R. (2001). The nature and origin of instructional objects. In D. Wiley (Ed.), *The instructional use of learning objects.* Retrieved February 19, 2007, from http://www.reusability.org/read/

Hamel, C. J., & Ryan-Jones, D (2002, November). Designing instruction with learning objects. *International Journal of Educational Technology, 3*(1). Retrieved February 19, 2007, from http://www.ed.uiuc.edu/ijet/v3n1/hamel/index.html

Littlejohn, A. (Ed.). (2003) *Reusing online resources: A sustainable approach to e-learning.* London: Kogan Page.

LTSC—IEEE Learning Technology Standards Committee. (2003). *Standard for learning object metadata* (LOM P1484.12). Retrieved February 19, 2007, from http://ltsc.ieee.org/wg12/index.html

Morris, E. (2005). Object oriented learning objects. *Australasian Journal of Educational Technology, 21*(1), 40-59. Retrieved February 19, 2007, from http://www.ascilite.org.au/ajet/ajet21/morris.html

Morris, E. J. S., & Zuluaga, C. P. (2003). Educational effectiveness of 100% online I.T. courses. In G. Crisp & D. Thiele (Eds.), *Interact : Integrate : Impact: Proceedings of the 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education* (pp. 353-363). Adelaide, Australia: Adelaide University. Retrieved

February 19, 2007, from http://www.adelaide.edu.au/ascilite2003/program/conf_prog_index.htm

MIT OCW. (2005). *MIT OpenCourseWare process.* Retrieved February 19, 2007, from http://ocw.mit.edu/OcwWeb/Global/AboutOCW/publication.htm

Page-Jones, M. (1998). *The practical guide to structured systems design* (2nd ed.). Retrieved February 19, 2007, from Wayland Systems Inc., http://www.waysys.com/ws_content_bl_pgssd_ch06.html

Pfleeger, S. L. (2001). *Software engineering theory and practice* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Priestley, M. (1996). *Practical object-oriented design with UML*. Maidenhead, Berkshire, UK: McGraw-Hill International.

Rathsack, R. (2001). *Conflict resolution styles*. Retrieved February 19, 2007, from http://www.wisc-online.com/objects/index.asp?objID=PHR300

Wiley, D. (2001). Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. In D. Wiley (Ed.), *The instructional use of learning objects.* Retrieved February 19, 2007, from http://www.reusability.org/read/

Yourdon, E., & Constantine, L. (1978). *Structured design: Fundamentals of a discipline of computer program and systems design* (Yourdon Press Computing Series). Englewood Cliffs, NJ: Prentice Hall.

Zuluaga, C. P., & Morris, E. J. S. (2003). A learning management model for mixed mode delivery using multiple channels (Internet, Intranet, CD-ROM, Satellite TV). In G. Crisp & D. Thiele (Eds.), *Interact : Integrate : Impact: Proceedings of the 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education* (pp. 562-568). Adelaide, Australia: Adelaide University. Retrieved February 19, 2007, from http://www.adelaide.edu.au/ascilite2003/program/conf_prog_index.htm

Zuluaga, C. P., Morris, E. J. S., & Fernandez, G. (2002). Cost-effective development and delivery of 100% online I.T. courses. In A. Williamson, C. Gunn, A. Young & T. Clear (Eds.), *Winds of change in the sea of learning: Proceedings of the 19th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education* (pp. 759-766). Auckland, New Zealand: UNITEC Institute of Technology. Retrieved February 19, 2007, from http://www.ascilite.org.au/conferences/auckland02/proceedings/papers/109.pdf

This work was previously published in Learning Objects for Instruction: Design and Evaluation, edited by P. Northrup, pp. 70-93, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 2.13
Covert End User Development:
A Study of Success

Elaine H. Ferneley
University of Salford, UK

ABSTRACT

End User Development (EUD) of system applications is typically undertaken by end users for their own, or closely aligned colleagues, business needs. EUD studies have focused on activity that is small scale, is undertaken with management consent and will ultimately be brought into alignment with the organisation's software development strategy. However, due to the increase pace of today's organisations EUD activity increasing takes place without the full knowledge or consent of management, such developments can be defined as covert rather than subversive, they emerge in response to the dynamic environments in which today's organisations operate. This paper reports on a covert EUD project where a wide group of internal and external stakeholders worked collaboratively to drive an organisation's software development strategy. The research highlights the future inevitability of external stakeholders engaging in end user development as, with the emergence of wiki and blog-like environments, the boundaries of organisations' technological artifacts become increasingly hard to define.

INTRODUCTION

In today's environment of rapid business change facilitated by users with increased technical capabilities, there is a tacit understanding that end user development (EUD) activity is inevitable—development tools are more accessible, and end users are now technologically mature and expected to be proactive in their use of technology to enable enactment of their employment roles (Jawahar & Elango, 2001; Nelson & Todd, 1999). As the end user takes control of the development effort and develops systems with little or no input from information technology (IT) specialists so the ultimate level of end user involvement has arrived—the end user is no longer simply consulted, they have assumed the roles of the designer, developer and tester, they are the IT specialist for their software requirement (Cheney, Mann, & Amoroso, 1986; McGill, 2004).

To date, studies have focused on EUD that management is fully aware of and endorses, the assumption is that EUD activity is small scale and that it will ultimately be brought into the organisation's software development strategy.

However, due to the increased pace of today's organisations, EUD activity increasingly takes place without the full knowledge or consent of management. Such developments can be defined as covert rather than subversive, and it can be argued that they emerge in response to the dynamic environments in which today's organisations operate (Nelson & Todd, 1999; Ouellette, 1999; McLean, Kapperlman, & Thompson., 1993).

This paper reports on a field study on the effects of covert EUD activity in a publishing company. The paper aims to enhance our understanding of covert EUD activity using an interpretive approach. We draw on the literature on the social construction of technology (SCOT) and apply this to covert EUD activity identifying a technology "path" (MacKenzie & Wajcman, 1985). The "path" may be born from an individual vision, but the multifaceted nature of technology requires disparate actors to contribute to technology success. Whilst the paper does not purport to offer definitive solutions, the experiences reported suggest valuable lessons for organisations faced with the challenge of managing the dichotomous relationship of encouraging worker proactivity manifested in EUD whilst controlling maverick EUD activity.

LITERATURE REVIEW

Authors have begun to recognise the futility of attempting to align business strategy and technological infrastructures and have acknowledged that technological "drift" is inevitable, (Ciborra et al., 2000; Sauer & Burn, 1997; Ciborra, 1994; Orlikowski, 1996). This process of "drift" is largely assumed to be an overt process, management being aware that it is happening and either attempting realignment (usually futilely) or allowing the technology to develop a certain momentum of its own (for examples see Kanellis & Paul, 2005; Hanseth & Braa, 1998; Rolland & Monteiro, 2002). What is less frequently considered is the notion of, and rationale for, covert IT implementations that result in "drift," and the literature that does exist is primarily concerned with covert activity with the intention of sabotage (for examples see Gordon, 1996; Conti, 2005; Verton, 2001; Graham, 2004).

Such covert activity, whether for altruistic or subversive purposes, necessitates a degree of improvisation—using current resources to create new forms and order from tools and materials at hand, such an approach has been defined by anthropologists as "bricolage" (Levi-Strauss, 1966). When considering information systems bricolage, "materials at hand" are usually considered to be information technology hardware and software artefacts. However, it has also been suggested that the use of networking with preexisting professional and personal contacts is also a form of "network bricolage" (Mintzberg, 1994; Moorman & Miner, 1998; Baker, Miner, & Eesley, 2003).

RESEARCH DESIGN

To examine covert EUD activity within an organisation from multiple stakeholder perspectives requires an understanding of the social and contextual relationships that influence the organisation; there can be no single explanation of success. Our epistemological assumptions are that no individual account of social reality can be proven correct. Therefore, the research method employed has been interpretivist, with the aim being to understand the perspectives of the various stakeholders and the historical and socially situated contexts in which they reside (Hirschheim, Klein, & Lyytinen, 1996). The opportunity to gain access to a covert end user developer group emerged during the course of a wider longitudinal study that was undertaken by the author and a postgraduate student that examined the effect on various stakeholders of implementing IT solutions in small- to medium-sized enterprises (SMEs). The postgraduate student was employed by one

of the studied SMEs (PublishCo) and became one of the covert EUDs, simultaneously exploring the dynamics and rationale for EUD whilst repeatedly intervening and stimulating change. Therefore, the research employed participatory action research: "…members of the organisation we study are actively engaged in the quest for information and ideas to guide their future actions" (Whyte, Greenwood, & Lazes, 1991). The research largely complied with the major characteristics of information systems action research as identified by Baskerville (1999) and Baskerville and Wood-Harper (1996). However, there is one area in which the researchers did not fully conform to the established principles for action research: informed consent. Whilst formal consent from PublishCo's management to study the effects of the introduction of IT into the organisation was gained, as the research evolved into a more specific study on covert EUD the researchers were faced with a moral dilemma—whether to inform management of the covert activity or whether to abuse management confidence and collude rather than confess. During the rest of this paper we will discuss how this dilemma was accommodated, and it is up to the reader to decide if we acted morally.

Field Study: PublishCo

PublishCo is a small publishing house established in 1895, and it is involved in the publication of niche specialist magazines for the pet owning, breeding, and exhibiting community. Circulation is global, although the customer base at the start of the fieldwork was largely European. Over the course of 4 years data has been collected from multiple levels and perspectives across the organisation, from technology suppliers and from customers. The findings are the results from 4 years of field diaries, over 200 hours of transcribed interviews and numerous hours of observation.

This paper reports on the covert development of a Web-based portal that has resulted in system integration across the company, and has contributed to increasing PublishCo's revenue by approximately 60%, improved the efficiency of the company's processes, built better customer relations, and simultaneously improved the working lives and morale of its employees. A summary of

Table 1. Emerging characteristics, roles and influences for stakeholders in covert end user development

Stakeholder Groups	**Predevelopment**	**System Development**	**System Implementation**
Internal End User Developer	**Characteristics** Entrepreneurial Personal or financial reward **Role/influences** Participative approach Enrolling management Revision of organisation and IS, based on previous external experience, current education, etc. **Quote** *"Web publishing is the way forward for all niche magazines"* (Emp1). *"We've only got a single web page with our contact details… it's too dated"* (Emp2)	**Characteristics** Risk taking. **Role/influences** External stakeholders—developer /software vendor, customers, competitors Access to technology—beta-software, packages External promotional events and training **Quote***"There was so much to implement and limited resources so illegal software was the only solution"* (Emp3)	**Characteristics** Efficiency gains Technical rationalisation (networking, desktop, etc.) **Role / influences** Recognition of need to manage & integrate technology Upgrade to reflect changing use **Quote** *'as functionality was released customers started using it straight away, it was obvious immediately that the decision to move online was the right one … we're always adding new things, it changes almost weekly* (Emp2)

(continued on following page)

Table 1. continued

Management	May not be aware EUDs occurring **Characteristics** Willingness to listen to strategic technologist **Role/influences** Trust change agent—internal or external Change-friendly climate Encourage autonomous "can do" employees **Quote** *"We were really wary of the developing a comprehensive website, especially as the dotcom catastrophe was just happening"* (Man2)	**Characteristics** Radical change (large or small scale) Business-led development **Role/influences** Acceptance of new direction Commit to technology education Technology as +ve lever for social change within the organisation **Quote** *"We can not envisage our customers using a website, their largely older, we don't expect them to be interested in an online site … we regard the development as a necessary commercial distraction"* (Man1)	**Characteristics** Financial commitment Technology management **Role / influences** Availability of financial support for ongoing implementation Ensure legality of EUD systems Vision alignment with business, organization and technology strategies **Quote** *'the web site has made a significant difference to our business, the discussion forums keep all of us abreast of what's going on in* [the field] *and we're* [management] *now actively contributing to those discussions'* (Man2)
External IT Agencies	**Characteristics** Long term commitment to organisation End user focussed **Role/influences** Long term financial gain Participative approach **Quote***"We needed to develop a reference site, so PublishCo implementation was an ideal opportunity"* (Salesman3)	**Characteristics** Exploratory, risky development (large or small scale) **Role/Influences** Technology & training provider **Quote** *"At the time we didn't see the effect that using lots of different tools would have, we wanted a reference site and believed using those tools was a means to an end"* (Salesman3)	**Characteristics** Educator **Roles / Influences** Training Emerging technology vision **Quote** *'we expected to use the site as a sales example but the end users have been using all sorts of free resources to cobble together a site that's not as professional looking as we'd hoped for'* (Salesman2
External End Users/ Customers	**Characteristics** Innovators **Role/influences** Stimulate technology vision **Quote** *"Online dog breeding forums were starting to emerge but really very few dog breeders were talking on them"* (Customer2)	**Characteristics** Reviewers Testers **Role/influences** External influences	**Characteristics** Explorers **Role / influences** Technology 'path' leaders **Quote** *'the portal effectively gave a voice to the community'* (Customer5)

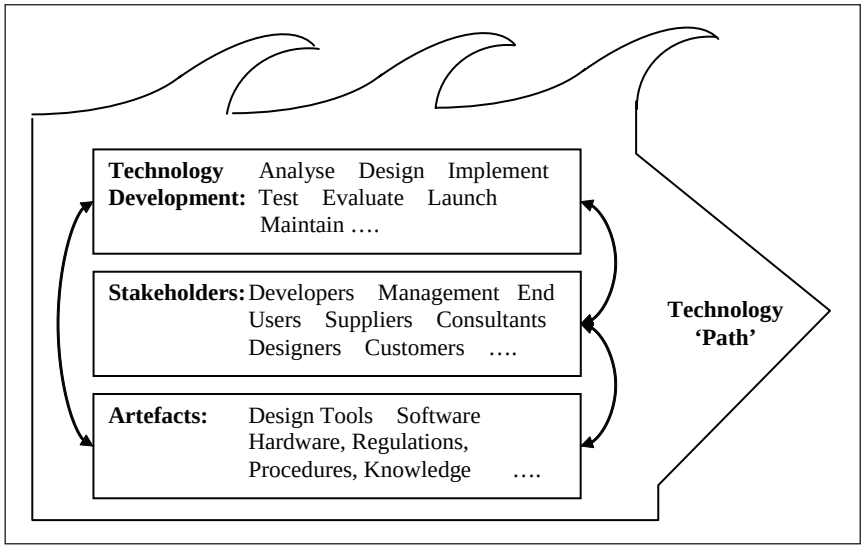
the various stakeholders' roles and perspectives is presented in Table 1.

IMPLICATIONS OF COVERT END USER DEVELOPMENT

Within the limitations of this field study, by considering the rationale and world views of these disparate groups who have all embraced EUD as a means of moving down a technology "path," a model for harnessing end user development activity emerges. The internal covert developers viewed the technology as emancipatory. They saw the technology as an opportunity to change the organisation's business model and introduce a new revenue stream, thereby potentially increasing sales turnover and hence commission. However, in the portal development the technical artefacts themselves became the technology path driver,

and as the end user developers gained enthusiasm and technical knowledge they introduced more, disparate functionality into the portal, and the technology developed a momentum of its own. These disparate elements now define the portal—its quirky, eclectic style clearly differentiates it from the competition and have made it the market leader. The IT consultants also assumed the role of covert developers. The usual assumption is that IT consultants view technical implementations as a source of income, yet in the presented study the implementation also provided them with a learning opportunity. They were prepared to work with internal, budding developers to provide them with the requisite skills to ensure the systems could evolve with the business. However, they also actively deceived management by presenting illusionary demonstrations and condoned or instigated the use of illegal software. Management's interpretation of the emerging technology was as a distraction and cost overhead. The end users and consultants perceived that they had limited understanding of the presentations and conversations that took place regarding technology, and subsequent interviews confirmed that members of management had sought advice from family members, personal friends, or peer networks rather than question their employees or contracted IT consultants. However, it emerged that they were aware that some covert development was occurring and chose to allow it to remain underground rather than stifle the proactivity that their employees were demonstrating. Customers' interpretation of the technologies has been as communication enabler. The portal facilitates communication in many directions: customer to customer, customer to journalist, customer to management, customer to advertiser. Indeed, it could be argued that the customer has become the final technology path driver: Functionality is added or removed from the portal dependent on customer use, editorial is driven by their discussion forum conversations, whilst the content of the online directories is created and uploaded by the customers themselves. As the portal develops so it is interpreted in different ways, including a sales forum (extensive online directories), an organisational tool (calendar of

Figure 1. Technology path



events), and a dating agency (putting readers with similar requirements in touch for the purposes of pursuing their niche interest).

Reflecting on the findings of the field study, our inquiry illustrates that (a) covert EUD may be driven by a wider stakeholder group than the end user developers themselves, and (b) although EUD activity is traditionally seen as a microlevel activity, the challenge is to harness the developing technical artefacts to achieve maximum business benefit without jeopardizing the stability of the organisation as a whole. We discuss these implications in turn.

The traditional view of EUD is that the activity is small scale and undertaken to satisfy end users' own needs. Yet as end users become more technically sophisticated and as technology itself becomes more homogeneous, user friendly and reconfigurable, so a wider set of stakeholder groups are gaining common or easily transferable technical know-how. This allows traditionally disparate stakeholder groups from both inside and outside organisations to act collectively in covert activity; in the field study, transferred technological know-how enabled the internal end users and the technology vendors to work together to covertly developed information systems solutions. Indeed, whilst the ultimate level of end user involvement has traditionally been seen as the end user taking ownership of the system and becoming the developer himself (Cheney et al., 1986), it is feasible for external stakeholder groups to effectively become the end user developer. In the field study, as the customer stakeholder group has gained technical know-how, it has become able to drive the portal's ever changing design, and it provides the content in the form of uploaded reviews, customer profiles, and commentary via discussion forums. Indeed, the portal could be viewed as an emerging wiki or blog-like environment owned by PublishCo (Leuf & Cunningham, 2001).

REFERENCES

Baker, T., Miner, A. S., & Eesley, D. T. (2003). Improvising firms: Bricolage, account giving and improvisational competencies in the founding process. *Research Policy, 32*, 255-276.

Baskerville, R. (1999). Investigating information systems with action research. *Communications of the Association for Information Systems, 2*.

Baskerville, R., & Wood-Harper, A. T. (1996). A critical perspective on action research as a method for information systems research. *Journal of Information Technology, 11*, 235-246.

Cheney, P. H., Mann, R. I., & Amoroso, D. L. (1986). Organizational factors affecting the success of end user computing. *Journal of Management Information Systems, 3*, 65-80.

Ciborra, C., Braa, K., Cordella, A., Dahlbom, B., Failla, A., Hanseth, O., Hepso, V., Ljungberg, J., Monteiro, E., & Simon, K. A. (2000). *From control to drift*. Oxford: Oxford University Press.

Conti, G. (2005). Why computer scientists should attend hacker conferences. *Communications of the Association for Information Systems, 48*, 23-24.

Dosi, G. (1982). Technological paradigms and technological trajectories. *Research Policy, 11*, 147-162.

Gordon, S. (1996). In *The 6th International Virus Bulletin Conference*Brighton, UK.

Graham, P. (2004). *Hackers and painters: Big ideas from the computer age*. O'Reilly.

Hanseth, O., & Braa, K. (1998). In R. Hirschheim, M. Newmann, & J. I. DeGross (Eds.), *Proceedings of the 19th International Conferenceon Information Systems* (pp. 188-196). Helsinki, Finland.

Hirschheim, R., Klein, H. K., & Lyytinen, K. (1996). Exploring the intellectual structures of information systems development: A social action

theory analysis. *Accounting, Management and Information Technologies, 6,* 1-64.

Jawahar, I. M., & Elango, B. (2001). The effect of attitudes, goal setting and self efficacy on end user performance. *Journal of End User Computing, 13,* 40-45.

Kanellis, P., & Paul, R. J. (2005). User behaving badly: Phenomena and paradoxes from an investigation into information systems misfit. *Journal of Organizational and End User Computing, 17,* 64-91.

Kemp, R., Schot, J., & Hoogma, R. (1988). Regime shifts to sustainability through process of niche formation: the approach of strategic niche management. *Technology Analysis & Strategic Management, 10,* 175-195.

Leuf, B., & Cunningham, W. (2001). *The wiki way: Quick collaboration on the Web.* Addison Wesley Longman.

Levi-Strauss, C. (1966). *The savage mind*. Oxford: Oxford University Press.

MacKenzie, D., & Wajcman, J. (1985). *The social shaping of technology.* Open University Press.

McGill, T. (2004). The effect of end user development on end user success. *Journal of Organizational and End User Computing, 16,* 41.

McLean, E. R., Kapperlman, L. A., & Thompson, J. P. (1993). Converging end user and corporate computing. *Communications of the Association for Information Systems, 36,* 76-91.

Mintzberg, H. (1994). *The rise and fall of strategic planning.* New York: Free Press.

Moorman, C., & Miner, A. S. (1998). Organizational improvisation and organizational memory. *Academy of Management Review, 23,* 698-723.

Nelson, R. R., & Todd, P. (1999). Strategies for managing EUC on the Web. *Journal of End User Computing, 11,* 24-31.

Orlikowski, W. J. (1996). Improvising organizational transformation over time: A situated change perspective. *Information Systems Research, 7,* 63-92.

Ouellette, T. (July 26, 1999). Giving users the keys to their Web accounts. *Computerworld,* 66-67.

Rolland, K. H., & Monteiro, E. (2002). Balancing the local and the global in infrastructural information systems. *The Information Society, 18,* 87-100.

Sauer, C., & Burn, M. J. (1997). The Pathology of Alignment. In C. Sauer & P. Yetton (Eds.), *Steps to the Future*. San Francisco: Jossey Bass.

Verton, D. (2001). Hacker conferences highlight security threats. *PC World*.

Webopedia (2005). Retrieved September 14, 2006, from www.webopedia.com

Whyte, W. F., Greenwood, D. J., & Lazes, P. (1991). In W. F. Whyte (Ed.), *Participatory action research* (pp. 19-55). Newbury Park, CA: Sage.

ENDNOTES

1 Alternative terms in the literature are "regimes" (Kemp et al, 1998), or "trajectories" (Dosi, 1982)

2 A Wiki is a collaborative Web site comprised of the perpetual collective work of many authors that anyone is allowed to edit, delete, or modify. A blog is similar in structure but does not allow visitors to change the original posted material, only to add comments to the original content (Webopedia, 2005).

This work was previously published in Journal of Organizational and End User Computing, Vol. 19, Issue 1, edited by M. Mahmood, pp. 62-71, copyright 2007 by Idea Group Publishing (an imprint of IGI Global).

Chapter 2.14
A Social Ontology for Integrating Security and Software Engineering

E. Yu
University of Toronto, Canada

L. Liu
Tsinghua University, China

J. Mylopoulos
University of Toronto, Canada

ABSTRACT

As software becomes more and more entrenched in everyday life in today's society, security looms large as an unsolved problem. Despite advances in security mechanisms and technologies, most software systems in the world remain precarious and vulnerable. There is now widespread recognition that security cannot be achieved by technology alone. All software systems are ultimately embedded in some human social environment. The effectiveness of the system depends very much on the forces in that environment. Yet there are few systematic techniques for treating the social context of security together with technical system design in an integral way. In this chapter, we argue that a social ontology at the core of a requirements engineering process can be the basis for integrating security into a requirements driven software engineering process. We describe the ***i**** agent-oriented modelling framework and show how it can be used to model and reason about security concerns and responses. A smart card example is used to illustrate. Future directions for a social paradigm for security and software engineering are discussed.

INTRODUCTION

It is now widely acknowledged that security cannot be achieved by technological means alone. As more and more of our everyday activities rely on software, we are increasingly vulnerable to lapses

in security and deliberate attacks. Despite ongoing advances in security mechanisms and technologies, new attack schemes and exploits continue to emerge and proliferate.

Security is ultimately about relationships among social actors — stakeholders, system users, potential attackers — and the software that are instruments of their actions. Nevertheless, there are few systematic methods and techniques for analyzing and designing social relationships as technical systems alternatives are explored.

Currently, most of the research on secure software engineering methods focuses on the technology level. Yet, to be effective, software security must be treated as originating from high-level business goals that are taken seriously by stakeholders and decision makers making strategic choices about the direction of an organisation. Security interacts with other high-level business goals such as quality of service, costs, time-to-market, evolvability and responsiveness, reputation and competitiveness, and the viability of business models. What is needed is a systematic linkage between the analysis of technical systems design alternatives and an understanding of their implications at the organisational, social level. From an analysis of the goals and relationships among stakeholders, one seeks technical systems solutions that meet stakeholder goals.

In this chapter, we describe the ***i**** agent-oriented modelling framework and how it can be used to treat security as an integral part of software system requirements engineering. The world is viewed as a network of social actors depending on each other for goals to be achieved, tasks to be performed, and resources to be furnished. Each actor reasons strategically about alternate means for achieving goals, often through relationships with other actors. Security is treated as a high-level goal held by (some) stakeholders that need to be addressed from the earliest stages of system conception. Actors make tradeoffs among competing goals such as functionality, cost, time-to-market, quality of service, as well as security.

The framework offers a set of security requirements analysis facilities to help users, administrators, and designers better understand the various threats and vulnerabilities they face, the countermeasures they can take, and how these can be combined to achieve the desired security results within the broader picture of system design and the business environment. The security analysis process is integrated into the main requirements process, so that security is taken into account from the earliest moment. The technology of smart cards and the environment surrounding its usage provides a good example to illustrate the social ontology of ***i****.

In the next section, we review the current challenges in achieving security in software systems, motivating the need for a social ontology. Given that a social modelling and analysis approach is needed, what characteristics should it have? We consider this in the following section. The two subsequent sections describe the ontology of the ***i**** strategic actors modelling framework and outline a process for analyzing the security issues surrounding a smart card application. The last section reviews several areas of related work and discusses how a social ontology framework can be complementary to these approaches.

BACKGROUND

Despite ongoing advances in security technologies and software quality, new vulnerabilities continue to emerge. It is clear that there can be no perfect security. Security inevitability involves tradeoffs (Schneier, 2003). In practice, therefore, all one can hope for is “good enough” security (Sandhu, 2003).

But how does one determine what is good enough? Who decides what is good enough? These questions suggest that software and information security cannot be addressed by technical specialists alone. Decisions about security are made ultimately by stakeholders — people who

are affected by the outcomes — users, investors, the general public, etc. — because the tradeoffs are about how their lives would be affected. In electronic commerce, consumers decide whether to purchase from a vendor based on the trustworthiness of the vendor's business and security practices. Businesses decide how much and where to invest on security to reduce exposure to a tolerable level. In healthcare, computerized information management can streamline many processes. But e-health will become a reality only if patients and the general public are satisfied that their medical records are protected and secure. Healthcare providers will participate only if liability concerns can be adequately addressed.

Tradeoffs are being made by participants regarding competing interests and priorities. Customers and businesses make judgments about what is adequate security for each type of business, in relation to the benefits derived from online transactions. Patients want their personal and medical information to be kept private, but do not want privacy mechanisms to interfere with the quality of care. In national defense, secrecy is paramount, but can also lead to communication breakdown. In each case, security needs to be interpreted within the context of the social setting, by each stakeholder from his/her viewpoint.

Current approaches to security do not allow these kinds of tradeoffs to be conveyed to system developers to guide design. For example, UML extensions for addressing security (see Chapter I for a review) do not lend themselves well to the modelling of social actors and their concerns about alternate security arrangements, and how they reason about tradeoffs. Access control models can specify policies, but cannot support reasoning about which policies are good for whom and what alternate policies might be more workable. They cannot explain why certain policies meet with resistance and non-compliance.

Each of the common approaches in security modelling and analysis focuses on selective aspects of security, which are important in their own right, but cannot provide the guidance needed to achieve "good enough" overall security. Most approaches focus on technical aspects, neglecting the social context, which is crucial for achieving effective security in practice. The technical focus is well served by mechanistic ontology (i.e., concepts that are suitable for describing and reasoning about automated machinery — objects, operations, state transitions, etc.). The importance of social context in security suggests that a different set of concepts is needed. From the previous discussion, we propose that the following questions are important for guiding system development in the face of security challenges:

- Who are the players who have an interest in the intended system and its surrounding context? Who would be affected by a change?
- What are their strategic interests? What are their business and personal objectives? What do they want from the system and the other players?
- What are the different ways in which they can achieve what they want?
- How do their interests complement or interfere with each other? How can players achieve what they want despite competing or conflicting interests?
- What opportunities exist for one player to advance its interests at the expense of others? What vulnerabilities exist in the way that each actor envisions achieving its objectives?
- How can one player avoid or prevent its interests from being compromised by others?

These are the kind of questions that can directly engage stakeholders, helping them uncover issues and concerns. Stakeholders need the help of technical specialists to think through these questions, because most strategic objectives are accomplished through technological systems.

Stakeholders typically do not know enough about technology possibilities or their implications. Technologists do not know enough about stakeholder interests to make choices for them. In order that stakeholder interests can be clarified, deliberated upon, and conveyed effectively to system developers, a suitable modelling method is needed to enable stakeholders and technologists to jointly explore these questions. The answers to these questions will have direct impact on system development, as they set requirements and guide technical design decisions.

We argue therefore that a social ontology is needed to enable security concerns to become a driving force in software system development. In the next section, we explore the requirements for such a social ontology.

APPROACH

If a treatment of security requires attention to the social context of software systems, can the social analysis be given full weight in a software engineering methodology that is typically dominated by a mechanistic worldview? How can the social modelling be reconciled and integrated with mainstream software modelling?

It turns out that a social paradigm for software system analysis is motivated not only by security concerns, but is consistent with a general shift in the context of software and information systems. The analysis of computers and information systems used to be machine-centric when hardware was the precious resource. The machine was at the centre, defining the human procedures and structures needed to support its proper functioning. Today, hardware and software are commoditized and distributed everywhere. Human practices and imagination determine how hardware and software are put to use, not the other way round. Pervasive networking, wired and wireless, has also contributed to blurring the notion of "system." Computational resources can be dynamically harnessed in ad hoc configurations (e.g., through Web services protocols in service-oriented architectures) to provide end-to-end services for a few moments, then dissolved and reconfigured for another ad hoc engagement. Even computational entities, in today's networked environment, are better viewed as participants in social networks than as fixed components in a system with predefined structure and boundary. Increasingly, the computational services that we desire will not be offered as a single pre-constructed system, but by a conglomeration of interacting services operated by different organisations, possibly drawing on content owned by yet other providers.

The questions raised in the previous section arise naturally from today's open networked environments, even if one were not focusing on security concerns. The relevance of a social ontology is therefore not unique to security. Competing interests and negative forces that interfere with one's objectives are ever present in every organisation and social setting. They are accentuated in an open network environment. In security scenarios, the negative forces are further accentuated as they materialize into full-fledged social structures, involving malicious actors collaborating with other actors, engaging in deliberate attacks, possibly violating conventions, rules, and laws. Security can therefore be seen as covering the more severe forms of a general phenomenon. Competing and conflicting interests are inherent in social worlds. Negative forces do not come only from well identified malicious external agents, but can be present legitimately within one's organisation, among one's associates, and even among the multiple roles that one person may play. It may not be possible to clearly separate security analysis from the analysis of "normal" business. We conclude, therefore, that a social ontology would serve well for "normal" business analysis, recognizing the increasingly "social" nature of software systems and their environments. A social ontology offers a smooth integration of the treatment of normal and security scenarios, as the latter merely refer

to one end of a continuum covering positive and negative forces from various actors.

Given this understanding, the social ontology should not be preoccupied with those concepts conventionally associated with security. For example, the concepts of asset, threat, attack, counter-measure are key concepts for security management. In the social ontology we aim to construct, we do not necessarily adopt these as primitive concepts. Instead, the social ontology should aim to be as general as possible, so that the concepts may be equally applicable to positive as well as negative scenarios. The general ontology is then *applied* to security. Special constructs unique to security would be introduced only if the expressiveness of the general constructs is found to be inadequate. The principle of Occam's razor should be applied to minimize the complexity of the ontology. If desired, shorthand notations for common recurring patterns can be defined in terms of the primitives. The premises behind a social ontology are further discussed in Yu (2001a, 2001b).

BASIC CONCEPTS OF THE *i** STRATEGIC MODELLING FRAMEWORK

The ***i**** framework (Yu, 1993, 1997) proposes an agent oriented approach to requirements engineering centering on the intentional characteristics of the agent. Agents attribute intentional properties such as goals, beliefs, abilities, commitments to each other and reason about strategic relationships. Dependencies give rise to opportunities as well as vulnerabilities. Networks of dependencies are analyzed using a qualitative reasoning approach. Agents consider alternative configurations of dependencies to assess their strategic positioning in a social context. The name ***i**** (pronounced eye-star) refers to the concept of multiple, distributed "intentionality."

The framework is used in contexts in which there are multiple parties (or autonomous units) with strategic interests, which may be reinforcing or conflicting in relation to each other. The ***i**** framework has been applied to business process modelling (Yu, 1993), business redesign (van der Raadt, Gordijn, & Yu, 2005; Yu et al., 2001), requirements engineering (Yu, 1997), architecture modelling (Gross & Yu, 2001), COTS selection (Franch & Maiden, 2003), as well as to information systems security.

There are three main categories of concepts: actors, intentional elements, and intentional links. The framework includes a strategic dependency (SD) model — for describing the network of relationships among actors, and a strategic rationale (SR) model — for describing and supporting the reasoning that each actor has about its relationships with other actors.

Actor

In ***i****, an *actor* () is used to refer generically to any unit to which intentional dependencies can be ascribed. An actor is an active entity that carries out actions to achieve its goals by exercising means-ends knowledge. It is an encapsulation of intentionally, rationality and autonomy. Graphically, an actor is represented as a circle, and may optionally have a dotted boundary, with intentional elements inside.

Intentional Elements: Goal, Softgoal, Task, Resource and Belief

The intentional elements in ***i**** are goal, task, softgoal, resource and belief. A goal () is a condition or state of affairs in the world that the stakeholders would like to achieve. A goal can be achieved in different ways, prompting alternatives to be considered. A goal can be a business goal or a system goal. Business goals are about the business or state of the affairs the

individual or organisation wishes to achieve in the world. System goals are about what the target system should achieve, which, generally, describe the functional requirements of the target system. In the ***i**** graphical representation, goals are represented as a rounded rectangle with the goal name inside.

A *softgoal* () is typically a quality (or non-functional) attribute on one of the other intentional elements. A softgoal is similar to a (hard) goal except that the criteria for whether a softgoal is achieved are not clear-cut and *a priori*. It is up to the developer to judge whether a particular state of affairs in fact sufficiently achieves the stated softgoal. Non-functional requirements, such as performance, security, accuracy, reusability, interoperability, time to market and cost are often crucial for the success of a system. In ***i****, non-functional requirements are represented as softgoals and addressed as early as possible in the software lifecycle. They should be properly modelled and addressed in design reasoning before a commitment is made to a specific design choice. In the ***i**** graphical representation, a softgoal is shown as an irregular curvilinear shape.

*Task*s () are used to represent the specific procedures to be performed by agents, which specifies a particular way of doing something. It may be decomposed into a combination of subgoals, subtasks, resources, and softgoals. These sub-components specify a particular course of action while still allowing some freedom. Tasks are used to incrementally specify and refine solutions in the target system. They are used to achieve goals or to "operationalize" softgoals. These solutions provide operations, processes, data representations, structuring, constraints, and agents in the target system to meet the needs stated in the goals and softgoals. Tasks are represented graphically as a hexagon.

A *resource* () is a physical or informational entity, which may serve some purpose. From the viewpoint of intentional analysis, the main concern with a resource is whether it is available. Resources are shown graphically as rectangles.

The *belief* () construct is used to represent domain characteristics, design assumptions and relevant environmental conditions. It allows domain characteristics to be considered and properly reflected in the decision making process, hence facilitating later review, justification, and change of the system, as well as enhancing traceability. Beliefs are shown as ellipses in ***i**** graphical notation.

Strategic Dependency Model

A strategic dependency (SD) model consists of a set of nodes and links. Each node represents an actor, and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. We call the depending actor the *depender*, and the actor who is depended upon the *dependee*. The object around which the dependency relationship centers is called the *dependum*. By depending on another actor for a dependum, an actor (the depender) is able to achieve goals that it was not able to without the dependency, or not as easily or as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals. A *dependency* link () is used to describe such an inter-actor relationship. Dependency types are used to differentiate the kinds of freedom allowed in a relationship.

In a *goal dependency*, an actor depends on another to make a condition in the world come true. Because only an end state or outcome is specified, the dependee is given the freedom to choose how to achieve it.

In a *task dependency*, an actor depends on another to perform an activity. The depender's goal for having the activity performed is not given. The activity description specifies a particular course

of action. A task dependency specifies standard procedures, indicates the steps to be taken by the dependee.

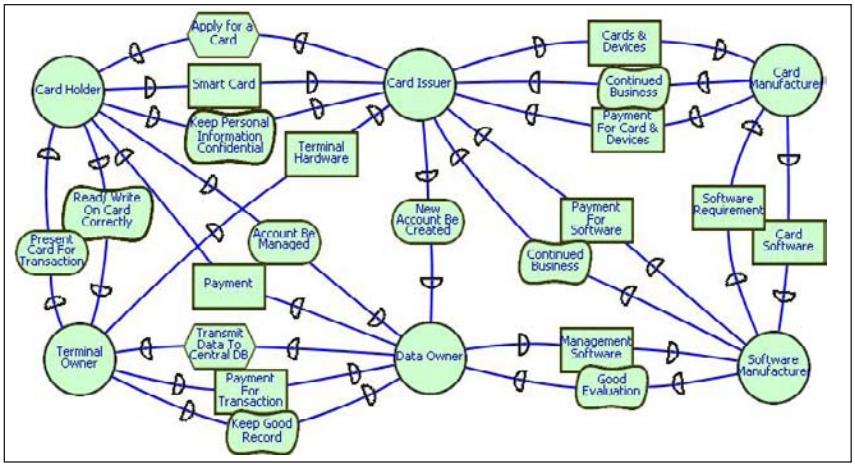
In a *resource dependency*, an actor depends on another for the availability of an entity. The depender takes the availability of the resource to be unproblematic.

The fourth type of dependency, *softgoal dependency*, is a variant of the goal dependency. It is different in that there are no *a priori*, cut-and-dry criteria for what constitutes meeting the goal. The meaning of a softgoal is elaborated in terms of the methods that are chosen in the course of pursuing the goal. The dependee contributes to the identification of alternatives, but the decision is taken by the depender. The notion of the softgoal allows the model to deal with many of the usually informal concepts. For example, a service provider's dependency on his customer for continued business can be achieved in different ways. The desired style of continued business is ultimately decided by the depender. The customer's softgoal dependency on the service provider for "keep personal information confidential" indicates that there is not a clear-cut criterion for the achievement of confidentiality. The four types of dependencies reflect different levels of freedom that is allowed in the relationship between depender and dependee.

Figure 1 shows a SD model for a generic smart card-based payment system involving six actors. This example is adapted from Yu and Liu (2001). A Card Holder depends on a Card Issuer to be allocated a smart card. The Terminal Owner depends on Card Holder to present the card for each transaction. The Card Issuer in turn depends on the Card Manufacturer and Software Manufacturer to provide cards, devices, and software. The Data Owner is the one who has control of the data within the card. He depends on the Terminal Owner to submit transaction information to the central database. In each case, the dependency means that the depender actor depends on the dependee actor for something in order to achieve some (internal) goal.

The goal dependency New Account Be Created from the Card Issuer to the Data Owner means that it is up to the Data Owner to decide how to create a new account. The Card Issuer does not care how a new account is created; what matters is that, for each card, an account should be created. The Card Issuer depends on the Card Holder to apply for a card via a task dependency by specifying standard application procedures.

Figure 1. Strategic dependency model of a generic smart card system



If the Card Issuer were to indicate the steps for the Data Owner to create a new account, then the Data Owner would be related to the Card Issuer by a task dependency instead.

The Card Issuer's dependencies on the Card Manufacturer for cards and devices, the manufacturer's dependencies on Card Issuer for payment are modelled as resource dependencies. Here the depender takes the availability of the resource to be unproblematic.

The Card Holder's softgoal dependency on the Card Issuer for Keep Personal Information Confidential indicates that there is not a clear-cut criterion for the achievement of confidentiality. In the Manufacturer's softgoal dependency on Card Issuer, Continued Business could be achieved in different ways. The desired style of continued business is ultimately decided by the depender.

The strategic dependency model of Figure 1 is not meant to be a complete and accurate description of any particular smart card system. It is intended only for illustrating the modelling features of ***i****.

In conventional software systems modelling, the focus is on information flows and exchanges — what messages actors or system components send to each other. With the social ontology of ***i****, the focus is on intentional relationships — what are the actors' expectations and constraints on each other. Since actors are intentional, strategic, and have autonomy, they reflect on their relationships with other actors. If these relationships are unsatisfactory, they will seek alternative ways of associating with others.

Security concerns arise naturally from this perspective. A social ontology therefore provides a way to integrate security into software system engineering from the earliest stages of conception, and at a high level of abstraction.

Intentional Links

Dependencies are intentional relationships between actors. Within each actor, we model intentional relationships in terms of means-ends, decomposition, contribution, and correlation links.

- **Means-ends** links (⊣▷⊢) are used to describe how goals can be achieved. Each task connected to a goal by a means-ends link is one possible way of achieving the goal.
- **Decomposition** links (—┼) define the sub-elements of a task, which can include sub-tasks, sub-goals, resources, and softgoals. The softgoals indicate the desired qualities that are considered to be part of the task. The sub-tasks may in turn have decomposition links that lead to further sub-elements. Sub-goals indicate the possibility of alternate means of achievement, with means-ends links leading to tasks.
- A **contribution** link (→) describes the qualitative impact that one element has on another. A contribution can be negative or positive. The extent of contribution is judged to be partial or sufficient based on Simon's concept of satisficing (Simon, 1996), as in the NFR framework (Chung, Nixon, Yu, & Mylopoulos, 2000). Accordingly, contribution link types include: *help* (positive and partial), *make* (positive and sufficient), *hurt* (negative and partial), *break* (negative and sufficient), *some+* (positive of unknown extent), *some-* (negative of unknown extent). *Correlation* links (dashed arrows) are used to express contributions from one element to other elements that are not explicitly sought, but are side effects.

Strategic Rationale Model

The strategic rationale (SR) model provides a detailed level of modelling by looking "inside" actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in SR models not only as external dependencies, but also as internal ele-

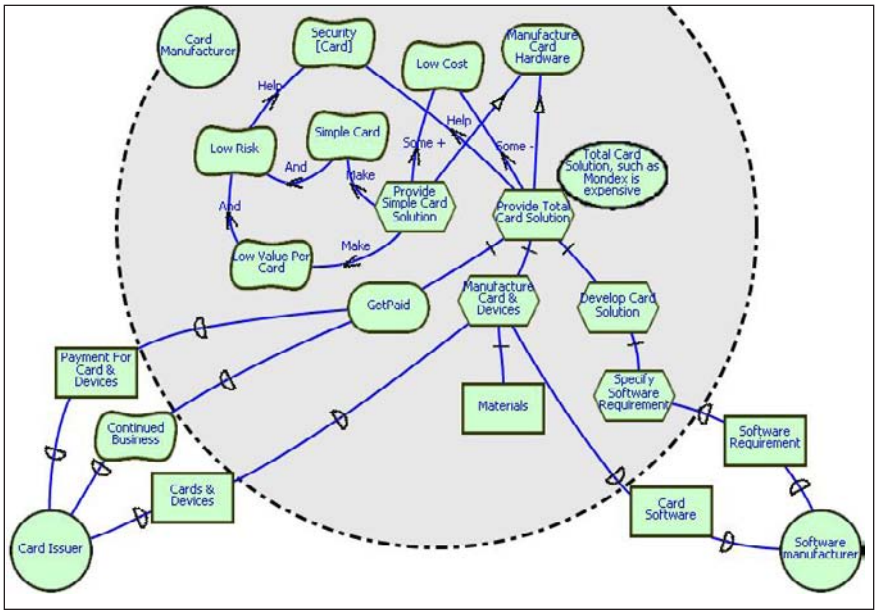
ments arranged into a predominantly hierarchical structure of means-ends, task-decompositions and contribution relationships.

The SR model in Figure 2 elaborates on the rationale of a Card Manufacturer. The Card Manufacturer's business objective Manufacture Card Hardware is modeled as a "hard" functional goal (top right corner). Quality requirements such as Security and Low Cost are represented as softgoals. The different means for accomplishing the goal are modeled as tasks. The task Provide Total Card Solution can be further decomposed into three sub-components (connected with task-decomposition links): sub-goal of Get Paid, sub-task Develop Card Solution, and sub-task Manufacture Card & Devices. To perform the task Manufacture Card & Devices, the availability of Materials need to be taken into consideration, which is modeled as a resource.

In the model, task node Provide Simple Card Solution (such as the Millicent solution), and Provide Total Card Solution (such as the Mondex solution) are connected to the goal with means-ends links. This goal will be satisfied if any of these tasks is satisfied. Provide Total Card Solution will help the Security of the system (represented with a *Help* contribution link to *Security*), while Provide Simple Card Solution is considered to have no significant impact on security if it is applied to cards with small monetary value. The Simple Card Solution is good for the goal of Low Cost whereas the Total Card Solution is bad. This is supported by the belief that "Total Card Solution, such as Mondex, is expensive." Beliefs are usually used to represent such domain properties, or design assumption or environmental condition, so that traceability of evidence of design decision could be explicitly maintained with the model.

During system analysis and design, softgoals such as Low Cost and Security [card] are systematically refined until they can be operationalized and implemented. Unlike functional goals, nonfunctional qualities represented as softgoals frequently interact or interfere with each other, so the graph of contributions is usually not a strict tree structure (Chung et al., 2000).

Figure 2. Strategic rationale model of card manufacturer



Agents, Roles, and Positions

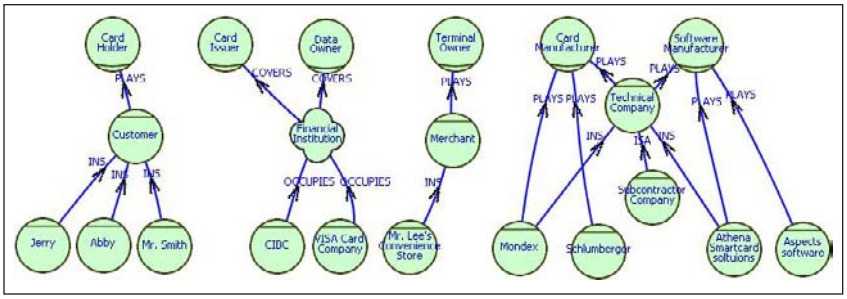
To model complex relationships among social actors, we further define the concepts of agents, roles, and positions, each of which is an actor, but in a more specialized sense.

A *role* ( ) is an abstract actor embodying expectations and responsibilities. It is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. An *agent* ( ) is a concrete actor with physical manifestations, human or machine, with specific capabilities and functionalities. A set of roles packaged together to be assigned to an agent is called a position. A *position* ( ) is intermediate in abstraction between a role and an agent, which often has an organisational flavor. Positions can COVER roles. Agents can OCCUPY positions. An agent can PLAY one or more roles directly. The INS construct is used to represent the instance-and-class relation. The ISA construct is used to express conceptual generalization/specialization. Initially, human actors representing stakeholders in the domain are identified together with existing machine actors. As the analysis proceeds, more actors are identified, including new system agents, when certain design choices have been made, and new functional entities are added.

Figure 3 shows some actors in the domain. At the top, six generic abstract roles are identified, including the Card Holder, the Terminal Owner, the Data Owner, the Card Issuer, the Card Manufacturer, and the Software Manufacturer. These actors are modeled as roles since they represent abstractions of responsibilities and functional units of the business model. Then concrete agents in smart card systems are identified. For instance, actors in a Digital Stored Value Card system include Customer, Merchant, Subcontractor Company, and their instances. These agents can play one or more roles in different smart card systems. Here, Financial Institution is modeled as a position that bridges the multiple abstract roles it covers, and the real world agents occupying it. Initially, human/organisational actors are identified together with existing machine actors. As the requirements analysis proceeds, more actors could be added in, including new system agents such as security monitoring system, counter-forgery system, etc., when certain design choices have been made, and new functional entities are added.

An *agent* is an actor with concrete, physical manifestations, such as a human individual. An agent has dependencies that apply regardless of what role he/she/it happens to be playing. For example, in Figure 3, if Jerry, a Card Holder desires a good credit record, he wants the credit record to go towards his personal self, not to the positions and abstract roles that Jerry might occupy or play. We use the term agent instead of person for generality, so that it can be used to refer to human as well as artificial (hardware, software,

Figure 3. Actor hierarchy (roles, positions, and agents) in a smart card system



or organisational) agents. Customer and Merchant are represented as agent classes and groups. Dependencies are associated with a role when these dependencies apply regardless of who plays the role. For example, we consider Card Holder an abstract role that agents can play. The objective of obtaining possession of the card, and deciding when and whether to use it, are associated with the role, no matter who plays the role.

The INS construct represents the instance-and-class relation. For example, Mr. Lee's Convenience Store is an instance of Merchant, and Jerry is an instance of Customer. The ISA construct expresses conceptual generalization/ specialization. For example, a Subcontractor Company is a kind of Technical Company. These constructs are used to simplify the presentation of strategic models with roles, positions, and agents. There can be dependencies from an agent to the role it plays. For example, a Merchant who plays the role of Terminal owner may depend on that role to attract more customers. Otherwise, he may choose not to play that role.

Roles, positions, and agents can each have subparts. In general, aggregate actors are not compositional with respect to intentional properties. Each actor, regardless of whether it has parts, or is part of a larger whole, is taken to be intentional. Each actor has inherent freedom and is therefore ultimately unpredictable. There can be intentional dependencies between the whole and its parts (e.g., a dependency by the whole on its parts to maintain unity).

DOMAIN REQUIREMENTS ANALYSIS WITH *i**

We now illustrate how the social ontology of ***i**** allows security issues to be identified and addressed early in the requirements process. We continue with the example of smart card systems design. Security in smart card systems is a challenging task due to the fact that different aspects of the system are not under a single trust boundary. Responsibilities are split among multiple parties. The processor, I/O, data, programs, and network may be controlled by different, and potentially hostile, parties. By discussing the security ramifications of different ways of splitting responsibilities, we aim to show how the proposed modelling framework can help produce a proper understanding of the security systems that employ smart cards. Figure 4 shows the basic steps to take during the process of domain requirements analysis with ***i****, before we consider security. The process can be organised into the following iterative steps.

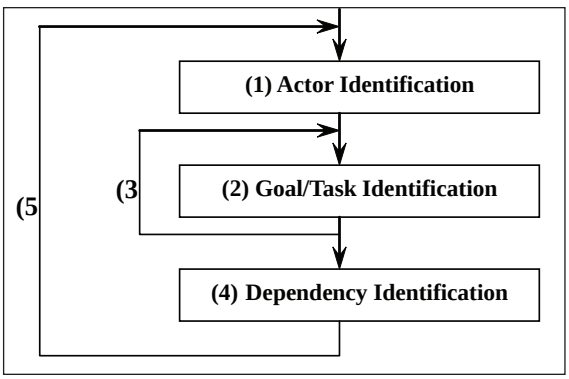
Actor Identification

In step (1), the question "who is involved in the system?" will be answered. According to the definition given above, we know that all intentional units may be represented as actors. For example, in any smart card based systems, there are many parties involved. An actor hierarchy composed of roles, positions, and agents such as the ones in Figure 3 is created.

Goal/Task Identification

In the step (2) of the requirements analysis process, the question "what does the actor want to achieve?"

*Figure 4. Requirements elicitation process with i**



will be answered. As shown in the strategic rationale (SR) model of Figure 2, answers to this question can be represented as goals capturing the high-level objectives of agents. During system analysis and design, softgoals such as low cost and security are systematically refined until they can be operationalized and implemented. Using the SR model, we can reason about each alternative's contributions to high-level non-functional quality requirements including security, and possible tradeoffs.

The refinements of goals, tasks and softgoals (step (3) in Figure 4) are considered to have reached an adequate level once all the necessary design decisions can be made based on the existing information in the model. The SR model in Figure 3 was created by running through steps (1), (2), (3) in Figure 4 iteratively.

Strategic Dependency Identification

In the step (4) of the requirements analysis process, the question "how do the actors relate to each other?" will be answered. Figure 1 shows the SD model for a generic smart card-based payment system. By analyzing the dependency network in a Strategic Dependency model, we can reason about opportunities and vulnerabilities. A Strategic Dependency model can be obtained by hiding the internal rationales of actors in a Strategic Rationale model. Thus, the goal, task, resource, softgoal dependencies in a Strategic Dependency model can be seen as originating from SR models.

The kinds of analysis shown above answers questions such as "who is involved in the system? What do they want? How can their expectations be fulfilled? And what are the inter-dependencies between them?" These answers initially provide a sketch of the social setting of the future system, and eventually result in a fairly elaborate behavioral model where certain design choices have already been made. However, another set of very important questions has yet to be answered (i.e., what if things go wrong)? What if some party involved in the smart card system does not behave as expected? How bad can things get? What prevention tactics can be considered?" These are exactly the questions we want to answer in the security requirements analysis.

SECURITY REQUIREMENTS ANALYSIS WITH *i**

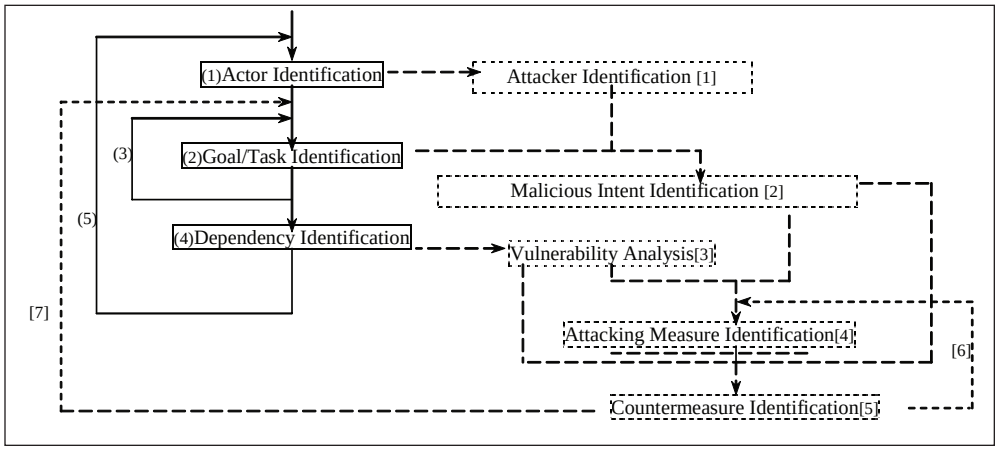
We now extend the process to include attacker analysis, vulnerability analysis, and countermeasure analysis. The dashed lines and boxes on the right hand side of Figure 5 indicate a series of analysis steps to deal with security. These steps are integrated into the basic domain requirements engineering process, such that threats from potential attackers are anticipated and countermeasures for system protection are sought and equipped wherever necessary. Each of the security related analysis steps (step [1] to [7]) will be discussed in detail in the following subsections.

Attacker Analysis

The attacker analysis steps aim to identify potential system abusers and their malicious intents. The basic premise here is that all the actors are assumed "guilty until proven innocent." In other words, given the result of the basic ***i**** requirements modelling process, we now consider any one of the actors (roles, positions, or agents) identified so far can be a potential attacker to the system or to other actors. For example, we want to ask, "In what ways can a terminal owner attack the system? How will he benefit from inappropriate manipulation of the card reader, or transaction data?"

In this analysis, each actor is considered in turn as an attacker. This attacker inherits the intentions, capabilities, and social relationships of the corresponding legitimate actor (i.e., the internal goal hierarchy and external dependency relationships

*Figure 5. Security requirements elicitation process with **i****



in the model). This may serve as a starting point of a forward direction security analysis (step [1] in Figure 5). A backward analysis starting from identifying possible malicious intents and valuable business assets can also be done.

Proceeding to step [2] of the process, for each attacker identified, we combine the capabilities and interests of the attacker with those of the legitimate actor. For simplicity, we assume that an attacker may be modeled as a role or an agent. To perform the attacker analysis, we consider that each role may be played by an attacker agent, each position may be occupied by an attacker agent, and that each agent may play an attacker role (Figure 6). The analysis would then reveal the commandeering of legitimate resources and capabilities for illicit use. The intents and strategies of the attackers are explicitly represented and reasoned about in the models.

This approach treats all attackers as insider attackers, as attacks are via associations with normal actors. We set a system boundary, then exhaustively search for possible attackers. Random attackers such as Internet hackers/crackers, or attackers breaking into a building can also be dealt with by being represented as sharing the same territory with their victim. By conducting analysis on the infrastructure of the Internet, we may identify attackers by treating Internet resources as resources in the ***i**** model. By conducting building security analysis, break-in attackers, or attackers sharing the same workspace can be identified. Alternatively, we could adopt an opposite assumption, i.e., assume there is a trusted perimeter for each agent, all the potential threat sources within this trusted perimeter are ignored, measures will only be taken to deal with threats from outside of the perimeter.

As shown in the Strategic Rationale model in Figure 7, the motives of Attacker in the smart card

Figure 6. Modelling attackers in strategic actors model

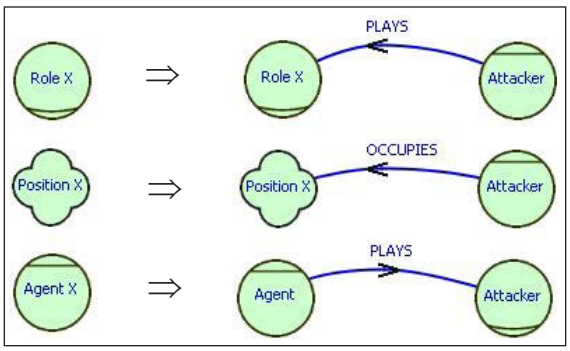
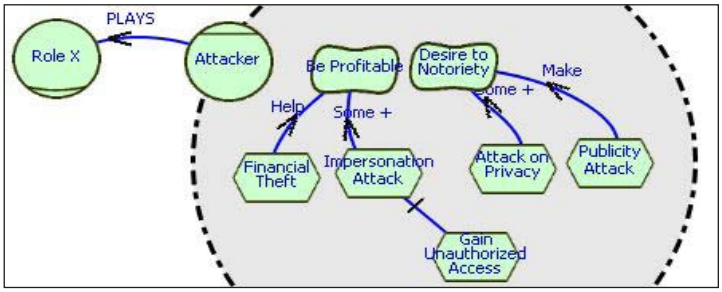


Figure 7. Motives of attacker in a smart card system



system may be modeled as intentional elements in an i* model. An attacker may be motivated by financial incentives (softgoal Be Profitable), or by non-financial ones (e.g., Desire for Notoriety). These malicious intents may lead to various attack strategies, such as Financial Theft, Impersonation Attack, Gain Unauthorized Access, Attack on Privacy, and Publicity Attack.

Dependency Vulnerability Analysis

Dependency vulnerability analysis aims at identifying the vulnerable points in the dependency network (step [3] in Figure 5). A dependency relationship makes the depender inherently vulnerable. Potential attackers may exploit these vulnerabilities to actually attack the system, so that their malicious intents can be served. ***i**** dependency modelling allows a more specific vulnerability analysis because the potential failure of each dependency can be traced to a depender and to *its* dependers. The questions we want to answer here are "which dependency relationships are vulnerable to attack?", "What are the chain effects if one dependency link is compromised?" The analysis of dependency vulnerabilities does not end with the identification of potential vulnerable points. We need to trace upstream in the dependency network, and see whether the attacked dependency relationship impacts other actors in the network.

Figure 8 is a simplified version of the SD model of Figure 4, showing only the softgoal dependencies. We assume that each of the actors in the SD model can be a potential attacker. And as an attacker, an actor will fail to deliver the expected dependencies directed to it, of whom it is the dependee.

For instance, the Card Holder depends on the Terminal Owner to Read/Write Card Correctly. To analyze the vulnerability arising from this dependency, we consider the case where the terminal owner is not trustworthy. And we try to identify the potential attacks by answering question of "In what possible ways could the attacker break this dependency relationship?" To do this, we elaborate on the agent Attacker Playing Terminal Owner. Starting from attacker's potential motivations, we refine the high-level goals of the attackers (and possible attack routes) based on analysis of the SD and SR models of the normal operations of the smart card (e.g., what resources an actor accesses, what types of interactions exist, etc.). In this way, we may identify a number of potential attacks that are sufficient to make this dependency not viable (*Break*).

Proceeding to step [4], we now focus on how an attacker may attack the vulnerable points identified above by exploring the attacker's capacities. We model potential attacks (including fraud) as negative contributions from the attackers (from their specific methods of attack) toward the de-

Figure 8. Dependencies (in other words, vulnerable points) in a smart card system

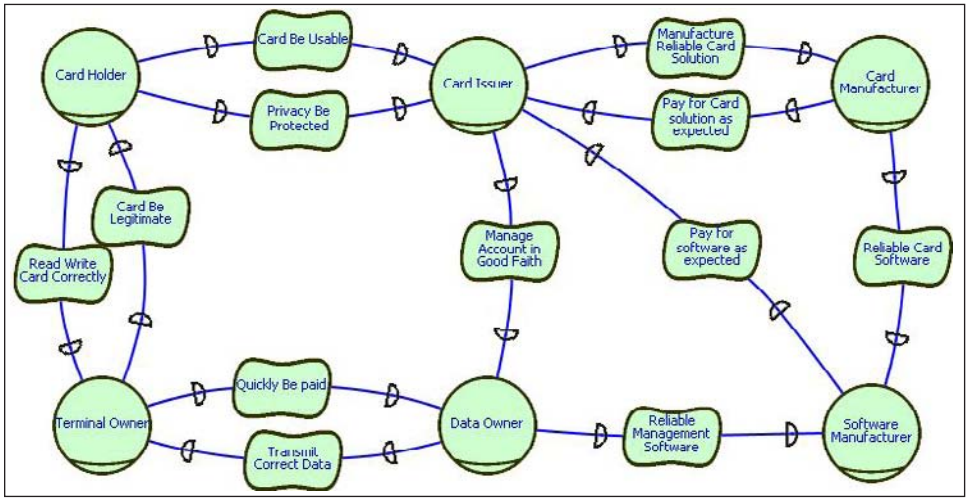
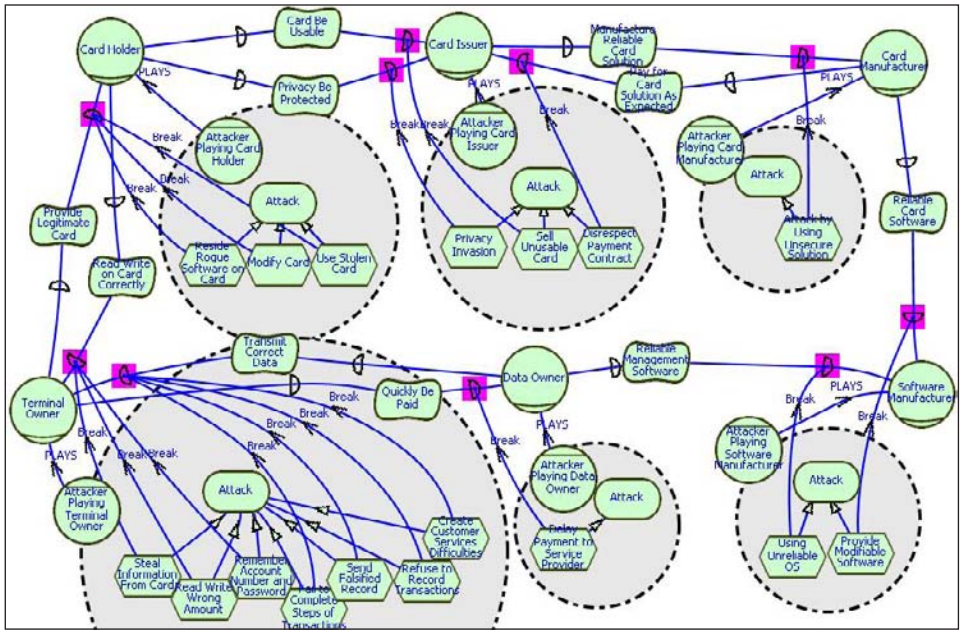


Figure 9. Attacks directed to vulnerable dependencies in a smart card system



pendee-side dependency link. A *Break* contribution indicates that the attack is sufficient to make the softgoal unviable. For clarity of analysis, we place the attack-related intentional elements into agents called "Attacker Playing Role X." Details of the attack methods (e.g., Steal Card Information, Send Falsified Records) can be elaborated by further means-ends and decomposition analysis. Thus, the steps and methods of the attack can be modeled and analyzed. Other internal details of the Terminal Owner are not relevant and are thus not included in the model. Negative contribution

links are used to show attacks on more specific vulnerabilities of the depender (e.g., refinements of Transact with Card).

The dependencies that could be broken are highlighted with a small square in Figure 9. When a dependency is compromised, the effect could propagate through the dependency network upstream along the dependency links. For example, if the Terminal Owner is not Quickly Be Paid, he may stop accepting card as a payment option.

Countermeasure Analysis

During countermeasure analysis, system designers make decisions on how to mitigate vulnerabilities and set up defenses against potential attackers. This type of analysis covers general types of attacks, and formulates solutions by selectively applying, combining, or instantiating prototypical solutions to address the specific needs of various stakeholders. The general types of attacks and the prototypical solutions can be retrieved from a taxonomy or knowledge repository.

Necessary factors for the success of an attack are attacker's motivations, vulnerabilities of the system, and attacker's capabilities to carry out the attack. Thus, to counteract a hypothetical attack, we seek measures that will sufficiently negate these factors. Based on the above analysis, we already understand the attackers' possible malicious intents and system vulnerabilities. As shown in Figure 5, countermeasure analysis is an iterative process. Adding protective measures may bring new vulnerabilities to the system, so a new round of vulnerability analysis and countermeasure analysis will be triggered (step [6]).

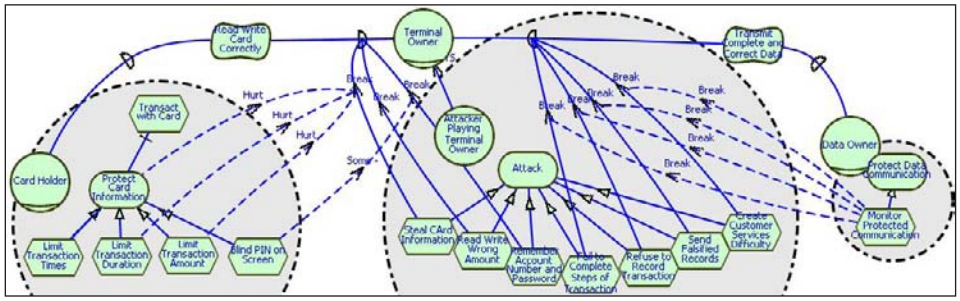
With the knowledge of some potential attacks and frauds, the depender may first look for trustworthy partners, or change their methods of operation, or add control mechanisms (countermeasures) to protect their interests. A countermeasure may prevent the attack from happening by either making it technically impossible, or by eliminating the attacker's intent of attack.

Figure 10 shows a SR model with defensive actions as well as attacks. Protection mechanisms are adopted to counteract specific attacks. In some cases, the protections are sufficient to defeat a strong attack (defense *Break* link (dotted arrow) pointing to an attack *Break* link). In other cases, countermeasures are only partially effective in defending against their respective attacks (through the *Hurt* or *Some*- contribution types).

Qualitative Goal-Reasoning Mechanism

A qualitative goal-reasoning process is used to propagates a series of labels through the models. A label (or satisficing status) on a node is used to indicate whether that intentional element (goal, task, resource, or softgoal) is viable or not (e.g.,

Figure 10. Resistance models defeating hypothetical attacks



whether a softgoal is sufficiently met). Labels can have values such as Satisfied "✓," Denied "✗," Weakly Satisfied "✓ " and Weakly Denied "✗," Undecided "?," etc. (Liu et al., 2003). Leaf nodes (those with no incoming contributions) are given labels by the analyst based on judgment of their independent viability. These values are then propagated "upwards" through the contribution network (following the direction of the contribution links, and from dependee to depender). The viability of the overall system appears in the high level nodes of the various stakeholders. The process is an interactive one, requiring the analyst to make judgments whenever the outcome is inconclusive given the combination of potentially conflicting contributions.

To begin, the analyst labels all the attack leaf nodes as Satisficed since they are all judged to be possible (Figure 11). Similarly, all the defense leaf nodes are judged to be viable, thus labelled Satisfied. The values are then propagated along contribution links. Before adding defense nodes, the Card Holder's dependency on the Terminal Owner for Read Write Card Correctly softgoal was labelled as Denied, because of the potentially strong attacks from Terminal Owner. However, as countermeasures are added, the influences of the attacks will be correspondingly weakened.

Regarding Read Write Card Correctly, three possible attacks are identified. One of them Steal Card Info is counteracted by three defense measures, though each one is partial (Hurt). Another attack Remember Account Number & Password has a defense of unknown strength (*Some-*). The third attack has no defensive measure. The softgoal dependency Read Write Card Correctly is thus judged to be weakly unviable (✗). On the other side, as the Data Owner's protection mechanism could sufficiently defeat the four possible attacks, the Transmit Complete and Correct Data softgoal dependency is thus judged to be viable (✓). Potential attacks lead to the erosion of viability of the smart card system. Incorporating sufficient countermeasures restores viability.

A prototype knowledge-based tool is being constructed to support this framework for analyzing information systems security.

Trust Analysis Based on System Configuration

In the models previously given, the various participants in a smart card system were modelled as abstract roles and analyzed generally. However, in real world smart card systems, various concrete physical or organisational parties play or occupy these roles. These are shown in Table 1. Thus, to actually understand their trust and security situations, we have to apply the generic model to the real world configurations. We consider

Figure 11. Countermeasure effectiveness evaluation model

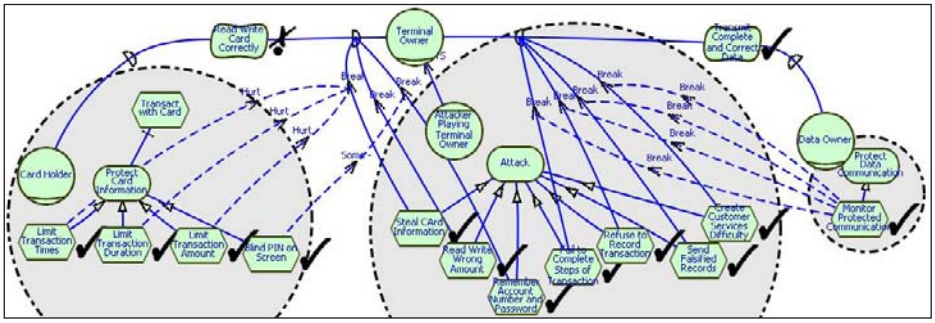


Table 1. Actors (roles, positions, and agents) in various smart card system configurations

Generic Smart Card Model	Card Holder	Terminal Owner	Card Issuer	Data Owner	Card Manufacturer	Software Manufacturer
Digital Stored Value card	Customer	Merchant	Financial Institution		Technology Company	
Digital Check Card	Customer	Merchant	Financial Institution	Customer	Technology Company	
Prepaid Phone Card	Customer	Phone Company				
Account-based Phone Card	Customer	Phone Company		Customer	Technology Company	
Key store card	User		Technology Company			
Employee Access Token	Employee	Employer				
Web browsing card	Customer		Financial Institution		Technology Company	

two representative kinds of smart card based systems. One is the Digital Stored Value Card, the other is the Prepaid Phone Card (Schneier & Shostack, 1998).

Digital Stored Value Card System

These are payment cards intended to be substitutes for cash. Both Mondex and VisaCash are examples of this type of system. The Customer is the Card Holder. The Merchant is the Terminal Owner. The Financial Institution that supports the system is both the Data Owner and the Card Issuer. The Smart Card Technology Company, such as Mondex, is both the Card Manufacturer and the Software Manufacturer.

In such a configuration, the previously separated roles of Data Owner and Card Issuer are Played by the same physical agent, namely, Financial Institution. Similarly, Card Manufacturer and Software Manufacturer are combined into one physical agent — the Smart Card Technology Company. Figure 12 describes the threat model of a digital stored value card. Here the Software Manufacturer's attack on Card Manufacturer can be ignored since they belong to the same agent — the Smart Card Technology Company. Also the attack from Data Owner to Card Issuer can be ignored since they both played by the Financial Institution. These two attacking-defending relationships are highlighted in Figure 11 with little squares.

Prepaid Phone Card System

These are special-use stored value cards. The Customer is the Card Holder. The Phone Company plays all the four roles of Terminal Owner, Data Owner, Manufacturer, and Card Issuer. Figure 13 shows the threat model of a prepaid card system. Under such a system configuration, more attack-defense pairs disappear. Only four possible attacks need to be considered now. Three of them are from the phone company, which includes violat-

Figure 12. A threat model of digital stored value card system

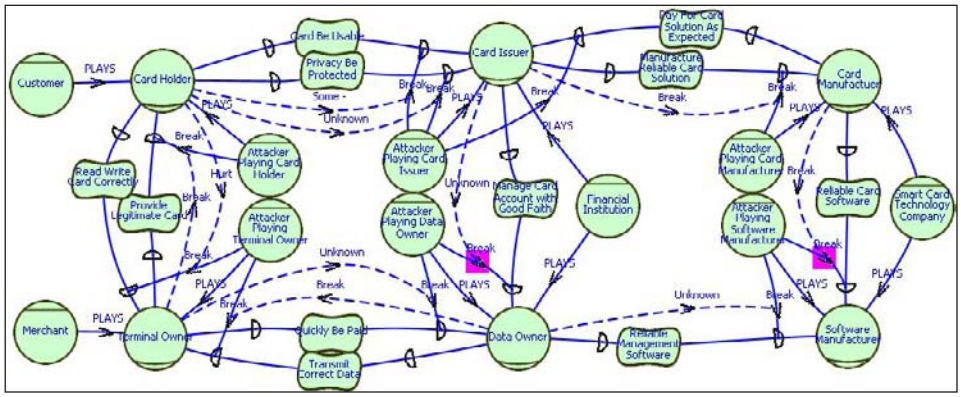
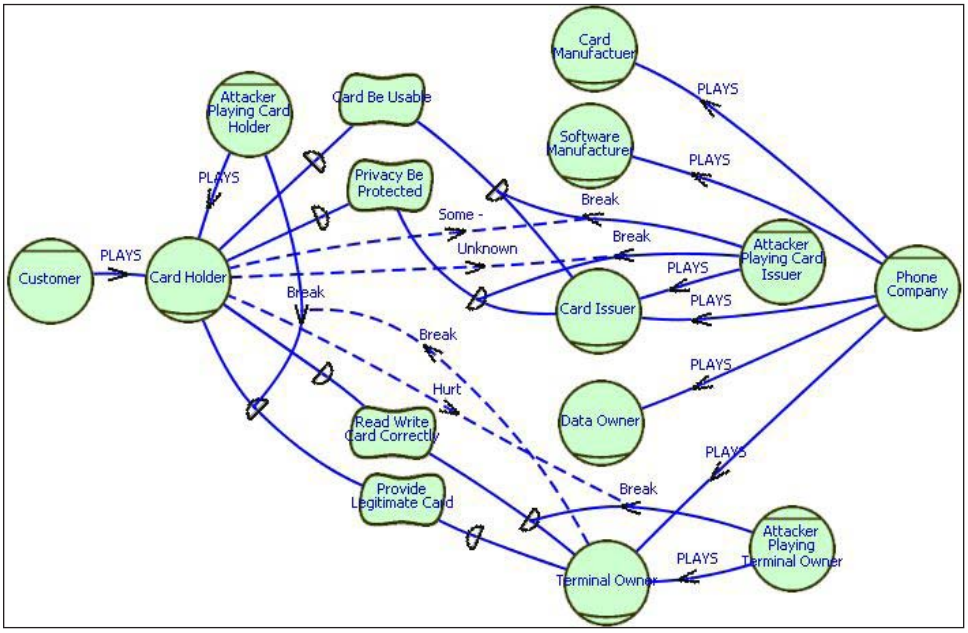


Figure 13. A threat model of prepaid phone card system



ing privacy, to issue unusable card, to read write card incorrectly. The other attack is from the Card Holder, who might use an illegitimate card.

Note that each time new roles are created, the possibility of new attacks arises. These models reflect Schneier's observation that the fewer splits we make, the more trustworthy the target system is likely to be (Schneier & Shostack, 1998).

RELATED WORK

This section is complementary to the review presented in Chapter I. Each approach to security and software engineering has an ontology, whether explicitly defined or implied. We expect that a social ontology can be complementary and beneficial to various approaches to integrating

security and software engineering. We begin with work from the security community, followed by software engineering approaches that have paid special attention to security.

Security Models

Formal models have been an important part of computer security since mainframe computing (Samarati & Vimercati, 2001). Security policies originate from laws, regulations, or organisational practices, and are typically written in natural language. Security models using mathematical formalisms can provide a precise formulation of the policies for implementation. More importantly, formally specified policy models can be mathematically verified to guarantee security properties. As mathematical abstractions, they provide unambiguous specifications that are independent of implementation mechanisms. Some concepts in security models include: subject, object, action, clearance level, user, group, role, task, principal, owner, etc.

Since security models are idealized abstractions, their application in real life requires a series of translations, involving interpretation and decision making at each stage. Organisational structures must be analyzed so as to select the appropriate models, or a combination of models. Policies need to be interpreted and codified properly to achieve the desired results. Real world entities and relationships are mapped to the model abstractions. Finally, the security model is mapped to security implementation mechanisms. The levels of abstractions used in security requirements, design, and implementation therefore mirror those in software system development and provide a basis for integration.

The social ontology outlined in this chapter can facilitate and augment an integrated security development process by enriching the reasoning support needed to arrive at decisions at each stage in the process. The ontology in existing security models are intended for the automated enforcement of specified security rules (e.g., to decide whether to give access). They do not support reasoning about why particular models or policies are appropriate for the target environment, especially when there are conflicting objectives and interpretations. Furthermore, many of the simplifying assumptions that formal models rely on do not hold in real life (Denning, 1999). The social ontology of strategic actors provides a framework for reasoning about the use of such models from a pragmatic, broader perspective.

In the development of new security models, there is a trend towards ontologies that are more closely aligned with the ontology of organisational work. For example, role based access control (RBAC) (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001; Sandhu, Coyne, Feinstein, & Youman, 1996) allows privileges to be organised according to organisational roles such as loan officer or branch manager. These trends are consistent with the proposed social ontology approach, though RBAC models, like other access control models, are meant for enforcement, not strategic organisational reasoning.

Security Management Frameworks

While formal computer security models focus on policies built into the automated system, the overall security of information and software systems depends very much on organisational practices. Security practices have existed long before the computer age. Many of the principles continue to apply and have been adapted to software systems. Standards have been defined to promote best practices (e.g., ISO 17799, 1999).

OCTAVE (Alberts & Dorofee, 2002), CRAMM, and FRAP (Peltier, 2001), are oriented toward decision making from a business perspective, leading to management, operational, and technical requirements and procedures. Although few frameworks have explicit information models, they do have implicit ontologies revolving around key concepts such as asset, attack, threat, vulner-

ability, countermeasure, and risk.

The main focus of these frameworks is on prescriptive guidelines. Tables and charts are used to enumerate and cross-list vulnerabilities and threats. Potential countermeasures are suggested. Risks are computed from potential losses arising from estimated likelihood of threats. Since quantitative estimates are hard to come by, most assessments rely on ratings such as low, medium, high.

While formal computer security models attempt to guarantee security (requiring simplifying assumptions that may depart from reality), security management frameworks acknowledge that security breaches will occur, and suggest countermeasures to reduce risk. This pragmatic stance is very much in the spirit of the social ontology proposed in this chapter. Security management frameworks can be augmented by the modelling of strategic actor relationships and reasoning about how their goals may be achieved or hindered.

Another drawback of checklists and guidelines is that they tend to be too generic. Experience and expert judgment are needed to properly apply them to specific systems and organisational settings. Such judgments are hard to trace or maintain over time as the systems evolve.

The explicit modelling of strategic relationships can provide a more specific analysis of sources of vulnerabilities and failures, thus also allowing countermeasures to be targeted appropriately. Using the strategic dependencies and rationales, one can trace the impact of threats along the paths to determine which business goals are affected. The impact on goals other than security can also be determined through the model since they appear in the same model. One can see how security goals might compete with or are synergistic with non-security goals, thus leading to decisions that take the overall set of goals into account. Using an agent-oriented ontology, one can determine which actors are most affected by which security threats, and are therefore likely to be most motivated to take measures. Tradeoffs are done from the viewpoint of each stakeholder. This approach provides a good basis for an ontology of security, which can mediate between business reasoning from an organisational perspective and system design reasoning from a technical perspective.

Some preliminary work have been done to integrate the ***i**** modelling ontology with risk-based security management approaches (Gaunard & Dubois, 2003; Mayer, Rifaut, & Dubois, 2005). Further extensions could incorporate economic theories and reasoning (e.g., Anderson, 2001; Camp & Lewis, 2004). The ontology of ***i**** can provide the structure representation of social relationships on which to do economic reasoning.

Software Systems Design Frameworks

Having considered work originating from the security side, we now turn to contributions from the software engineering and system development perspective.

Extensions to UML (see Chapter I for information of such approaches).The ontology of UML, consisting of objects and classes, activities, states, interactions, and so forth, with its security-oriented extensions, are useful for specifying the technical design of security features and functionalities, but does not support the reasoning that lead up to those requirements and designs. As indicated in the second section of this chapter, technical design notations are useful for recording the results of decisions, but do not offer support for arriving at those decisions. The social ontology proposed in this chapter can therefore complement UML-based approaches, such as the one presented in Chapter IX, by supporting the early-stage requirements modelling and reasoning that can then be propagated to the technical design stage, resulting in design choices expressed in UML-like design notations. Stakeholder deliberations and tradeoffs therefore

are effectively conveyed to technical designers. Conversely, the effect of technical choices can be propagated upstream to enable stakeholders to appreciate the consequences as they appear in the stakeholders' world.

Extensions to information systems modelling and design. In the information systems area, Pernul (1992) proposes secure data schemas (extension of entity-relationship diagrams) and secure function schemas (extension of data flow diagrams). In Herrmann and Pernul (1999) and Röhm and Pernul (1999), these models are extended to include a business process schema, with tasks, data/material, humans, legal bindings and information flow, and an organisational schema with role models and organisation diagrams to describe which activities are done where and by whom. Other information systems security approaches include the automated secure system development method (Booysen & Eloff, 1995) and the logical controls specification approach (Baskerville, 1993; Siponen & Baskerville, 2001).

These approaches illustrate the extension of conventional information systems ontologies to incorporate security-specific ontologies. Different concepts are added to each level of modelling (e.g., database schemas, process or function schemas, workflow schemas, and organisation diagrams). As with UML extensions, these approaches tend to emphasize the notation needed to express security features in the requirements specification or design descriptions and how those features can be analyzed. However, the notations (and the implied ontology) do not provide support for the deliberations that lead up to the security requirements and design. A social ontology that supports explicit reasoning about relationships among strategic actors, as outlined in this chapter, can be a helpful extension to these approaches.

Responsibility modelling. A number of approaches center around the notion of responsibility. In Strens and Dobson (1994), when an agent delegates an obligation, the agent becomes a responsibility principal, and the receiver of the delegation process is a responsibility holder. An obligation is a high-level mission that the agent can fulfill by carrying out activities. Agents cannot transfer their responsibilities, only their obligations. Three kinds of requirements are derived from responsibilities: need-to-do, need-to-know and need-for-audit. The need-to-know requirements relate to security policies — which subjects (e.g., users) should be allowed to access which objects (e.g., files, etc.) so that they are able to fulfill their responsibilities.

Backhouse and Dhillon (1996) also adopt a responsibilities analysis approach, incorporating speech acts theory. The model for automated profile specification (MAPS) approach (Pottas & Solms, 1995) uses responsibilities and role models to generate information security profiles (such as access control) from job descriptions and organisational policies.

This group of work has a more explicit ontology of social organisation. The emphasis is on the mappings between organisational actors and the tasks or activities they have to perform. While actors or agents have responsibilities, they are not viewed as having strategic interests, and do not seek alternate configurations of social relationships that favor those interests. The focus of attention is on functional behaviors and responsibilities. Security is treated as additional functions to be incorporated, and there are no attempts to deal with interactions and tradeoffs between security and other non-functional objectives such as usability or maintainability. The social ontology of ***i**** can therefore be quite complementary to these approaches. Other socio-organisational approaches are reviewed in Dhillon and Backhouse (2001).

Requirements Engineering Approaches to Security

While security needs to be integrated into all stages of software engineering, there is general agreement that integration starting from the

earliest stages is essential. It is well known that mistakes early in the software process can have far reaching consequences in subsequent stages that are difficult and costly to remedy. Fred Brooks (1995) had noted that the requirements stage is the most difficult, and suggested that software engineering should focus more on "building the right system," and not just on "building the system right."

In requirements engineering research, a large part of the effort has been devoted to verifying that the requirements statements are precise, unambiguous, consistent, and complete. Recently, more attention has been given to the challenge of understanding the environment and context of the intended system so that the requirements will truly reflect what stakeholders want.

Goal-oriented requirements engineering. Traditional requirements languages for software specification focus on structure and behavior, with ontologies that center around entities, activities, states, constraints, and their variants. A goal-oriented ontology allows systems to be placed within the intentional setting of the usage environment. Typically, goal-oriented requirements engineering frameworks employ AND/OR tree structures (or variants) to analyze and explore alternate system definitions that will contribute to stakeholder goals in different ways. Security can be readily integrated into such a framework since attacks and threats interfere with the normal achievement of stakeholder goals. Security controls and countermeasures can be derived from defensive goals to counteract malicious actions and intents.

The NFR framework: Security as softgoal. The NFR framework (Chung, 1993; Chung et al., 2000) is distinctive from most of the above cited approaches to security in that it does not start with vulnerabilities and risks, nor from security features and functions. It starts by treating security as one among many non-functional requirements. As with many other non-functional requirements such as usability, performance, or information accuracy, security is viewed as a goal whose operational meaning needs to be interpreted according to the needs of the specific application setting. This interpretation is done by a series of refinements in a goal graph until the point (called operationalization) where subgoals are sufficiently concrete as to be accomplishable by implementable actions and mechanisms, such as access control mechanisms or protocols. At each stage in the refinement, subgoals are judged to be contributing qualitatively to the parent goals in different ways. Because the nature and extent of the contribution requires judgement from experience and possibly domain expertise, the term softgoal is used, drawing on Simon's notion of satisficing (Simon, 1996).

The NFR framework thus offers a systematic approach for achieving "good enough" security — a practical objective in real life (Sandhu, 2003; Schneier, 2003) that have been hard to achieve in conventional mathematical formalisms. A formal treatment of the satisficing semantics of softgoals is offered in Chung et al. (2000).

The NFR framework is also distinctive in that it allows security goals to be analyzed and understood at the same time as other potentially competing requirements, for example, usability, performance, maintainability, and evolvability. In the past, it has been difficult to deal with these non-functional requirements early in the development life cycle. Typically functional requirements dominate the design process. Experienced and expert designers take non-functional requirement into account intuitively and implicitly, but without support from systematic frameworks, languages, or tools. The softgoal graph approach acknowledges that security needs to compete with other goals during requirements analysis and during design. Different aspects of security may also compete with each other. The NFR goal-oriented approach supports reasoning about tradeoffs among these competing goals and how they can be achieved.

Beyond clarifying requirements, the NFR softgoals are used to drive subsequent stages in

system design and implementation, thus offering a deep integration of security into the software engineering process.

A related body of work is in quality attributes of software architecture, for example, the ATAM approach (Kazman, Klein, & Clements, 2000) for architectural evaluation. Many of the basic elements are similar to the NFR framework. The classification of quality attributes and mechanisms (for security and other attributes), however, are viewed from an evaluation viewpoint. The taxonomy structure of quality attribute is not seen as goals to be elaborated based on tradeoffs encountered in the particular system. Quality attributes are concretized in terms of metrics, which are different for each quality, so trade-offs are difficult across different metrics.

The KAOS framework: Goals, obstacles, and anti-goals. KAOS (Dardenne, van Lamsweerde, & Fickas, 1993; van Lamsweerde, 2001, 2004; van Lamsweerde, Brohez, Landtsheer, & Janssens, 2003) is a goal-oriented requirements engineering framework that focuses on systematic derivation of requirements from goals. It includes an outer layer of informally specified goals, and an inner layer of formalized goal representation and operations using temporal logic. It is therefore especially suitable for real-time and safety critical systems. Refinement patterns are developed making use of temporal logic relationships.

The KAOS ontology includes obstacles, which impede goal achievement. The methodology provides techniques for identifying and resolving obstacles. To incorporate security analysis, attackers present obstacles to security goals. New security requirements are derived from attack generation and resolution.

Tree structures have been used in the security community for analyzing the structure of threats (Schneier, 1999), and in the safety community for the analysis of faults and hazards (Helmer et al., 2002). Experiences from these approaches can be incorporated into goal-oriented frameworks.

Agent-Oriented Requirements Engineering

The agent-oriented approach adopts goal-oriented concepts and techniques, but treats goals as originating from different actors. The ***i**** modelling framework views actors as having strategic interests. Each actor aims to further its own interests in exploring alternative conceptions of the future system and how the system will affect its relationships to other actors. This may be contrasted with other frameworks which may include some notion of actor which are non-intentional (e.g., in use case diagrams in UML) or non-strategic (e.g., in KAOS, where agents are passive recipients of responsibility assignments at the end of a goal refinement process).

i* adopts the notion of softgoal from the NFR framework, but makes further distinctions with goal, task, and resource. Softgoals are operationalized into tasks, which may in turn contain decompositions that include softgoals.

Security issues are traced to antagonistic goals and dependencies among attackers and defenders. As in the NFR framework, security is treated as much as possible within the same notational and reasoning framework as for other non-functional requirements (as softgoals), but extended to include functional elements (as goals, tasks, and resources). Security is therefore not treated in isolation, but interacts with other concerns at all steps throughout the process. The illustration of ***i**** in this chapter is based on the example in Yu and Liu (2000, 2001). Further illustrations are in Liu et al. (2002), Yu and Cysneiros (2001), Liu et al. (2003), Liu and Yu (2003, 2004).

The ***i**** approach has been adopted and extended in a number of directions. The Tropos framework (Bresciani, Perini, Giorgini, Giunchiglia, & Mylopoulos, 2004; Castro, Kolp, & Mylopoulos, 2002) further develops the ***i**** approach into a full-fledged software engineering methodology, using the agent-oriented social ontology originating from

requirements modelling to drive architectural design, detailed design, and eventual implementation on agent-based software platforms. Formal Tropos incorporates formalization techniques similar to KAOS, so that automated tools such as model checking can be applied to verify security properties (Liu et al., 2003).

A number of extensions to ***i**** have been developed to address specific needs of security modelling and analysis. Mouratidis et al. (2003a, 2003b, 2004, 2005, also Chapter VIII) introduced the concepts of security reference diagram and security constraints. Common security concepts such as secure entities, secure dependencies, and secure capabilities are reinterpreted within the ***i**** ontology. The security constraint concept attaches a security-related strategic dependency to the dependency that it applies to. An intuitive benefit of this concept is that the association between the two is indicated without having to refer to the internal rationale structures of actors. An attack scenarios representation structure that aims to support the analysis of specific attacking and protecting situations at a more detailed design stage is developed. New language structures developed include secure capability, and attacking link.

Giorgini et al. (2003, 2005; also Chapter VIII) introduced four new primitive relationships related to security requirements: trust, delegation, offer and owner relation. These new primitives offer an explicit treatment of security concepts such as permission, ownership, and authority, which allows a more detailed analysis.

In Crook, Ince, and Nuseibeh (2005), the problem of modelling access policies is addressed by extending the Tropos approach (Liu et al., 2003), to ensure that security goals can be achieved and that operational requirements are consistent with access policies.

Misuse/Abuse Cases

Misuse and abuse cases techniques (Alexander, 2001; Sindre & Opdahl, 2000, 2001; see also Review in Chapter I) are complementary to goal-oriented techniques as they offer different ways of structuring requirements knowledge (Rolland, Grosz, & Kla, 1999). Use cases are action-oriented and include sequence and conditionals. Goal refinements are (mostly) hierarchical covering multiple levels of abstraction. In addressing security requirements, the development of misuse/abuse cases can be assisted by using goal analysis. Conversely, goal analysis can be made concrete by considering positive and negative use cases and scenarios. Note that use cases are better suited to later stages in requirements analysis since they assume that the system boundary is already defined. Unlike the strategic actors in ***i****, actors in use cases are non-intentional and serve to delineate the boundary of the automated system.

CONCLUSION

In this chapter, we have argued that a social ontology can provide the basis for integrating security and software engineering. We presented the social ontology of ***i**** and illustrated how it can be used to include security goals when designing a smart card system. We have outlined how a social ontology is complementary to a number of techniques in security engineering and in software engineering, thus building common ground between the two areas.

ACKNOWLEDGMENT

The authors (1 & 3) gratefully acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada, Bell University Laboratories, and author (2) the National Key Research and Development Plan (973, no.2002CB312004) and NSF China (no. 60503030).

REFERENCES

Alberts, C., & Dorofee, A. (2002, July). *Managing information security risks: The OCTAVE (SM) approach*. Boston: Addison Wesley.

Alexander, I. (2002, September). Modelling the interplay of conflicting goals with use and misuse cases. *Proceedings of the 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ-02)*, Essen, Germany (pp. 9-10).

Alexander, I. (2003, January). Misuse cases: Use cases with hostile intent. *IEEE Software, 20*(1), 58-66.

Anderson, R. (2001). *Security engineering: A guide to building dependable distributed systems.* New York: Wiley.

Backhouse, J., & Dhillon, G. (1996). Structures of responsibilities and security of information systems. *European Journal of Information Systems, 5*(1), 2-10.

Baskerville, R. (1993). Information systems security design methods: Implications for information systems development. *Computing Surveys, 25*(4), 375-414.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer, 21*(5), 61-72.

Booysen, H. A. S., & Eloff, J. H. P. (1995). A methodology for the development of secure application systems. *Proceeding of the 11th IFIP TC11 International Conference on Information Security.*

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004) Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems, 8*(3), 203-236.

Brooks, F. (1995, August). *The mythical man-month: Essays on software engineering, 20th Anniversary Edition* (1st ed.). Boston: Addison-Wesley.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements driven information systems engineering: The Tropos project. *Information Systems, 27*(6), 365-389.

Chung, L. (1993). Dealing with security requirements during the development of information systems. In C. Rolland, F. Bodart, & C. Cauvet (Eds.), *Proceedings of the 5th International Conference Advanced Information Systems Engineering, CAiSE '93* (pp. 234-251). Springer.

Chung L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2000). *Non-functional requirements in software engineering.* Kluwer Academic Publishers.

CRAMM – CCTA (Central Computer and Telecommunications Agency, UK). *Risk analysis and management method*. Retrieved from http://www.cramm.com/cramm.htm

Crook, R., Ince, D., & Nuseibeh, B. (2005, August 29-September 2). On Modelling access policies: Relating roles to their organisational context. *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*, Paris (pp. 157-166).

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming, 20*(1-2), 3-50.

Denning, D. E. (1998). *The limits of formal security models.* National Computer Systems Security Award Acceptance Speech. Retrieved October 18, 1999, from www.cs.georgetown.edu/~denning/infosec/award.html

Dhillon, G., & Backhouse, J. (2001) Current directions in IS security research: Toward socio-organizational perspectives. *Information Systems Journal, 11*(2), 127-154.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, R., & Chandramouli, R. (2001, August). Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security, 4*(3), 224-74.

Franch, X., & Maiden, N. A. M. (2003, February 10-13). Modelling component dependencies to inform their selection. *COTS-Based Software Systems, 2nd International Conference, (ICCBSS 2003)* (pp. 81-91). Lecture Notes in Computer Science 2580. Ottawa, Canada: Springer.

Gaunard, P., & Dubois, E. (2003, May 26-28). Bridging the gap between risk analysis and security policies: Security and privacy in the age of uncertainty. *IFIP TC11 18th International Conference on Information Security (SEC2003)* (pp. 409-412). Athens, Greece. Kluwer.

Giorgini, P., Massacci, F., & Mylopoulos, J. (2003, October 13-16). Requirement engineering meets security: A case study on modelling secure electronic transactions by VISA and Mastercard. *The 22nd International Conference on Conceptual Modelling (ER'03)* (LNCS 2813, pp. 263-276). Chicago: Springer.

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2005). Modelling social and individual trust in requirements engineering methodologies. *Proceedings of the 3rd International Conference on Trust Management (iTrust 2005).* LNCS 3477. Heidelberg: Springer-Verlag.

Gross, D., & Yu, E. (2001, August 27-31). Evolving system architecture to meet changing business goals: An agent and goal-oriented approach. The *5th IEEE International Symposium on Requirements Engineering (RE 2001)* (pp. 316-317). Toronto, Canada.

Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., & Lutz, R. (2002). A software fault tree approach to requirements analysis of an intrusion detection system. In P. Loucopoulos & J. Mylopoulos (Ed.), *Special Issue on Requirements Engineering for Information Security. Requirements Engineering* (Vol. 7, No. 4, pp. 177-220).

Herrmann, G., & Pernul, G. (1999). Viewing business-process security from different perspectives. *International Journal of Electronic Commerce, 3*(3), 89-103.

ISO 17799. (1999). *Information security management — Part 1: Code of practice for information security.* London: British Standards Institution.

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for architectural evaluation (CMU/SEI-2000-TR-004).* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Liu, L., & Yu, E. (2003). Designing information systems in social context: A goal and scenario modelling approach. *Information Systems, 29*(2), 187-203.

Liu, L., & Yu, E. (2004). Intentional modelling to support identity management. In P. Atzeni et al. (Eds.), *Proceedings of the 23rd International Conference on Conceptual Modelling (ER 2004)* (pp. 555-566). LNCS 3288. Berlin, Heidelberg: Springer-Verlag.

Liu, L., Yu, E., & Mylopoulos, J. (2002, October 16). Analyzing security requirements as relationships among strategic actors. The *2nd Symposium on Requirements Engineering for Information Security (SREIS'02).* Raleigh, NC.

Liu, L., Yu, E., & Mylopoulos, J. (2003, September). Security and privacy requirements analysis within a social setting. *Proceedings of International Conference on Requirements Engineering (RE'03)* (pp. 151-161). Monterey, CA.

Lodderstedt, T., Basin, D. A., J, & Doser, R. (2002). SecureUML: A UML-based modelling language for model-driven security. *Proceedings of UML '02: Proceedings of the 5th International Conference on The Unified Modelling Language,* Dresden, Germany (pp. 426-441).

Mayer, N., Rifaut, A., & Dubois, E. (2005). Towards a risk-based security requirements engineering framework. *Workshop on Requirements Engineering For Software Quality (REFSQ'05), at the Conference for Advanced Information Systems Engineering (CAiSE),* Porto, Portugal.

McDermott, J., & Fox, C. (1999). Using abuse case models for security requirements analysis. *Proceedings 15th IEEE Annual Computer Security Applications Conference*, Scottsdale, USA (pp. 55-67).

Mouratidis, H., Giorgini, P., & Manson, G. A. (2003a). Integrating security and systems engineering: Towards the modelling of secure information systems. *Proceedings of the 15th Conference on Advanced Information Systems Engineering (CAiSE 03)* (Vol . LNCS 2681, pp. 63-78). Klagenfurt, Austria: Springer.

Mouratidis, H., Giorgini, P., & Manson, G. (2004, April 13-17). Using security attack scenarios to analyse security during information systems design. *Proceedings of the 6th International Conference on Enterprise Information Systems*, Porto, Portugal.

Mouratidis, H., Giorgini, P., & Schumacher, M. (2003b). Security patterns for agent systems. *Proceedings of the 8th European Conference on Pattern Languages of Programs*, Irsee, Germany.

Mouratidis, H., Kolp, M., Faulkner, S., & Giorgini. P. (2005, July). A secure architectural description language for agent systems. *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS05).* Utrecht, The Netherlands: ACM Press.

Peltier, T. R. (2001, January). *Information security risk analysis.* Boca Raton, FL: Auerbach Publications.

Pernul, G. (1992, November 23-25). Security constraint processing in multilevel secure AMAC schemata. The *2nd European Symposium on Research in Computer Security (ESORICS 1992)* (pp. 349-370). Toulouse, France. Lecture Notes in Computer Science 648. Springer.

Pottas, D., & Solms, S. H. (1995). Aligning information security profiles with organizational policies. *Proceedings of the IFIP TC11 11th International Conference on Information Security.*

Röhm, A. W., & Pernul, G. (1999). COPS: A model and infrastructure for secure and fair electronic markets. *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences.*

Rolland, C., Grosz, G., & Kla, R. (1999, June). Experience with goal-scenario coupling in requirements engineering. *Proceedings of the IEEE International Symposium on Requirements Engineering*, Limerick, Ireland.

Samarati, P., & Vimercati, S. (2001). Access control: Policies, models, and mechanisms. In R. Focardi & R. Gorrieri (Eds.), *Foundations of security analysis and design: Tutorial lectures* (pp. 137-196). LNCS 2171.

Sandhu, R. (2003, January/February). Good enough security: Towards a business driven discipline. *IEEE Internet Computing, 7*(1), 66-68.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996, February). Role-based access control models. *IEEE Computer, 29*(2), 38-47.

Schneier, B. (1999). *Attack trees modelling security threats.* Dr. Dobb's Journal, December. Retrieved from http://www.counterpane.com/attacktrees-ddj-ft.html

Schneier, B. (2003). *Beyond fear: Thinking sensibly about security in an uncertain world.* New York: Copernicus Books, an imprint of Springer-Verlag.

Schneier, B., & Shostack, A. (1998). *Breaking up is hard to do: Modelling security threats for smartcards.* First USENIX Symposium on Smart-Cards,

USENIX Press. Retrieved from http://www.counterpane.com/smart-card-threats.html

Simon, H. (1996). *The sciences of the artificial* (3rd ed.). MIT Press.

Sindre, G., & Opdahl, A. L. (2000). Eliciting security requirements by misuse cases. *Proceedings of the 37th Conference on Techniques of Object-Oriented Languages and Systems* (pp. 120-131). TOOLS Pacific 2000.

Sindre, G., & Opdahl, A. L. (2001, June 4-5). Templates for misuse case description. *Proceedings of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ2001),* Switzerland.

Siponen, M. T., & Baskerville, R. (2001). A new paradigm for adding security into IS development methods. In J. Eloff, L. Labuschagne, R. von Solms, & G. Dhillon (Eds.), *Advances in information security management & small systems security* (pp. 99-111). Boston: Kluwer Academic Publishers.

Strens, M. R., & Dobson, J. E. (1994). Responsibility modelling as a technique for requirements definition. *IEEE, 3*(1), 20-26.

van der Raadt, B., Gordijn, J., & Yu, E. (2005). Exploring Web services from a business value perspective. To appear in *Proceedings of the 13th International Requirements Engineering Conference (RE'05),* Paris (pp. 53-62).

van Lamsweerde, A. (2001, August 27-31). Goal-oriented requirements engineering: A guided tour. The *5th IEEE International Symposium on Requirements Engineering (RE 2001)* (p. 249). Toronto, Canada.

van Lamsweerde, A. (2004, May). Elaborating security requirements by construction of intentional anti-models. *Proceedings of ICSE'04, 26th International Conference on Software Engineering* (pp. 148-157). Edinburgh: ACM-IEEE.

van Lamsweerde, A., Brohez, S., Landtsheer, R., & Janssens, D. (2003, September). From system goals to intruder anti-goals: Attack generation and resolution for security requirements engineering. *Proceedings of the RE'03 Workshop on Requirements for High Assurance Systems (RHAS'03)* (pp. 49-56). Monterey, CA.

Yu, E. (1993, January). Modelling organizations for information systems requirements engineering. *Proceedings of the 1st IEEE International Symposium on Requirements Engineering* (pp. 34-41). San Diego, CA.

Yu, E. (1997, January 6-8). Towards modelling and reasoning support for early-phase requirements engineering. *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)* (pp. 226-235). Washington, DC.

Yu, E. (2001a, April). Agent orientation as a modelling paradigm. *Wirtschaftsinformatik, 43*(2), 123-132.

Yu, E. (2001b). Agent-oriented modelling: Software versus the world. *Agent-Oriented Software Engineering AOSE-2001 Workshop Proceedings* (LNCS 222, pp. 206-225). Springer Verlag.

Yu, E., & Cysneiros, L. (2002, October 16). Designing for privacy and other competing requirements. *The 2nd Symposium on Requirements Engineering for Information Security (SREIS'02).* Raleigh, NC.

Yu, E., & Liu, L. (2000, June 3-4). Modelling trust in the i* strategic actors framework. *Proceedings of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies,* Barcelona, Catalonia, Spain (at Agents2000).

Yu, E., & Liu, L. (2001). Modelling trust for system design using the ***i**** strategic actors framework. In R. Falcone, M. Singh, & Y. H. Tan (Eds.), *Trust in cyber-societies--integrating the human and artificial perspectives* (pp. 175-194). LNAI-2246. Springer.

Yu, E., Liu, L., & Li, Y. (2001, November 27-30). Modelling strategic actor relationships to support intellectual property management. The *20th International Conference on Conceptual Modelling (ER-2001)* (LNCS 2224, pp. 164-178). Yokohama, Japan: Spring Verlag.

This work was previously published in Integrating Security and Software Engineering: Advances and Future Visions, edited by H. Mouratidis and P. Giorgini, pp. 70-106, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 2.15
Social Structure Based Design Patterns for Agent-Oriented Software Engineering

Manuel Kolp
Université catholique de Louvain, Belgium

Stéphane Faulkner
University of Namur, Belgium

Yves Wautelet
Université catholique de Louvain, Belgium

ABSTRACT

Multi-agent systems (MAS) architectures are gaining popularity over traditional ones for building open, distributed, and evolving software required by today's corporate IT applications such as e-business systems, Web services, or enterprise knowledge bases. Since the fundamental concepts of multi-agent systems are social and intentional rather than object, functional, or implementation-oriented, the design of MAS architectures can be eased by using social patterns. They are detailed agent-oriented design idioms to describe MAS architectures composed of autonomous agents that interact and coordinate to achieve their intentions, like actors in human organizations. This article presents social patterns and focuses on a framework aimed to gain insight into these patterns. The framework can be integrated into agent-oriented software engineering methodologies used to build MAS. We consider the Broker social pattern to illustrate the framework. An overview of the mapping from system architectural design (through organizational architectural styles), to system detailed design (through social patterns), is presented with a data integration case study. The automation of creating design patterns is also discussed.

INTRODUCTION

This section introduces and motivates the research. In Section 1.1, we describe the advantages of using multi-agent systems over traditional systems. Section 1.2 presents the importance of *patterns* for designing information systems. We formulate our research proposal in Section 1.3. The section 1.4 introduces elements for work validation. The context of the research and an overview of the state of the art are given in Section 1.5. Finally, Section 1.6 presents the organization of the article.

Advantages of Multi-Agent Systems

The meteoric rise of Internet and World Wide Web technologies has created new application areas for enterprise software, including e-business, Web services, ubiquitous computing, knowledge management and peer-to-peer networks. These areas demand software that is robust, can operate within a wide range of environments, and evolve over time to cope with changing requirements. Moreover, such software has to be highly customizable to meet the needs of a wide range of users, and sufficiently secure to protect personal data and other assets on behalf of its stakeholders.

Not surprisingly, researchers are looking for new software designs that can cope with such requirements. One promising source for designing such business software is the area of multi-agent systems. Multi-agent system architectures appear to be more flexible, modular, and robust than traditional architectures including object-oriented ones. They tend to be open and dynamic in the sense they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

Multi-agent systems are based on the concept of agent which is defined as "a software component situated in some environment that is capable of flexible autonomous action in order to meet its design objective" (Aridor & Lange, 1998). An agent exhibits the following characteristics:

- **Autonomy:** an agent has its own internal thread of execution, typically oriented to the achievement of a specific task, and it decides for itself what actions it should perform at what time.
- **Situateness:** agents perform their actions in the context of being situated in a particular environment. This environment may be a computational one (e.g., a Web site) or a physical one (e.g., a manufacturing pipeline). The agent can sense and affect some portion of that environment.
- **Flexibility:** in order to accomplish its design objectives in a dynamic and unpredictable environment, the agent may need to act to ensure that its goals are achieved (by realizing alternative plan). This property is enabled by the fact that the agent is autonomous in its problem solving.

An agent can be useful as a stand-alone entity that delegates particular tasks on behalf of a user (e.g., a personal digital assistant and e-mail filter (Bauer, Muller & Odell, 2001), or a goal-driven office delivery mobile device (Castro, Kolp & Mylopoulos, 2002)). However, in the overwhelming majority of cases, agents exist in an environment that contains other agents. Such environment is a multi-agent system (MAS).

In MAS, the global behavior derives from the interaction among the constituent agents: they cooperate, coordinate or negotiate with one another. A multi-agent system is then conceived as a society of autonomous, collaborative, and goal-driven software components (agents), much like a social organization. Each role an agent can play has a well defined set of responsibilities (goals) achieved by means of an agent's own abilities, as well as its interaction capabilities.

This *sociality* of an MAS is well suited to tackling the complexity of today's organization software systems for a number of reasons:

- It permits a better match between system architectures and its organizational operational environment for example a public organization, a corporation, a non-profit association, a local community, and so forth.
- The autonomy of an agent (i.e., the ability an agent has to decide what actions it should take at what time (Aridor & Lange, 1998)) reflects the social and decentralized nature of modern enterprise systems (Bauer et al., 2001) that are operated by different stakeholders (Parunak, 1997).
- The flexible way in which agents operate to accomplish its goals is suited to the dynamic and unpredictable situations in which business software is now expected to run (see Zambonelli, Jennings, Omicini & Wooldridge, 2000; Zambonelli, Jennings & Wooldridge, 2000).

MAS architectures become rapidly complicated due to the ever-increasing complexity of these new business domains and their human or organizational actors. As the expectations of the stakeholders change every, the complexity of the systems and the communication technologies of organizations continually increases in today's dynamic environments. Developers are expected to produce architectures that must handle more difficult and intricate requirements that were not taken into account 10 years ago, making thus architectural design a central engineering issue in modern enterprise information system life-cycle (Aridor & Lange, 1998).

Patterns for Designing Systems

An important technique that helps to manage this complexity when constructing and documenting such architectures is the reuse of development experience and know how. Over the past few years, *design patterns* have significantly contributed to the reuse of design expertise, improvement application documentation and more flexible and adaptable designs (Bosch, 1998; Buschmann, Meunier, Rohnert, Sommerlad & Stal, 1996; Gamma, Helm, Johnson & Vlissides, 1995). The idea behind a pattern is to record the essence of a solution to a design problem so as to facilitate its reuse when similar problems are encountered (Cockburn, 1996; Pree 1994; Riehle & Züllighoven, 1996).

Considerable work has been done in software engineering on defining design patterns (Gamma et al., 1995; Buschmann et al., 1996; Bosch, 1998). Unfortunately, they focus on object-oriented (Fernandez & Pan, 2001) rather than agent-oriented systems. In the area of multi-agent systems, little emphasis has been put on social and intentional aspects. Moreover, the proposals of agent patterns that could address those aspects (see e.g., Aridor & Lange, 1998; Deugo, Oppacher, Kuester & Otte, 1999; Hayden, Carrick & Yang, 1999) are not aimed at the design level, but rather at the implementation of lower-level issues like agent communication, information gathering, or connection setup. For instance, the Foundation for Intelligent Physical Agents (FIPA, 2007) identified and defined a set of agent's interaction protocols that are only restricted to communication. This research fills this gap by propping a series of social patterns for the detailed design phase of Tropos so that pattern-oriented development can be fully integrated at higher level development stages.

A Framework for MAS Detailed Design

Since there is a fundamental mismatch between the concepts used by the object-oriented paradigm (and other traditional mainstream software engineering approaches) and the agent-oriented approach (Jennings & Wooldridge, 2001), there

is a need to develop high level patterns that are specifically tailored to the development of (multi-)agent systems using agent-oriented primitives.

Research objective is to take the principles of a social organization-based approach that contributes to reduce the distance between the system and the real world together with the important role of design patterns to help to reuse design experience. This research proposes a design framework and develops a catalogue of social patterns for making MAS design more efficient. Research contributions include:

- A framework composed of a set of complementary dimension for designing MAS. The concepts and notions used for each dimension are introduced and illustrated;
- A catalogue of social patterns to help the designer's tasks so that development time is reduced. Each social pattern in the catalogue will be designed in detail through this framework; and
- A tool for designing MAS. It allows the designer to: (i) design the components of a MAS-to-be constructed in a graphical way, (ii) reuse the catalogue of patterns to construct the MAS, and (iii) generate the code for automating the programmer task.

The research also brings secondary contributions:

- A set of predefined predicates integrated into an extended version of formal Tropos for formalizing each pattern; and
- The illustration of concepts introduced in our framework through a case study.

Validation

The social patterns for developing a business data integration application have been applied to multiple case studies. The reusability of these patterns and the code generation help to reduce the development tasks of the application on both designer and programmer sides.

Furthermore, an empirical experience to evaluate the benefits of pattern-oriented development should be to achieve similar case studies with and without the use of patterns and to evaluate the results on the basis of software metrics. To focus on the contribution of the design-patterns we point to structural complexity evaluation. Indeed, structural complexity focuses on MAS architecture and agent relationships, features that should be enriched using patterns.

Due to the lack of literature concerning agent-oriented software metrics evaluating structural complexity, we point to the use of existing object-oriented ones. As a preliminary tests suite, we claim for the use of the metrics proposed by Chidamber and Kemerer (1994). Those include:

- The depth of inheritance tree (DIT). This metric measures the maximum level of the inheritance hierarchy of a class. The root of the inheritance tree inherits from no class and has a DIT count of zero. Chidamber and Kemerer suggest that DIT can be used to indicate the complexity of the design, potential for reuse;
- The number of children (NOC). This metric counts the number of immediate subclasses belonging to a class. NOC was intended to indicate the level of reuse in a system and a possible indicator of the level of testing required;
- The lack of cohesion in methods (LCOM). This metric is intended to measure the lack of cohesion in the methods of a class. It is based on the principle that different attributes occurring in different methods of a class causes that class to be less cohesive than one where the same attribute is used in few methods of the class. It is viewed that a lack of cohesiveness as undesirable as it

is against encapsulation. Lack of cohesion could imply that the class should probably be split into two or more subclasses;
- The weighted methods per class (WMC). The sum of the complexities of the methods in a class;
- The coupling between objects (CBO). The number of other classes whose methods or instance attribute(s) are used by methods of this class;
- The response for a class (RFC). The sum of the number of methods in the class and the number of methods called by each of these methods, where each called method is counted once.

Context of the Research and Limitations

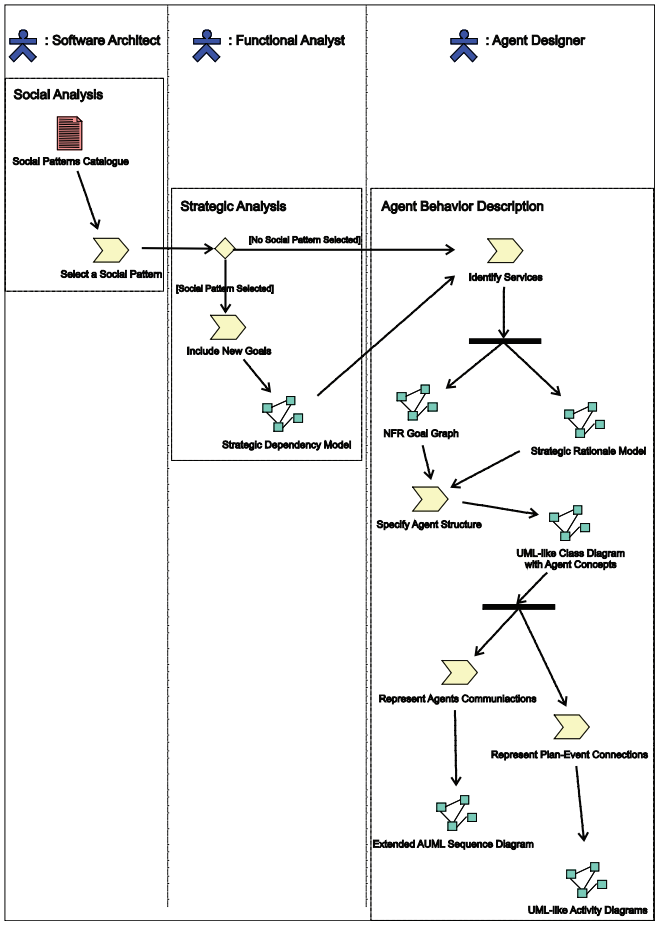
Design patterns are generally used during the *detailed design* phase of software methodologies. Agent-oriented methodologies such as TROPOS (Castro et al., 2002), GAIA (Woodridge, Jennings & Kinny, 2000), MASE (Wood, DeLoach & Sparkman, 2001) and MESSAGE (Caire et al., 2002) span the following steps of software engineering:

- Early requirements, concerned with the understanding of a problem by studying an organizational setting; the output is an organizational model which includes relevant actors, their goals and their interdependencies.
- Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design, where the system architecture is defined in terms of subsystems, interconnected through data, control, and dependencies.
- Detailed design, where the behavior of each architectural components is defined in detail.

The catalogue of social patterns proposed in (Kolp, Giorgini, & Mylopoulos, 2002) constitutes a contribution to the definition of agent-oriented design patterns. This article focuses on these patterns, conceptualizes a framework to explore them and facilitate the building of MAS during detailed design as well as the generation of code for agent implementation. It models and introspects the patterns along different complementary dimensions.

As pointed out, the patterns proposed into this article take place at Tropos' detailed design step. The process described hereafter is part of a broader methodology called I-Tropos (Wautelet, Kolp & Achbany, 2006) based on Tropos, driven by i* diagrams and organized following an iterative software development life cycle. This methodology is conceived to bring agent-oriented development to be adopted into real life development of huge enterprise information systems. Due to lack of space we only present the detailed design discipline in the form of a workflow, more details can be found in (Kolp, Faulkner & Wautelet, 2007). Figure 1 describes the workflow of the detailed design discipline using the *software process engineering metamodel* (*SPEM*) notation (see OMG, 2005). The software architect selects the most appropriate social patterns for the components under development from the catalogue overviewed in the article. New goals are included to the strategic dependency model (social dimension) according to the semantics of the pattern. The agent designer identifies services provided by each agent to achieve the goal dependencies. Each service belongs to an agent and is represented with an NFR goal analysis to refine the strategic rationale diagram (intentional dimension). The structure of each agent and its components such

Figure 1. The detailed design workflow



as plans, events, and beliefs are then specified with an agent UML class diagram (structural dimension). Agents communicate through events exchanged in the system and modeled in a temporal manner with extended agent UML sequence diagrams (communicational dimension). The synchronization and the relationships between plans and events are designed through agent oriented activity diagrams (dynamic dimension).

We nevertheless point out some important limitations of our research:

- We only consider the design of cooperative MASs. Indeed, MAS may be either cooperative or competitive. In a cooperative MAS, the agents cooperate together in order to achieve common goals. Inversely, a competitive MAS is composed of agents that pursue personal goals and defend their own interests. The design of competitive MAS is left for future developments;
- The patterns need to gain experience with their use. In this article, we have applied them on a case study. By doing so, we have explored the applicability of patterns and shown how our framework can help the design of MAS. However, it should be tested on more case studies;

- The dissertation only considers an MAS composed of more than one pattern as an "addition" of them. However, the combination of multiple patterns in an MAS is more complicated than that, and the emergence of conflicts remains possible. This issue needs further investigation.

Paper Organization

The article is organized as follows. In section 2, we describe the patterns. Section 3 proposes the framework and illustrates its different modeling dimensions through the broker pattern. A *data integrator* case study that illustrates the mapping from organizational styles (architectural design phase) to social patterns (detailed design phase), is presented in section 4. The automation of social patterns is overviewed in section 5, while section 6 overviews related work on software patterns. Finally, section 7 points to some conclusions.

SOCIAL PATTERNS

Social patterns can be classified in two categories. The *pair* patterns describe direct interactions between negotiating agents. The *mediation* patterns feature intermediate agents that help other agents to reach agreement about an exchange of services.

In the following, we briefly model patterns using i* (Do, Kolp, Hang Hoang, & Pirotte, 2003) and AUML (Bauer et al., 2001) sequence diagrams respectively to represent the social and communicational dimensions of each pattern. In i*, agents are drawn as circles and their intentional dependencies as ovals. An agent (the *depender*) depends upon another agent (the *dependee*) for an intention to be fulfilled (the *dependum*). Dependencies have the form *depender* → *dependum* → *dependee*. Note that i* also allows to model other kind of dependencies such as resource, task or strategic ones respectively represented as rectangles,

Figure 2. Social and communicational diagrams for the booking pattern

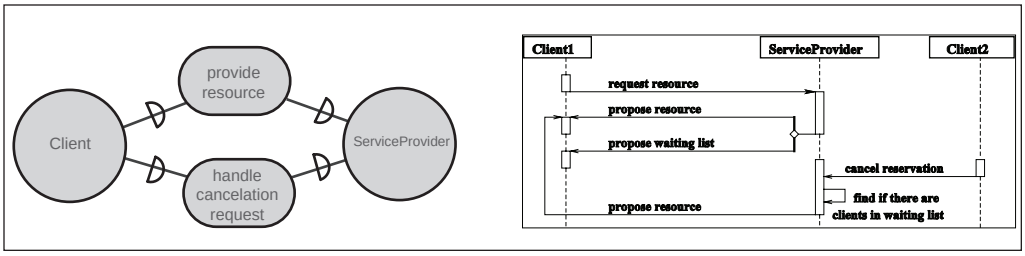
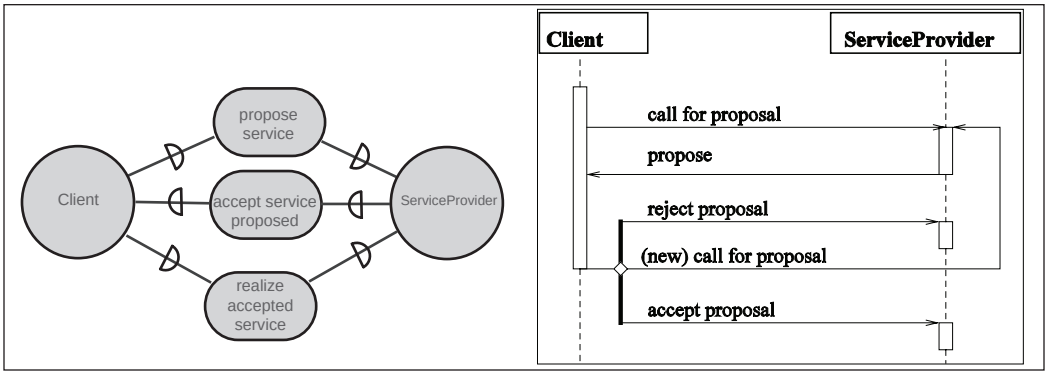


Figure 3. Social and communicational diagrams for the bidding pattern



hexagons and clouds as we will see in Figure 13. AUML extends classical sequence diagrams for agent oriented modeling. For instance, the diamond symbol indicates alternative events.

The broker, as well as the subscription and call-for-proposal patterns that are both part of the broker pattern, will be modeled in detail to explain the framework in section 3.

Pair Patterns

The **booking** pattern (Figure 2) involves a client and a number of service providers. The client issues a request to book some resource from a service provider. The provider can accept the request, deny it, or propose to place the client on a waiting list, until the requested resource becomes available when some other client cancels a reservation.

The **subscription** pattern involves a yellow-page agent and a number of service providers. The providers advertise their services by subscribing to the yellow pages. A provider that no longer wishes to be advertised can request to be unsubscribed.

The **call-for-proposals** pattern involves an initiator and a number of participants. The initiator issues a call for proposals for a service to all participants and then accepts proposals that offer the service for a specified cost. The initiator selects one participant to supply the service.

The **bidding** (Figure 3) pattern involves a client and a number of service providers. The client organizes and leads the bidding process, and receives proposals. At each iteration, the client publishes the current bid; it can accept an offer, raise the bid, or cancel the process.

Mediation Patterns

In the **monitor** pattern (Figure 4), subscribers register for receiving, from a monitor agent, notifications of changes of state in some subjects of their interest. The monitor accepts subscriptions, requests information from the subjects of interest, and alerts subscribers accordingly.

In the **broker** pattern, the broker agent is an arbiter and intermediary that requests services from providers to satisfy the request of clients.

In the **matchmaker** pattern (Figure 5), a matchmaker agent locates a provider for a given service requested by a client, and then lets the client interact directly with the provider, unlike brokers, who handle all interactions between clients and providers.

In the **mediator** pattern (Figure 6), a mediator agent coordinates the cooperation of service provider agents to satisfy the request of a client agent.

Figure 4. Social and communicational diagrams for the monitor pattern

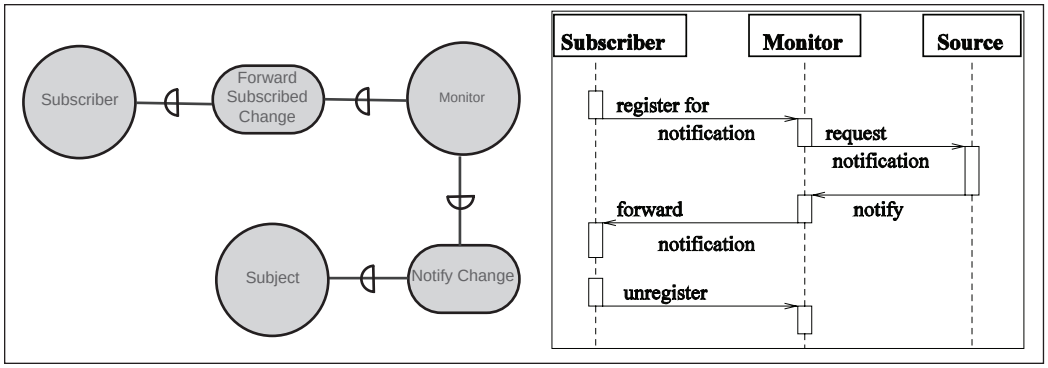


Figure 5. Social and communicational diagrams for the matchmaker pattern

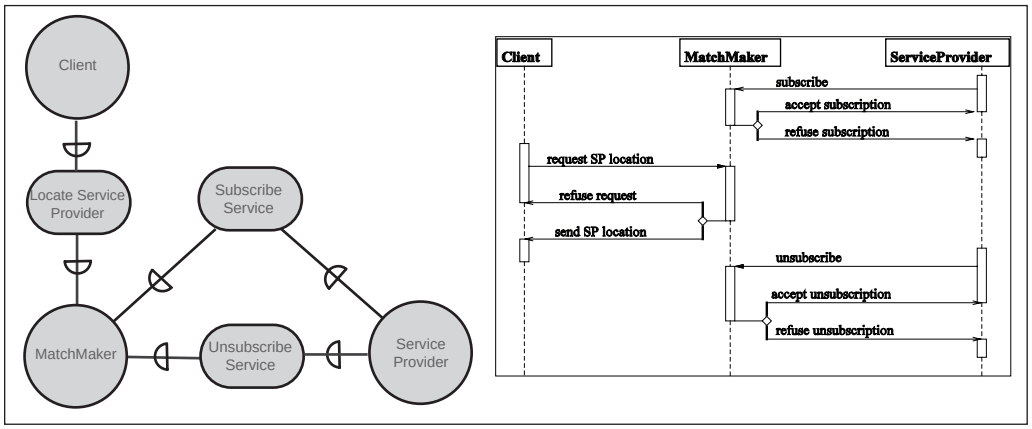


Figure 6. Social and communicational diagrams for the mediator pattern

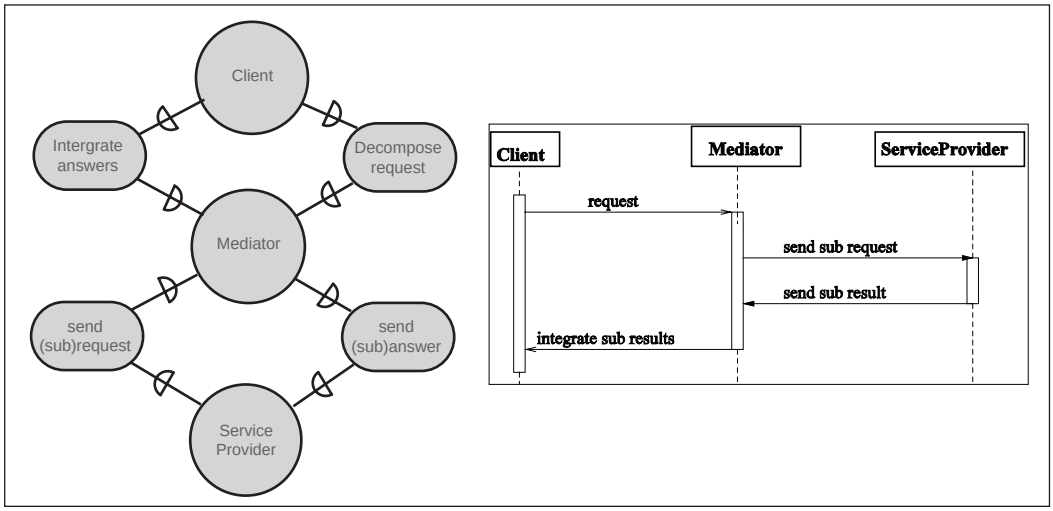
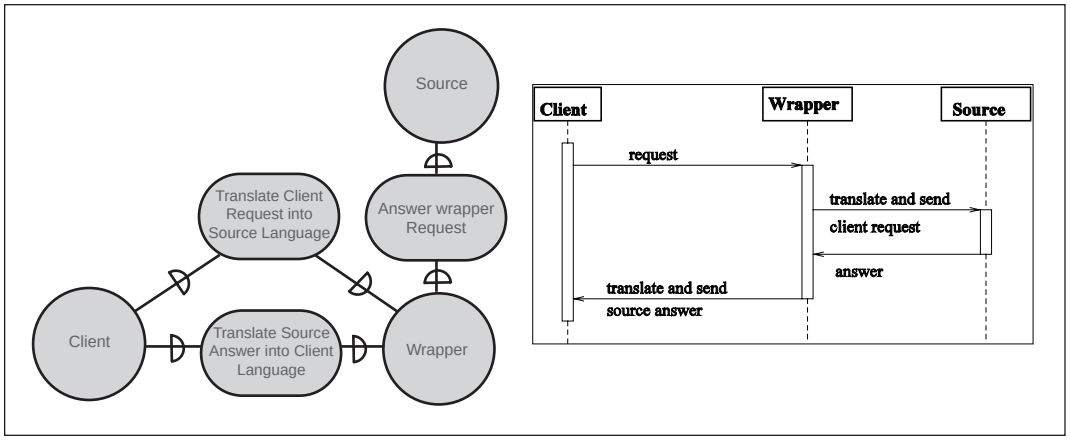


Figure 7. Social and communicational diagrams for the wrapper pattern



While a matchmaker simply matches providers with clients, a mediator encapsulates interactions and maintains models of the capabilities of clients and providers over time.

In the **embassy** pattern, an embassy agent routes a service requested by an external agent to a local agent. If the request is granted, the external agent can submit messages to the embassy for translation in accordance with a standard ontology. Translated messages are forwarded to the requested local agent and the result of the query is passed back out through the embassy to the external agent.

The **wrapper** pattern (Figure 7) incorporates a legacy system into a multi-agent system. A wrapper agent interfaces system agents with the legacy system by acting as a translator. This ensures that communication protocols are respected and the legacy system remains decoupled from the rest of the agent system.

A SOCIAL PATTERNS FRAMEWORK

This section describes a conceptual framework based on five complementary modeling dimensions, to investigate social patterns. The framework has been applied in the context of the Tropos development methodology (Castro et al., 2002). Each dimension reflects a particular aspect of a MAS architecture, as follows.:

- The *social dimension* identifies the relevant agents in the system and their intentional interdependencies.
- The *intentional dimension* identifies and formalizes services provided by agents to realize the intentions identified by the social dimension, independently of the plans that implement those services. This dimension answers the question: "What does each service do?"
- The *structural dimension* operationalizes the services identified by the intentional dimension in terms of agent-oriented concepts like beliefs, events, plans, and their relationships. This dimension answers the question: "How is each service operationalized?"
- The *communicational dimension* models the temporal exchange of events between agents.
- The *dynamic dimension* models the synchronization mechanisms between events and plans.

The social and the intentional dimensions are specific to MAS. The last three dimensions (structural, communicational, and dynamic) of the architecture are also relevant for traditional (non-agent) systems, but we have adapted and extended them with agent-oriented concepts. They are for instance the modeling dimensions used in object-oriented visual modeling languages such as UML.

The rest of this section details the five dimensions of the framework and illustrates them through the broker pattern (Yu, 1995).

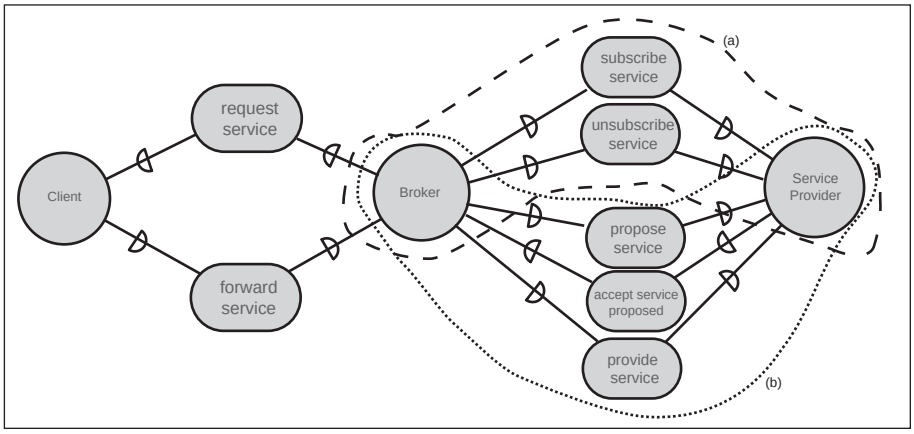
This pattern involves an arbiter intermediary that requests services from providers to satisfy the request of clients. It is designed through the framework as follows.

Social Dimension

The social dimension specifies a number of agents and their intentional interdependencies using the i* model (Do et al., 2003). Figure 8 shows a social diagram for the broker pattern.

The broker pattern can be considered as a combination of (1) a subscription pattern (shown enclosed within dashed boundary (a)), that allows service providers to subscribe their services to the broker agent and where the broker agent plays the role of a yellow-page agent, (2) one of the other pair patterns—booking, call-for-proposals,

Figure 8. Social diagram for the broker pattern



or bidding—whereby the broker agent requests and receives services from service providers (in Figure 13, it is a call-for-proposals pattern, shown enclosed within dotted boundary (b)), and (3) interaction between the broker and the client: the broker agent depends on the client for sending a service request and the client depends on the broker agent to forward the service.

To formalize intentional interdependencies, we use formal Tropos (Fuxman, Pistore, Mylopoulos, & Traverso, 2001), a first-order temporal-logic language that provides a textual notation for i* models and allows to describe dynamic constraints. A *forward service* dependency can be defined in formal Tropos as follows:

Dependum Forward Service
Mode: Achieve
Depender: Client *cl*
Dependee: Broker *br*
Fulfillment:
($\forall$ *sr*: ServiceRequest, *st*: ServiceType)
request(*cl, br, sr*) $\wedge$ provide(*br, st*) $\wedge$ ofType(*sr, st*)
$\rightarrow \Diamond$ received(*cl, br, st*)
[Broker *br* successfully provides its service to client *cl* if all requests *sr* from *cl* to *br*, that are of a type *st* that *br* can handle, are eventually satisfied]

Intentional Dimension

While the social dimension focuses on interdependencies between agents, the intentional dimension aims at modeling agent rationale. It is concerned with the identification of *services* provided by agents and made available to achieve the intentions identified in the social dimension. Each service belongs to one agent. Service definitions can be formalized by its fulfillment condition.

Table 1 lists several services of the broker pattern with an informal definition. With the `FindBroker` service, a client finds a broker that can handle a given service request. The request is then sent to the broker through the `SendServiceRequest` service. The broker can query its belief knowledge with the `QuerySPAvailability` service and answer the client through the `SendServiceRequestDecision` service. If the answer is negative, the client records it with its `RecordBRRefusal` service. If the answer is positive, the broker records the request (`RecordClientServiceRequest` service) and then broadcasts a call (`CallForProposals` service) to potential service providers. The client records

Table 1. Some services of the broker pattern

Service Name	Informal Definition	Agent
FindBroker	Find a broker that can provide a service	Client
SendServiceRequest	Send a service request to a broker	Client
QuerySPAvailability	Query the knowledge for information about the availability of the requested service	Broker
SendService RequestDecision	Send an answer to the client	Broker
RecordBRRefusal	Record a negative answer from a broker	Client
RecordBRAcceptance	Record a positive answer from a broker	Client
RecordClient ServiceRequest	Record a service request received from a Client	Broker
CallForProposals	Send a call for proposals to service providers	Broker
RecordAndSend SPInformDone	Record a service received from a service provider	Broker

acceptance by the broker with the `RecordBRAcceptance` service.

The call-for-proposals pattern could be used here, but this presentation omits it for brevity.

The broker then selects one of the service providers among those that offer the requested service. If the selected provider successfully returns the requested service, it informs the broker, which records the information and forwards it to the client (`RecordAndSendSPInformDone` service).

Services can be formalized in formal Tropos as illustrated below for the FindBroker service.

Service FindBroker (*sr*: ServiceRequest)
Mode: Achieve
Agent: Client *cl*
Fulfillment:
($\exists$ *br* : Broker, *st*: ServiceType)
provide(*br, st*) $\wedge$ ofType (*sr, st*)
$\rightarrow \Diamond$ known(*cl, br*)
[*FindBroker* is fulfilled when client *cl* has found (*known* predicate) *Broker br* that is able to perform (*provide* predicate) the service requested.]

Structural Dimension

While the intentional dimension answers the question "What does each service do?", the structural dimension answers the question "How is each service operationalized?" Services are operationalized as *plans*, that is, sequences of actions.

The knowledge that an agent has (about itself or its environment) is stored in its *beliefs*. An agent can act in response to the *events* that it handles through its plans. A plan, in turn, is used by the agent to read or modify its beliefs, and send events to other agents or post events to itself.

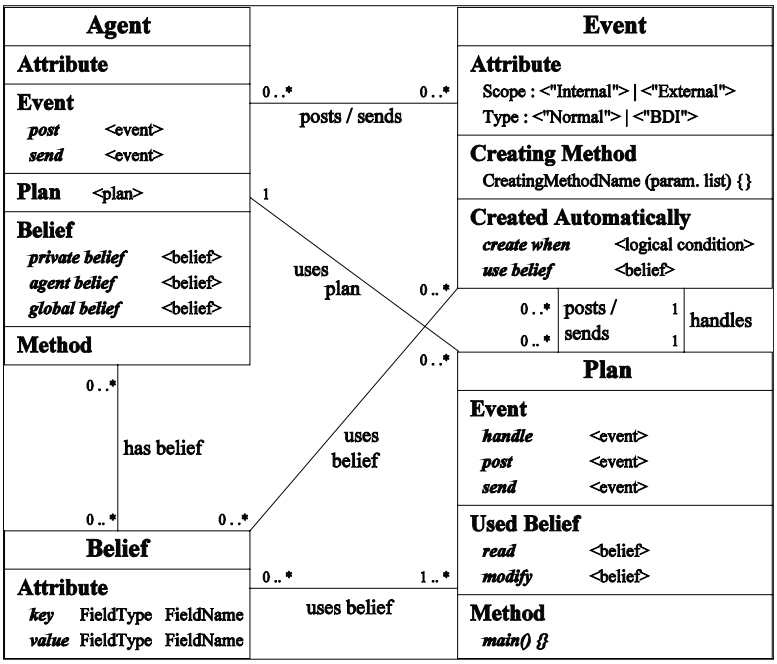
The structural dimension is modeled using a UML style class diagram extended for MAS engineering.

The required agent concepts extending the class diagram model are defined in the following.

Structural Concepts

Figure 9 depicts the concepts and their relationships needed to build the structural dimension.

Figure 9. Structural diagram template



Each concept defines a common template for classes of concrete MAS (for example, agent in Figure 9 is a template for the broker agent class of Figure 10).

A **belief** describes a piece of the knowledge that an agent has about itself and its environment. Beliefs are represented as tuples composed of a key and value fields.

Events describe stimuli, emitted by agents or automatically generated, in response to which the agents must take action. As shown in Figure 9, the structure of an event is composed of three parts: declaration of the attributes of the event, declaration of the methods to create the event, declaration of the beliefs and the condition used for an automatic event. The third part only appears for automatic events. Events can be described along three dimensions:

- *External or internal* event: external events are sent to other agents while internal events are posted by an agent to itself. This property is captured by the *scope* attribute.
- *Normal or BDI* event: an agent has a number of alternative plans to respond to a BDI (belief-desire-event) event and only one plan in response to a normal event. Whenever an event occurs, the agent initiates a plan to handle it. If the plan execution fails and if the event is a normal event, then the event is said to have failed. If the event is a BDI event, a set of plans can be selected for execution and these are attempted in turn. If all selected plans fail, the event is also said to have failed. The event type is captured by the *type* attribute.
- *Automatic or nonautomatic* event: an automatic event is automatically created when certain belief states arise. The *create when* statement specifies the logical condition which must arise for the event to be automatically created. The states of the beliefs

that are defined by *use belief* are monitored to determine when to automatically create events.

A **plan** describes a sequence of actions that an agent can take when an event occurs. As shown by Figure 9, plans are structured in three parts: the event part, the belief part, and the method part. The Event part declares events that the plan `handles` (i.e., events that trigger the execution of the plan) and events that the plan produces. The latter can be either `posted` (i.e., sent by an agent only to itself) or `sent` (i.e., sent to other agents). The belief part declares beliefs that the plan `reads` and those that it `modifies`. The method part describes the plan itself, that is, the actions performed when the plan is executed.

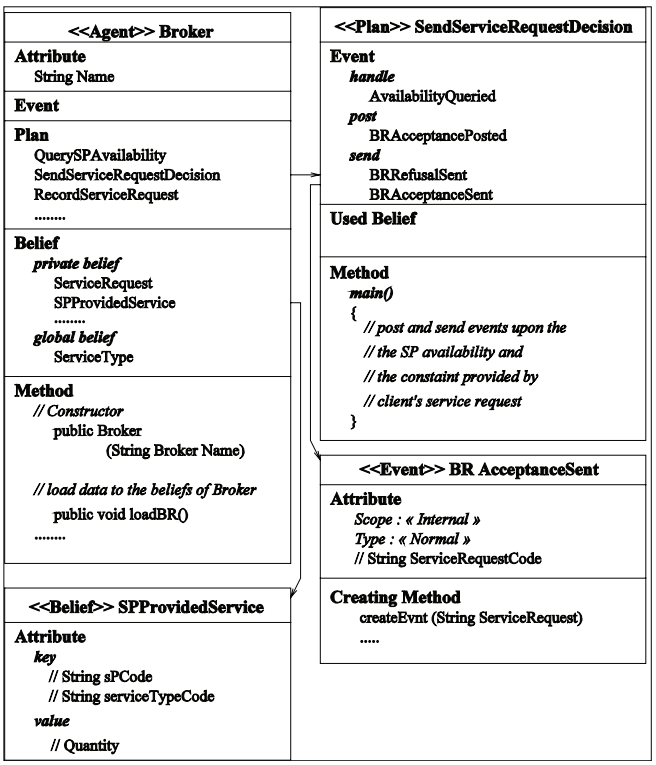
The **agent** concept defines the behavior of an agent, as composed of five parts: the declaration of its attributes, of the events that it can post or send explicitly (i.e., without using its plans), of the plans that it uses to respond to events, of the beliefs that make up its knowledge, and of its methods.

The beliefs of an agent can be of type *private*, *agent*, or *global*. A *private* access is restricted to the agent to which the belief belongs. *Agent* access is shared with other agents of the same class, while *global* access is unrestricted.

Structural Model for the Broker Pattern

Figure 10 depicts the broker pattern components. For brevity, each construct described earlier is illustrated only through one component. Each component can be considered as an instantiation of the (corresponding) template in Figure 9.

Figure 10. Structural diagram—some components of the broker pattern



`Broker` is one of the three agents composing the broker pattern. It has plans such as `QuerySPAvailability`, `SendServiceRequestDecision`, and so forth. When there is no ambiguity, by convention, the plan name is the same as the as the name of the service that it operationalizes. The private belief `SPProvidedService` stores the service type that each service provider can provide. This belief is declared as private since the broker is the only agent that can manipulate it. The `ServiceType` belief stores the information about types of service provided by service providers and is declared as global since its must be known both by the service provider and the broker agent.

The constructor *method* allows a name to be given to a broker agent when created. This method may call other methods, for example `loadBR()`, to initialize agent beliefs.

`SendServiceRequestDecision` is one of the plans that the broker uses to answer the client: the `BRRefusalSent` event is sent when the answer is negative, `BRAcceptanceSent` when the broker has found service provider(s) that may provide the requested service. In the latter case, the plan also posts the `BRAcceptancePosted` event to invoke the process of recording the service request and the 'call for proposals' process between the broker and services providers. The `SendServiceRequestDecision` plan is executed when the `AvailabilityQueried` event (containing the information about the availability of the service provider to realize the client's request) occurs.

`SPProvidedService` is one of the broker's beliefs used to store the services provided by the service providers. The service provider code `sPCode` and the service type code `serviceTypeCode` form the belief key. The corresponding `quantity` attribute is declared as value field.

`BRAcceptanceSent` is an event that is sent to inform the client that its request is accepted.

At a lower level, each plan could also be modeled by an activity diagram for further detail if necessary.

Communication Dimension

Agents interact with each other by exchanging events. The communicational dimension models, in a temporal manner, events exchanged in the system. We adopt the sequence diagram model proposed in AUML (Bauer et al., 2001) and extend it: *agent_name/role:pattern_name* expresses the role (*role*) of the agent (*agent_name*) in the pattern; the arrows are labeled with the name of the exchanged events.

Figure 11 shows a sequence diagram for the broker pattern. The client (`customer1`) sends a service request (`ServiceRequestSent`) containing the characteristics of the service it wishes to obtain from the broker. The broker may alternatively answer with a denial (`BRRefusalSent`) or an acceptance (`BRAcceptanceSent`).

In the case of an acceptance, the broker sends a call for proposal to the registered service providers (`CallForProposalSent`). The call for proposal (CFP) pattern is then applied to model the interaction between the broker and the service providers. The service provider either fails or achieves the requested service. The broker then informs the client about this result by sending an `InformFailureServiceRequestSent` or a `ServiceForwarded`, respectively.

The communication dimension of the subscription pattern (SB) is given at the top-right and the communication dimension of the call-for- proposals pattern (CFP) is given at the bottom-right part of Figure 11. The communication specific for the broker pattern is given in the left part of the figure.

Dynamic Dimension

As described earlier, a plan can be invoked by an event that it handles and it can create new events. Relationships between plans and events can become complex rapidly. To cope with this problem, we propose to model the synchronization and the relationships between plans and events

Figure 11. Communication diagram—broker

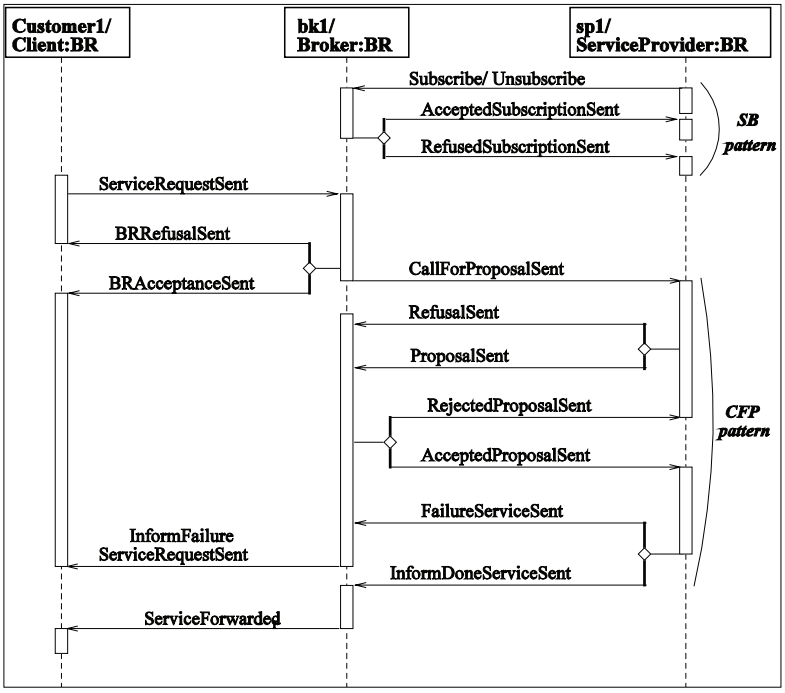
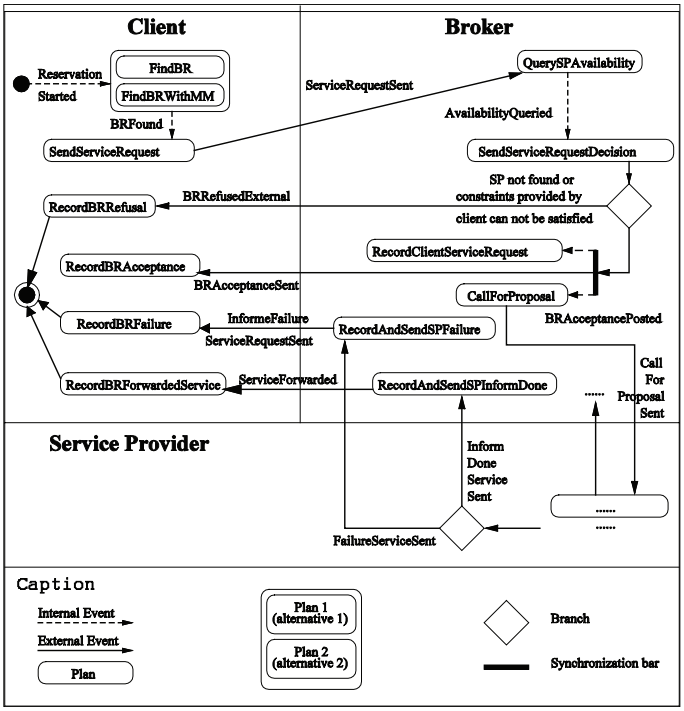


Figure 12. Dynamic diagram—broker



with activity diagrams extended for agent-oriented systems. These diagrams specify the events that are created in parallel, the conditions under which events are created, which plans handle which events, and so on.

An internal event is represented by a dashed arrow and an external event by a solid arrow. As mentioned earlier, a BDI event may be handled by alternative plans. They are enclosed in a round-corner box. Synchronization and branching are represented as usual.

We omit the dynamic dimension of the subscription and the CFP patterns, and only present in Figure 12 the activity diagram specific to the broker pattern. It models the flow of control from the emission of a service request sent by the client to the reception by the same client of the realized service result sent by the broker. Three swimlanes, one for each agent of the broker pattern, compose the diagram. In this pattern, the `FindBroker` service described in Section 3.2.2, is either operationalized by the `FindBR` or the `FindBRWithMM` plans (the client finds a broker based on its own knowledge or via a matchmaker).

FROM ORGANIZATIONAL ARCHITECTURAL STYLES TO SOCIAL DESIGN PATTERNS

A key aspect to conduct MAS architectural design is the specification and use of *organizational styles* (Castro et al., 2002; Do, Faulkner, & Kolp, 2003; Kolp et al., 2002) that are socially-based architectural designs inspired from models and concepts from organization theory (e.g., Mintzberg, 1992; Scott, 1998; Yoshino & Srinivasa Rangan, 1995) and strategic alliances (e.g., Dussauge & Garrette, 1999; Morabito, Sack & Bhate, 1999; Segil, 1996) that analyze the structure and design of real-world human organization. These are for instance the structure-in-fives, the matrix, the joint-venture, the hierarchical contracting, and so forth.

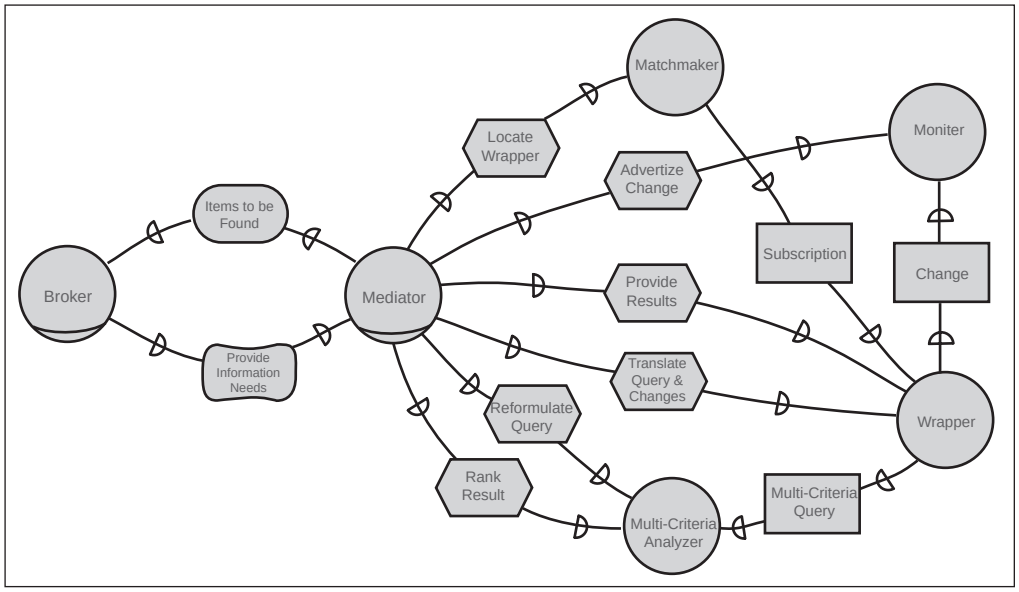
As described in (Castro et al., 2002; Zambonelli et al., 2000), in MAS architectural design, organizational styles are used to give information about the system architecture to be: every time an organizational style is applied, it allows to easily point up, to the designer, the required organizational actors and roles. Then the next step needs to detail and relate such (organizational) actors and roles to more specific agents in order to proceed with the agent behavior characterization. Namely, each actor in an organization-based architecture is much closer to the real world system actor behavior that we consequently aim to have in software agents. As a consequence, once the organizational architectural reflection has figured out the MAS global structure in terms of actors, roles, and their intentional relationships, a deepener analysis is required to detail the agent behaviors and their interdependencies necessary to accomplish their roles in the software organization. To effectively deal with such a purpose, developers can be guided by social patterns proposed in this article.

Social patterns offer a microscopic view of the MAS at the *detailed design* phase to express in deeper detail organizational styles during the architectural design. To explain the necessary relationship between *styles* and *patterns* we consider an original *data integrator* case study and overview how a MAS designed from some style at the architectural level is decomposed into social patterns at the detailed design level.

The **data integrator** allows users to obtain information that come from different heterogeneous and distributed sources. Sources range from text file systems agent knowledge bases. Information from each source that may be of interest is extracted, translated and filtered as appropriate, merged with relevant information from other sources to provide the answer to the users' queries (Widom, 1995).

Figure 13 shows an MAS architecture in i* for the data integrator that applies the *joint-venture* style (Castro et al., 2002; Do et al., 2003) at the

Figure 13. A joint-venture MAS architecture expressed in terms of social patterns—A data integration example



architectural design level. In a few words, the joint venture organizational style is a meta-structure that defines an organizational system that involves agreement between two or more independent partners to obtain the benefits of larger scale, shared investment and lower maintenance costs. A specific joint management actor coordinates tasks and manages the sharing of resources between partner actors. Each partner can manage and control itself on a local dimension and may interact directly with other partners to exchange resources, such as data and knowledge. However, the strategic operation and coordination of such a system, and its actors on a global dimension, are the only responsibility of the joint management actor in which the original actors possess equity participations.

Joint-venture's roles at the architectural design level are expressed in the detailed design level in terms of patterns, namely the broker, the matchmaker, the monitor, the mediator, and the wrapper. The *joint management private interface* is assumed by a mediator, the joint-venture partners are the *wrapper*, the *monitor*, the *multi-criteria analyzer* and the *matchmaker*. The *public interface* is assumed by the *broker*.

The system works as follows. When a user wishes to send a request, she contacts the *broker* agent which is an intermediary to select one or many *mediator(s)* that can satisfy the user information needs. Then, the selected mediator(s) decomposes the user's query into one or more subqueries to the sources, synthesizes the source answers and return the answers to the broker.

If the mediator identifies a recurrent user information request, the information that may be of interest is extracted from each source, merged with relevant information from other sources, and stored as knowledge by the mediator. This stored information constitutes a materialized view that the mediator will have to maintain up-to-date.

A *wrapper* and a *monitor* agent are connected to each information source. The *wrapper* is responsible for translating the subquery issued by

the mediator into the native format of the source and translating the source response in the data model used by the mediator.

The *monitor* is responsible for detecting changes of interest (e.g., change which affects a materialized view) in the information source and reporting them to the mediator. Changes are then translated by the wrapper and sent to the mediator.

It may be also necessary for the mediator to obtain the information concerning the localization of a source and its connected wrapper that are able to provide current or future relevant information. This kind of information is provided by the *matchmaker* agent which then lets the mediator interacts directly with the correspondent wrapper.

Finally, the *multi-criteria analyzer* can reformulate a subquery (sent by a mediator to a wrapper) through a set of criteria in order to express the user preferences in a more detailed way, and refine the possible domain of results.

AUTOMATION

The main motivation behind design patterns is the possibility of reusing them during system detailed design and implementation. Numerous CASE tools such as Rational Rose (IBM Rational Rose, 2007) and Together (Borland Together, 2007) include code generators for object-oriented design patterns. Programmers identify and parameterize, during system detailed design, the patterns that they use in their applications. The code skeleton for the patterns is then automatically generated and programming is thus made easier.

For agent-oriented programming, SKwyRL (Do et al., 2003), for instance, proposes a code generator to automate the use of social patterns introduced in section 2. Figure 13 shows the main window of the tool. It has been developed in Java and produces code for JACK (JACK Intelligent Agents, 2006), an agent-oriented development environment built on top of Java. JACK extends

Figure 14. JACK code generation

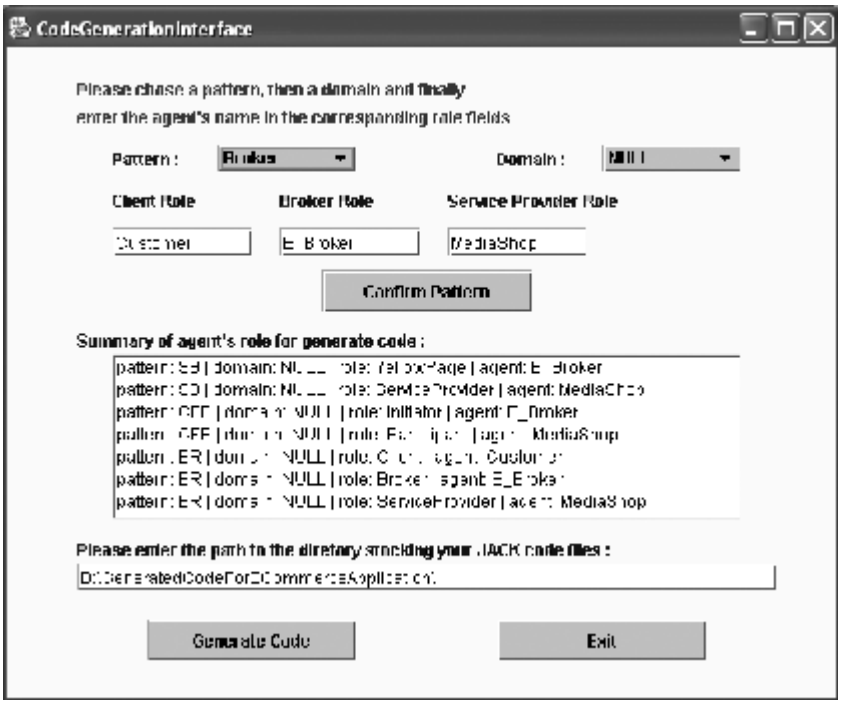
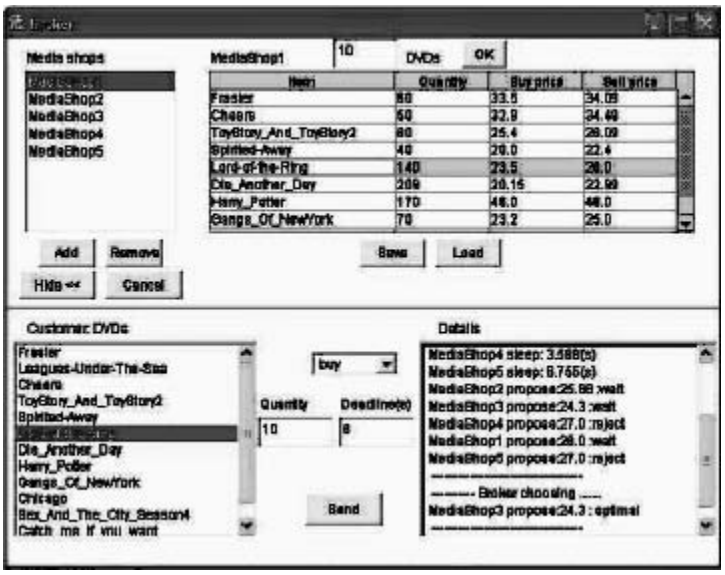


Figure 15. An e-business broker



Java with specific capabilities to implement agent behaviors. On a conceptual point of view, the relationship of JACK to Java is analogous to that between C++ and C. On a technical point of view, JACK source code is first compiled into regular Java code before being executed.

In SKwyRL's code generator, the programmer first chooses which social pattern to use, then the roles for each agent in the selected pattern (e.g., the `E_Broker` agent plays the *broker* role for the Broker pattern but can also play the *initiator* role for the CallForProposals pattern and the *yellow page* role for the subscription pattern in the same application). The process is repeated until all relevant patterns have been identified. The code generator then produces the generic code for the patterns (.agent, .event, .plan, .bel JACK files).

The programmer has to add the particular JACK code for each generated files and implement the graphical interface if necessary.

Figure 15 shows an example of the (e-business) broker for the data integrator presented in section 4. It was developed with JACK and the code skeleton was generated with SKwyRL's code generator using the broker pattern explained in the article. The bottom half of the figure shows the interface between the customer and the broker. The customer sends a service request to the broker asking for buying or sending DVDs. He chooses which DVDs to sell or buy, selects the corresponding DVD titles, the quantity and the deadline (the time-out before which the broker has to realizes the requested service). When receiving the customer's request, the broker interacts with the media shops to obtain the DVDs. The interactions between the broker and the media shops are shown on the bottom-right corner of this figure. The top half of the figure shows the items that are provided by each media shop.

RELATED WORK

As already said, a lot of work has been devoted to software patterns these last 15years. Patterns for software development are one of software engineering problem-solving discipline that has its roots in a design movement in contemporary

architecture and the documentation of best practices and lessons learned in all vocations. The goal of patterns is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development. Patterns help create a shared language for communicating insight and experience about these problems and their solutions.

Ward Cunningham and Kent Beck developed a small set of five patterns (Beck & Cunningham, 1987), for guiding Smalltalk programmers to design user interface. Jim Coplien introduced a catalog of for C++ patterns, called *idioms* (Coplien, 1991). Software patterns then became popular with the wide acceptance of the Gang of Four or GoF (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) book (Gamma et al., 1995).

However, the patterns in the GoF book are only one kind of pattern—the object-oriented *design patterns*. There are many other kinds of patterns. For example, Martin Fowler's "*Analysis Patterns*" (Fowler, 1997) describe the models of business processes that occur repeatedly in the analysis phase of software development; *organizational patterns* (Coplien & Schmidt, 1995) are about software-development organizations and about people who work in such organizations. *Process patterns* (Ambler, 1998) relate to the strategies that software professionals employ to solve problems that recur across organizations. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, helped popularize these kinds of patterns (organizational and process patterns) (Buschmann et al., 1996).

Contrary to pattern, that represents a "best practice," an *anti-pattern* represents a "lesson learned." There are two kinds of "anti-patterns:" those that describe a bad solution to a problem which resulted in a bad situation and those that describe how to get out of a bad situation and how to proceed from there to a good solution. Anti-pattern is initially proposed by Andrew Koenig. Anti-patterns extend the field of software patterns research into exciting new areas and issues, including: refactoring, reengineering, system extension, and system migration (Brown, Malveau, Hays, et al., 1998; Love, 1997; Opdyke, 1992; Webster, 1995).

Recent popularity of autonomous agents and agent-oriented software engineering has led to the discovery of agent patterns (Aridor & Lange, 1998;, Deugo et al., 1999; Do et al., 2003; Hayden et al., 1999; Mouratidis, Giorgini & Manson, 2003), that capture good solutions to common problems in agent design in many aspect such as security, architecture, organization, and so forth. However, as pointed out earlier, little focus has been put on social and intentional considerations and these agent patterns rather aim at the implementation level. The framework presented in the chapter should add more detail to the design process of agent oriented software engineering (Do et al., 2003, Do et al., 2003).

CONCLUSION

Nowadays, software engineering for new enterprise application domains such as e-business, knowledge management, peer-to-peer computing, or Web services is forced to build up open systems able to cope with distributed, heterogeneous, and dynamic information issues. Most of these software systems exist in a changing organizational and operational environment where new components can be added, modified or removed at any time. For these reasons and more, multi-agent systems (MAS) architectures are gaining popularity in that they do allow dynamic and evolving structures which can change at run-time.

An important technique that helps to manage the complexity of such architectures is the reuse of development experience and know-how. Like any architect, software architects use patterns to guide system development. Over the years, patterns have become an attractive approach to reusing architectural design knowledge in software engineering. Patterns describe a problem com-

monly found in software designs and prescribe a flexible solution for the problem, so as to ease the reuse of that solution.

As explored in this article, MAS architectures can be considered social structures composed of autonomous and proactive agents that interact and cooperate with each other to achieve common or private goals. Since the fundamental concepts of multi-agent systems are intentional and social, rather than implementation-oriented, social abstractions could provide inspiration and insights to define patterns for designing MAS architectures.

This article has focused on social patterns. With real-world social behaviors as a metaphor, social patterns are agent-oriented design patterns that describe MAS as composed of autonomous agents that interact and coordinate to achieve their intentions, like actors in human organizations.

The article has described such patterns, a design framework to introspect them and formalize their "code of ethics," answering the question: what can one expect from a broker, mediator, embassy, and so forth. It aims to be used during the detail design phase of any agent-oriented methodology detailing the patterns following different point of views.

REFERENCES

Ambler, S. (1998). *Process patterns: Building large-scale systems using object technology.* Cambridge: Cambridge University Press.

Aridor, Y. & Lange, D. B. (1998). Agent design patterns: Elements of agent application design. *In Proc. of the 2nd Int. Conf. on Autonomous Agents, Agents'98, (pp.108-115).* Minneapolis, USA

Bauer, B., Muller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multiagent interaction. *In Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering, AOSE'00,* (pp.91-103). Limerick, Ireland,

Beck, K., & Cunningham, W. (1987). Using pattern languages for object-oriented programs. *workshop on the Specification and Design for Object-Oriented Programming (OOPSLA'87).*

Borland Together. (2007). Retrieved August 16, 2007 from Available at: http://www.borland.com/downloads/download_together.html

Bosch, J. (1998). Design patterns as language constructs. *Journal of Object-Oriented Programming, 11*(2), 18-32.

Brown, W. J., Malveau R. C., Hays, W., McCormick Iii, H., & Mowbray, T. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis.* John Wiley & Sons.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture—a system of patterns.* John Wiley & Sons.

Caire J. & al. (2002). Agent-oriented analysis using MESSAGE/UML. *In Proceedings of the 2nd Int. Workshop on Agent-Oriented Software Engineering, LNCS* 2222, (pp. 119-135).

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: The tropos project. *Information Systems, 27*(6), 365-389.

Chidamber, & S.R, Kemerer, C.F. (1994). A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476-493.

Cockburn, A. (1996). The interaction of social issues and software architecture. *Communication of the ACM, 39*(10), 40-49.

Coplien, J. & Schmidt D. (1995). *Pattern languages of program design.* Addison-Wesley.

Coplien, O. (1991). *Advanced C++ programming styles and idioms.* Addison-Wesley International.

Deugo, D., Oppacher, F., Kuester, J., & Otte, I. V. (1999). Patterns as a means for intelligent software engineering. *In Proceedings of the Int. Conf. on Artificial Intelligence*, IC-AI'99, Las Vegas, Nevada, USA, (pp. 605-611).

Do, T. T., Faulkner, S., & Kolp, M. (2003). Organizational multi-agent architectures for information systems. *In Proceedings of the 5th International Conference on Enterprise Information Systems, ICEIS 2003,* (pp. 89-96).

Do, T. T., Kolp, M., Hang Hoang T. T., & Pirotte, A. (2003). A framework for design patterns for Tropos. *In Proceedings of the 17th Brazilian Symposium on Software Engineering, SBES 2003.*

Dussauge, P., & Garrette, B. (1999*). Cooperative strategy: Competing successfully through strategic Alliances.* Wiley and Sons.

Fernandez, E. B., & Pan, R. (2001). A pattern language for security models. *In Proceedings of the 8th Conference on Pattern Language of Programs, PLoP 2001.*

FIPA. (2007). *The foundation for intelligent physical agent (FIPA).* http://www.fipa.org/

Fowler, M. (1997). *Analysis patterns: Reusable object models.* Addison-Wesley.

Fuxman, A., Pistore, M., Mylopoulos, J., & Traverso, P. (2001). Model checking early requirements specifications in Tropos. *In Proc. of the 5th IEEE Int. Symposium on Requirements Engineering, RE'01,* (pp. 174-181).

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Hayden, S., Carrick, C., & Yang, Q. (1999). Architectural design patterns for multiagent coordination. *In Proceedings of the 3rd Int. Conf. on Agent Systems, Agents'99,* Seattle, USA.

IBM Rational Rose. (2007). Retrieved August 16, 2007 from http://www-306.ibm.com/software/rational

JACK Intelligent Agents. (2006). Retrieved August 16, 2007 from http://www.agent-software.com

Jennings, N. R. & Wooldridge, M. (2001). Agent-oriented software engineering. In Handbook of Agent Technology, AAAI/ MIT Press.

Kolp, M., Faulkner, S., & Wautelet, Y. (2007). Social-centric design of multi-agent architectures. In P. Giorgini, N. Maiden, J. Mylopoulos, E. Yu (Eds.), Social modeling for requirements engineering, in the Cooperative Information Systems series. MIT Press.

Kolp, M., Giorgini, P., & Mylopoulos, J. (2002). Information systems development through social structures. *In Proceedings of the 14th Int. Conference on Software Engineering and Knowledge Engineering, SEKE'02,* Vol. 27, (pp. 183-190).

Love, T. (1997). *Object lessons.* Cambridge University Press.

Mintzberg, H. (1992). *Structure in fives: Designing effective organizations.* Prentice-Hall.

Morabito, J., Sack, I., & Bhate, A. (1999). *Organization modeling: Innovative architectures for the 21st century.* Prentice Hall.

Mouratidis H., Giorgini P., & Manson G. (2003). Modeling secure multiagent systems. *In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, ACM press,* (pp. 859-866).

OMG. (2005). The software process engineering Mmtamodel Specification. Version 1.1.

Opdyke, W. F. (1992). Refactoring object-oriented frameworks. PhD Thesis, University of Illinois at Urbana-Champaign.

Parunak, V. (1997). Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research, 75,* 69-101.

Pree, W. (1994). Design patterns for object oriented development. Addison Wesley.

Riehle, D. & Züllighoven, H. (1996). Understanding and using patterns in software development. *Theory and Practice of Object Systems, 2*(1), 3-13.

Scott, W.R. (1998). *Organizations: Rational, natural, and open systems.* Prentice Hall.

Segil, L. (1996). Intelligent business alliances: How to profit using today's most important strategic tool. Times Business.

Wautelet, Y., Kolp, M. & Achbany Y. (2006). I-Tropos: An Iterative SPEM-Centric Software Project Management Process. Technical Report IAG Working paper 06/01, IAG/ISYS Information Systems Research Unit, Catholic University of Louvain, Belgium. http://www.iag.ucl.ac.be/wp/

Webster, B. F. (1995). *Pitfalls of object oriented development.* John Wiley & Sons Inc.

Widom, J. (1995). Research problems in data warehousing. *In Proceedings of the Fourth Int. Conf. on Information and Knowledge Management,* ACM Press, (pp. 25-30).

Wood, M., DeLoach, S. A., & Sparkman C. (2001). Multi-agent system engineering. *International Journal of Software Engineering and Knowledge Engineering, 11*(3), 231-258.

Woodridge, M., Jennings, N. R., & Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems, 3*(3), 285-312.

Yoshino, M.Y., & Srinivasa Rangan, U. (1995). *Strategic alliances: An entrepreneurial approach to globalization.* Harvard Business School Press.

Yu, E. (1995). Modeling strategic relationships for process reengineering. PhD thesis, University of Toronto, Department of Computer Science.

Zambonelli, F., Jennings, N.R., Omicini, A., & Wooldridge, M. (2000). Agent-oriented software engineering for internet applications. In *coordination of internet agents: Models, technologies and applications* (pp. 326-346). Springer Verlag.

Zambonelli, F., Jennings, N.R., & Wooldridge, M. (2000). Organizational abstractions for the analysis and design of multi-agent systems. *In Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering,* (pp. 243-252).

This work was previously published in International Journal of Intelligent Information Technologies, Vol. 4, Issue 2, edited by V. Sugumaran, pp. 1-23, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 2.16
Women in the Free/Libre Open Source Software Development

Yuwei Lin
Vrije Universiteit Amsterdam, The Netherlands

INTRODUCTION

Free/libre open source software (FLOSS) has become a prominent phenomenon in the ICT field and the wider public domain for the past years. However, according to a FLOSS survey on FLOSS developers in 2002, "women do not play a role in the [FLOSS] development; only 1.1% of the FLOSS sample is female." (Ghosh, Glott, Krieger, & Robles, 2002). In the mainstream research on FLOSS communities, many researchers also overlook different processes of community-building and diverse experiences of members, and presume a stereotyped male-dominated "hacker community" (e.g., Levy, 1984; Raymond, 2001; Himanen, 2001; Thomas, 2002). Moreover, issues around gender inequality are often ignored and/or muted in the pile of FLOSS studies. Female programmers often are rejected ex/implicitly from the software labour market (Levesque & Wilson 2004). The requirements of female users are not respected and consulted either (European Commission, 2001). This feature is opposite to the FLOSS ideal world where users should be equally treated and embraced (op. cit.). While many researchers endeavour to understand the FLOSS development, few found a gender-biased situation problematic. In short, women are almost invisible in current FLOSS-related literature. Most policies targeting at advocating FLOSS are also gender blind.

Thus, this essay highlights the need for increased action to address imbalances between women's and men's access to and participation in the FLOSS development in cultural (e.g., chauvinistic and/or gender-biased languages in discussions on mailing lists or in documentations), economic (e.g., unequal salary levels for women and men), political (e.g., male-dominated advocacy environment) and technical (e.g., unbalanced students gender in technical tutorials) spheres. On the other hand, it also emphasises the powerful potential of FLOSS as a vehicle for advancing gender equality in software expertise. FLOSS helps transport knowledge and experience of software engineering through distributing

source code together with the binary code almost without any limit. Many FLOSS licences such as the General Public Licence (GPL) also facilitates the flow of information and knowledge. In other words, if appropriately harnessed, FLOSS stands to meaningfully contribute to and mutually reinforce the advancement of effective, more expedited solutions to bridging the gender digital divide.

In the end, this article points out that while women in more advanced countries have a better chance of upgrading their ICT skills and knowledge through participating in the FLOSS development, the opportunity is less available for women in the developing world. It is worth noting that although the gender issues raised in this article are widespread, they should not be considered as universally indifferent. Regional specificities in gender agenda in software engineering should be addressed distinctly (UNDP/UNIFEM, 2004).

TOWARD A FEMINIST ANALYTICS[1] ON THE GENDER ISSUES IN THE FLOSS DEVELOPMENT

To a degree, the gender problems in the FLOSS development can be seen as an extension of the ongoing gender issues in new-tech service industries and/or software industry (e.g., Mitter & Rowbotham, 1995). These long-term problems mainly include low-level work content, unequal payment, emotional distress from discrimination and prejudice, physical ache from the long working hour in front of the computer, division of labour within the home (child-rearing), essentialist notions of women's roles, sexism, informal networks, prejudice, lack of role models and support, and "glass ceilings." Generally speaking, women within the software industry have to work harder than men in order to get the same respect and conquer the glass-ceiling problem in this patriarchy world (DeBare, 1996).

Although FLOSS has dramatically changed the way software is produced, distributed, supported, and used, and has a visible social impact enabling a richer digital inclusion, most of the gender problems existing in the software industry have been duplicated in the FLOSS field.

A FLOSS *social world* (Lin, 2004) is different from what Turkle (1984) argues: "computer systems [mainly proprietary] represent a closed, controllable microworld—which appeals to more men than women" (Turkle, 1984). It requires a holistic perspective to capture the complexity and dynamics within and across the social world. While the heterogeneity and the contingency in this social world are not yet fully explored, analysis from a feminist perspective is almost absent. Little attention has been paid to the internal differences and to the private arena linked with the FLOSS innovation system. However, this methodological lack has not stopped us from observing the gender problems within the field. Instead, by means of the FLOSS development, some gender problems in ICT become even more apparent.

Additionally, in a world of volunteers, we clearly see that men and a competitive worldview are more present in all forms of media. Many women participating in the FLOSS development are invisible: their labour in fields such as NGOs that help implement and promote FLOSS, documentation translation, book editing, teaching and tutoring (e.g., E-Riders[2]) are less visible than male-dominated coding work. Indeed, FLOSS advocates have not adequately addressed this critique of gender equality. They tend to treat the FLOSS community as a monolithic culture—to pay more attention to differences between and among groups than to differences within them. They are so eager uniting the voices on freedom of information that they give little or no recognition to the fact that FLOSS groups, "like the societies in which they exist (though to a greater or lesser extent), are themselves *gendered*, with substantial differences of power and advantage between men and women" (Okin, 1999).

A number of key dilemmas that hinder women's participation in the FLOSS development can be summarised:

1. **A Lack of "Mentors" and Role Models:** It is true that there is a very low percentage of female participants in the FLOSS social world. However, we should not overlook the importance and possible future of outstanding female figures in the FLOSS field. It is difficult to make the majority of male peers respect these female figures. I am not suggesting that men all look down on women, but it is more difficult for women to be assertive in front of male-dominated audience. The whole way the world is constructed means there are just men at every level, which makes it really hard for women to get their feet in the door. A way of overcoming this is to establish more female figures in the world. While few in the computer world actually know that Ms. Ada Byron is the first programmer in the world, how could we expect people to recognise women's ability?
2. **Discriminated Languages Online and/or Offline (e.g., Phrases in Documentaries):** Many female FLOSS developers have complained the highly unfriendly atmosphere within the social world, online (e.g., mailing lists, IRC) and/or offline (e.g., documentation). For instance, referred to prospective readers, existed FLOSS documentation usually use single sex term, he, rather than she or they. This kind of gender-biased words subtly exclude women from participating in the FLOSS development. While the online languages are in a direct way full of men's jargon, reading the documentation offline does not make a female developer/user feel more included in the field. If women need to be encouraged to participate in FLOSS-related discussions, a sexist or discriminative surrounding is definitely not attractive.
3. **A Lack of Women-Centred View in the FLOSS Development:** The consequence of the lack of female FLOSS developers is that there is a greater amount of female-unfriendly software in the FLOSS system. Some scholars in science and technology studies (STS) have pointed out that technologies are gendered both in their design and use (e.g., Edwards, 1993; Wajcman, 2004).The social relations of gender within and across the FLOSS social world are reflected in and shaped by the design of FLOSS. And such a lack of women's perspective on software design and use restricts women's participation in the FLOSS development and, in turn, forms the stereotyped fact that women are almost absent in the FLOSS development because they are less adequate in programming or less likely to be advanced computer users. This absence of female developers would also be a loss of the FLOSS development, and result in inequalities in an ICT-based society as a threat to social cohesion and social order.
4. **A Male-Dominated Competitive Worldview:**
"[The FLOSS market] is literally a war for the best and brightest. If we don't get there, somebody else will" (Andrew Clark, director of strategy and market intelligence for the venture capital group at IBM—interviewed with C|Net.com on February 14, 2005).
As Arun and Arun (2001) point out, "The project-based, competitive nature of software development reproduces a masculine culture, which further interacts with the different career patterns of women and social norms and tends to disadvantage women." While languages in a similar tone with Clark's words above repeatedly turn up in the mass media such as advertisements from big computer companies, the male-led competitive worldview is continuously represented and reinforced in the society. Since there

are fairly clear disparities of power between the sexes within the FLOSS social world, a gender-imbalanced world is ensued. The more powerful male members are those who are generally in a position to determine and articulate the group's beliefs, practices, and interests. Although not all proposals associated with FLOSS are potentially antifeminist under such conditions, but they somehow duplicate and forward the view that might limit the capacities of women and girls to freely choose lives that they would like to live. It is very alarming that a large amount of perspectives and purposes regarding the FLOSS development is determined by white men. This imbalance might give a distorted world view; it is much better to have views from all people from different social worlds.

5. **No Sympathy from Women Peers:** There are many more spoken or unspoken problems for women to take part in the FLOSS development (e.g., Henson, 2002; Spertus, 1991). However, facing these gender inequalities, many women remain remote and feel no need of tackling these problems. While some women-centred online groups have networked together to address the gender issue in the FLOSS movement, many female programmers still do not share the same view on an ongoing and enlarging gap between men and women software developers. While gender issue in FLOSS is not addressed in most of the literature and also not recognised by female peers, it is difficult to network women to tackle the coherent patriarchal hegemony in the computer world.

HOW CAN FLOSS EMPOWER WOMEN?

Like many other ICTs, FLOSS carries the powerful potential as a vehicle for advancing social equality. It opens up an opportunity for women to learn how to communicate and interact with software designers and speak out what kind of software they want (e.g., file bug report, join the user group and online forum etc.), to have access to source code and fork the software (e.g., to have a female-friendly version of the software), if they are interested and competent.

There are three main ends in current "women movement in the FLOSS community": (1) providing women-friendly software and services; (2) creating a women-friendly environment for developing and using FLOSS; and (3) fostering a gender-balanced ICT innovation system for both competition and collaboration. These three points have close connections with one another. In order to create software that engage and build on women's ideas and visions, we need to create a more women-friendly environment for the purpose of attracting more women to participate in the FLOSS development. Encompassing such a women-centred view of design, which usually resembles a more sympathetic and inclusive way of doing, will possibly foster a gender-balanced ICT innovation system that is not only friendly to women but also to various minorities in our society. This system, unlike the current one based on a highly competition-oriented approach, will draw on aptitudes and competences of diverse actors in the FLOSS social world so as to develop a holistic environment which is based on a collaboration-oriented approach.

Networking is important in democratising the access and dissemination of knowledge and establishing a base for a citizenship defined by gender equity. In order to encourage women's participation and also to explain the operation of FLOSS to women, some female developers/users have started to network and form online groups such as, LinuxChix[3], KDE Women[4], Gnurias[5], GenderChangers[6], and Debian-women[7]. They act to dispel the unfriendly wording in documents and in online peer groups, to report this kind of sexist bug reports to other developers, to give

online tutorials. These networking and gathering, online or offline, would serve as a base for gender inclusion.

CONCLUSION AND FUTURE RESEARCH

The essay aims to identify the current challenges of gender politics and help formulate strategies and recommendations in order to advance and to empower women in FLOSS. It is anticipated that through conceptualising and documenting the current gender issues in the FLOSS development, it will help enlarge the knowledge base for gender-sensitive policies on ICTs, and propose a women-centred policy towards developing and implementing FLOSS. While FLOSS denotes a new milestone for software development and knowledge making in a broad sense that might alter the social relations of gender, "in this technoscientific advanced era, feminist politics make wider differences in women-machine relationship than the technologies themselves" (Wajcman, 2004). As such, a gender-sensitive agenda for developing FLOSS is urgently needed.

In terms of future research, in order to get a comprehensive overview of the current gender digital divide in the FLOSS social world, more research, both qualitative and quantitative, needs to be conducted. The former would allow us to understand women's experiences and needs better through ethnographical observation, interviews, and focus groups, while the latter would give a fuller picture of general gender problems. Researchers across disciplines are encouraged to analyse FLOSS activities more critically with regard to gender, and to develop conceptual frameworks and methodologies for better understanding and analysing the relationship between FLOSS and gender. Additionally, in encouraging the FLOSS development, governments and organisations should pay extra attention to gender-related issues as well, and take initiatives to include women in the FLOSS development. Holding training workshops for female developers might be a feasible way of bridging the gender digital divide in the FLOSS social world. Other efforts such as design of products and Web sites for women and girls, supporting networks for female professions in FLOSS shall be encouraged.

However, in speaking of implementing and developing FLOSS, most of the cases are centralised or situated in more developed countries. One should bear in mind that there are many undocumented activities that have happened in the developing world. When strengthening the advantages of FLOSS, we should not overlook many problems emerging from implementing FLOSS in developing countries, such as a lack of sufficient training and support. The digital divide shall be considered as a symptom of inequality, not the cause of it. There is a need of understanding what local people really need: water, food, jobs, decent healthcare and sanitation, or software and ICT infrastructure. The gender issue of ICT might be more complex than we thought as well. Female participants very often suffer from hybrid discriminations, both from the male-dominated FLOSS world and the socio-cultural patriarchy in the society. Although virtual groups such Linuxchix Brazil[8] and Linuxchix Africa[9] have started providing women with help on solving problems in implementing Linux, more efforts need to be spent on documenting, analysing and deconstructing the patriarchal hegemony embedded in the whole ICT infrastructure. As such, like many other fields concerned with gendering, this essay is a mere beginning of a feminist accounts about the FLOSS development—an analytic stage on which "we need to place the details contributed by ethnographic research, cultural critiques, sociological surveys, legal scholarship on men and women in their many specific conditions and subjectivities" (Sassen, 1999, p. 2).

REFERENCES

Arun, S., & Arun, T. G. (2001). Gender at work within the software industry: An Indian perspective. *Journal of Women and Minorities in science and engineering, 7*(3), 42-58.

DeBare, I. (1996, January 21). Women in computing: Logged on or left out? [Special report]. *Sacramento Bee*.

Edwards, P. (1993). *Gender and the cultural construction of computing.* Adapted from "From 'impact' to social process: Case studies of computers in politics, society, and culture (Chapter IV-A)," in Handbook of Science and Technology Studies. Beverly Hills, CA: Sage Press.

European Commission. (2001). *Public report on the consultation meeting on European perspectives for open source software*. Retrieved from ftp://ftp.cordis.lu/pub/ist/docs/ka4/tesss-OSS-report.pdf

Ghosh, R. A., Glott, R., Krieger, B., & Robles, G. (2002). *Free/Libre and open source software: Survey and study* (Deliverable D18: Final Report, Part IV: Survey of Developers). International Institute of Infonomics, University of Maastricht and Berlecon Research GmbH. Retrieved original version of this document from http://www.infonomics.nl/FLOSS/report/

Henson, V. (2002). *How to encourage women in Linux*. Retrieved from http://www.tldp.org/HOWTO/Encourage-Women-Linux-HOWTO/index.html

Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. London: Secker & Warburg.

Levesque M., & Wilson, G. (2004). Women in software: Open source, cold shoulder. *Software Development*. Retrieved February 20, 2005, from http://www.sdmagazine.com/documents/s=9411/sdm0411b/sdm0411b.html?temp=TgtgS9YUY8

Levy, S. (1984). *Hackers: Heroes of the computer revolution*. Garden City, NY: Anchor Press/Doubleday.

Lin, Y. W. (2004). *Hacking practices and software development: A social worlds analysis of ICT innovation and the role of open source software*. PhD thesis, Department of Sociology, University of York, UK.

Mitter S., & Rowbotham, S. (1995). *Women encounter technology: Changing patterns of employment in the third world*. London: Routledge and the United Nations University.

Okin, S. M. (1999) *Is multiculturalism bad for women?* Retrieved from http://www.boston review.net/BR22.5/okin.html

Raymond, E. S. (2001). *How to become a hacker*. Retrieved June 23, 2005, from http://www.catb.org/~esr/faqs/hacker-howto.html

Sassen, S. (1999). *Blind spots: Towards a feminist analytics of today's global economy*. Retrieved from http://www.uwm.edu/Dept/IGS/presentation/sassen.pdf

Spertus, E. (1991). *Why are there so few female computer scientists?* (MIT Artificial Intelligence Laboratory Technical Report 1315). Retrieved February 20, 2005, from http://www.mills.edu/ACAD_INFO/MCS/SPERTUS/Gender/pap/pap.html

Thomas, D. (2002). *Hacker culture*. Minneapolis, MN: University of Minnesota Press.

Turkle, S. (1984). *The second self: Computers and the human spirit*. New York: Simon and Schuster.

UNDP Bratislava Regional Center and UNIFEM Central and Eastern Europe. (2004). *Bridging the gender digital divide: A report on gender and information communication technologies (ICT) in Central and Eastern Europe and the*

commonwealth of independent states (CIS). UNDP/UNIFEM.

Wajcman, J. (2004). *TechnoFeminism.* Cambridge, UK: Polity Press.

KEY TERMS

Debian GNU/Linux and Debian-Women: Created by the *Debian Project*, is a widely used free software distribution developed through the collaboration of volunteers from around the world. Since its inception, the released system, The Debian Women project, founded in May 2004, seeks to balance and diversify the Debian Project by actively engaging with interested women and encouraging them to become more involved with Debian.

Ethnography: Refers to the qualitative description of human social phenomena, based on months or years of fieldwork. Ethnography may be "holistic," describing a society as a whole, or it may focus on specific problems or situations within a larger social scene.

FLOSS: Free/libre open source software (FLOSS), generically indicates non-proprietary software that allows users to have freedom to run, copy, distribute, study, change and improve the software.

Focus Group: A focus group is a form of qualitative research in which a group of people are asked about their attitude towards a product, concept, advertisement, idea, or packaging. Questions are asked in an interactive group setting where participants are free to talk with other group members.

GPL: General Public License (GPL) is a free software licence that guarantees the freedom of users to share and change free software. It has been the most popular free software license since its creation in 1991 by Richard Stallman.

Hegemony: Is the dominance of one group over other groups, with or without the threat of force, to the extent that, for instance, the dominant party can dictate the terms of trade to its advantage; or more broadly, that cultural perspectives become skewed to favor the dominant group.

KDE(K Desktop Environment): A free desktop environment and development platform built with Trolltech's Qt toolkit. It runs on most Unix® and Unix®-like systems, such as Linux, BSD, and Solaris.

ENDNOTES

1 By "feminism" I mean the belief that women should not be disadvantaged by their sex, that the moral equality of men and women should be endorsed, and that all forms of oppression should be demolished.
2 http://www.eriders.org
3 http://www.linuxchix.org/
4 http://women.kde.org/
5 http://www.gnurias.org.br/
6 http://www.genderchangers.org/
7 http://women.alioth.debian.org/
8 http://www.linuxchix.org.br
9 http://www.africalinuxchix.org

This work was previously published in Encyclopedia of Gender and Information Technology, edited by E. Trauth, pp. 1286-1291, copyright 2006 by Idea Group Reference (an imprint of IGI Global).

Chapter 2.17
Managing Intellectual Capital and Intellectual Property within Software Development Communities of Practice

Andy Williamson
Wairua Consulting Limited, New Zealand

David M. Kennedy
Hong Kong Institute of Education, Hong Kong

Ruth DeSouza
Wairua Consulting Limited, New Zealand

Carmel McNaught
Chinese University of Hong Kong, Hong Kong

INTRODUCTION

In this article, we will develop a framework for educational software development teams that recognizes the conflicts and tensions that exist between the different professional groups and will assist software teams to recognize the intellectual capital created by individuals and teams. We will do so by recognizing the inherent relationship between the tangible elements of intellectual property and the intangible organizational assets that form the basis of intellectual capital and by discussing how knowledge generated by a project team can become an explicit asset.

BACKGROUND

Universities are increasingly becoming developers of complex software-based applications. In-house development ranges from teaching aids and online learning resources to large information systems

products that could ultimately become successful commercial ventures. Increased product complexity is easily recognized, yet research shows that the organizational aspects of a software development project are more likely to affect performance and outcomes than technical issues (Xia & Lee, 2004). Successful development and deployment of today's complex educational systems and environments comes with an imperative for an array of different and unique skill sets for the various stages of each project. One can view a software development team as a microcosm of the wider community of practice of software development professionals who work in information and knowledge management in higher education. As Wenger (1998) observes, such communities of practice are not random but constructed around required skills and through a process of negotiation based on mutuality and accountability.

Workforce mobility has increased: academic staff members regularly and easily move between institutions; development and design staff have many opportunities for contract-based work, move to other academic institutions or into the private sector. The ideas that lie behind a successful process or product are increasingly drawn from a wider pool of talent and, as people move around, these ideas are being taken with them and disseminated through informal and new work practices into a wider community of practice. How then does a team, formed to design and develop a technology-rich educational or systems environment, manage and control issues of intellectual capital and intellectual property such that all of those who contribute throughout the life of a project are acknowledged and rewarded fairly and appropriately for that contribution, even after they have left the project?

Team Formation and Relationships

Additional complexity leads to specialization (Jacobson, Booch & Rumbaugh, 1998). New ways of working bring with them a shift in power, where the academic expert will often lack the technical skills, time or resources to turn ideas into reality. Instead, they must rely on a team of experts from other disciplines to interpret their ideas, evolve them, and deliver the finished product. As complexity increases, communication between team members becomes paramount; specialist educational designers are required to translate pedagogy into functional specifications that can be understood by software developers and graphic designers. Modern software teams are project-based, where resources come and go as required.

Software development communities of practice exist within a larger organizational context. Roles and responsibilities will vary and are negotiated depending on the toolset and architecture used, the size of the project, and the culture of the organization (Phillips, 1997; Williamson et al., 2003). Project team members can be full- or part-time employees (academic or non-academic) or contractors retained specifically for the project. As such, these roles exhibit complex relationships and interfaces between each other and the project. In Figure 1, a range of typical roles and relationships found in a tertiary education software development project are shown.

During the various stages of the development process, various players move into prominent roles. One way to illustrate this shifting set of work responsibilities is to list the main players at each stage of the process. We will do this using the classic instructional systems design (ISD) model (Dick & Carey, 1990) as it is so well known. (There are many other models, many of which are discussed in Bannan-Ritland, 2003.) The key players at each stage of the ISD model are listed in Table 1. In reality, each team parcels out the work depending on the skill set of individuals in the team.

It is important to be aware of the different communities of practice that exist in this field and ensure that the role of individual team members is able to be promoted appropriately. Professional

Figure 1. Intra-project relationships in software development teams (Williamson et al., 2003, p. 345)

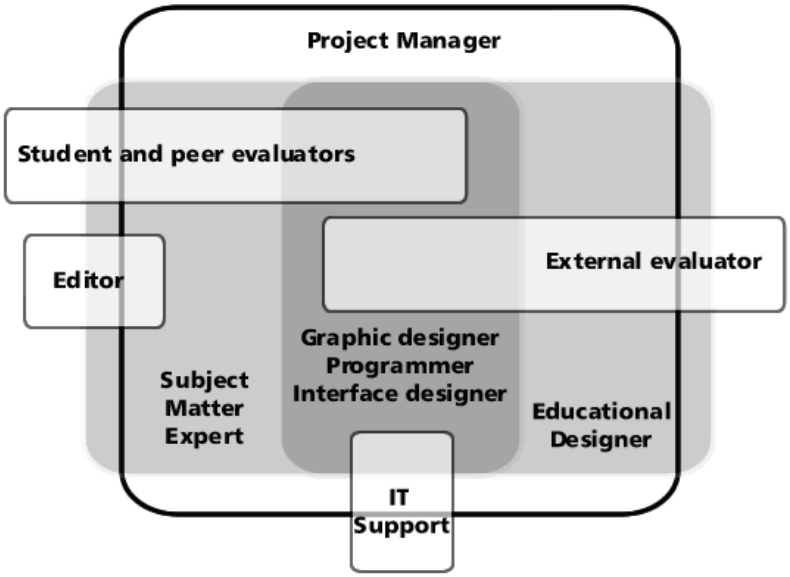


Table 1. Key players at each stage of the ISD model

Stage of the ISD model	Key players
Needs assessment	Subject matter expert
Analysis	Subject matter expert, Educational designer
Design	Subject matter expert, Educational designer, Project manager
Development	Project manager, Graphic designer, Programmer, Interface designer, Editor
Formative evaluation	Student and peer evaluators, Subject matter expert, Educational designer
Revision	Project manager, Graphic designer, Programmer, Interface designer, Editor
Implementation	Subject matter expert, IT support
Summative evaluation	External evaluator
Maintenance	IT support, Subject matter expert (Interface designer)

recognition can come through either publication, a portfolio of work or through the finished product, and the importance of a successful project to the career development of individuals should not be underestimated. It is important to ensure that academic dissemination of successful projects through publication recognizes the contribution made by all team members, including the non-academic members. Many myths persist in relation to acknowledging the veracity of contribution with regard to educational software, and these often have the potential to leave team members feeling their effort and ideas have gone unrecognized and, at worst, feeling they have been exploited (Williamson et al., 2003). In the second half of this article, we will develop a framework that ensures appropriate outlets for reward and recognition of individual contributions within academic software development teams. Before doing this, we will define what is meant by intellectual capital and intellectual property.

Defining Intellectual Capital and Intellectual Property

Florida (2002) argues that the principal factors for successful software development are talent, knowledge, and intellectual capital (IC). The connection of new ideas and existing knowledge within an organizational context leads to the creation of IC. Stewart (1999) defines IC as the sum of everything everybody in a company, organization, or team knows and which provides some advantage over their competitors. Davidson and Voss (2001) agree, describing individual IC as "the sum of individual imagination, intelligence and ideas" (p. 60). They then extend this definition to encapsulate a model for organizational IC that is based on the talent of individuals (human capital), the knowledge that is captured within systems and processes (structural capital), and the characteristics of relationships with customers and suppliers (customer/supplier capital). Organizational IC comes from the "interplay of all three (structural capital augments the value of human capital, leading to an increase in customer/supplier capital)" (p. 61). In terms of this discussion as it relates to the appropriate recognition and acknowledgement of individual contributions within software development teams, human capital is our primary focus. Human capital is "what walks out of the door at the end of the day" (p. 68); it is a vital intangible.

If IC is the intangible but invaluable contribution of human talent to a project, then Intellectual Property (IP) is a formal measurable subset. It is the tangible product that results from the idea. The UK Patent Office (United Kingdom Patent Office, n.d.) defines four formal types of IP:

- patents for inventions
- trademarks for brands
- designs for product appearance
- copyright for material (including software and multimedia).

This definition is then extended to cover a much broader and often more intangible grouping that extends to trade secrets, plant varieties, geographical indications, and performers rights. While many see copyright as a way of protecting IP, it is only a subset. Copyright provides recognition of their invention to the creators of software or multimedia products in order for them to be able to obtain economic rewards for their efforts (Macmillan, 2000). Historically, comparisons have been drawn between software development and the traditional arts, such comparisons reinforcing an argument that IP law is focused on the protection of software such that others are not able to modify the source product (White, 1997). It is important to note that copyright extends only to a tangible product, it does not lend protection to the more intangible areas of IC such as ideas and individual contribution. Since copyright has a primarily commercial imperative, it is a limited and perhaps inappropriate mechanism for acknowledging contribution. This is of greater importance in higher educational settings since copyright of educational materials can reside with the institution (particularly with off-campus courses), rather than the individual, and very few educational software products developed for specific content domains in higher education are ever commercialized (Alexander, McKenzie & Geissinger, 1998).

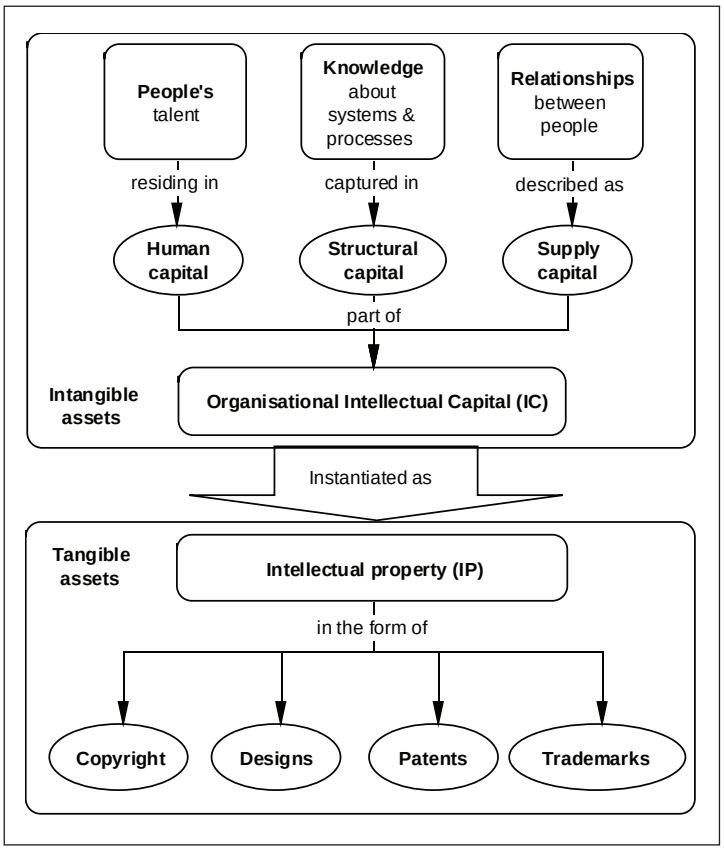
The Relationship between IC and IP

A relationship exists between the tangible elements of intellectual property and the three forms of intellectual capital (the intangible organizational assets) discussed in the preceding section. These are shown in Figure 2.

IC/IP Management Framework

Having addressed the complexity of educational software development teams and defined IC and IP within an educational software context, we will

Figure 2. Intellectual capital and intellectual property (Williamson et al., 2002, p. 342)



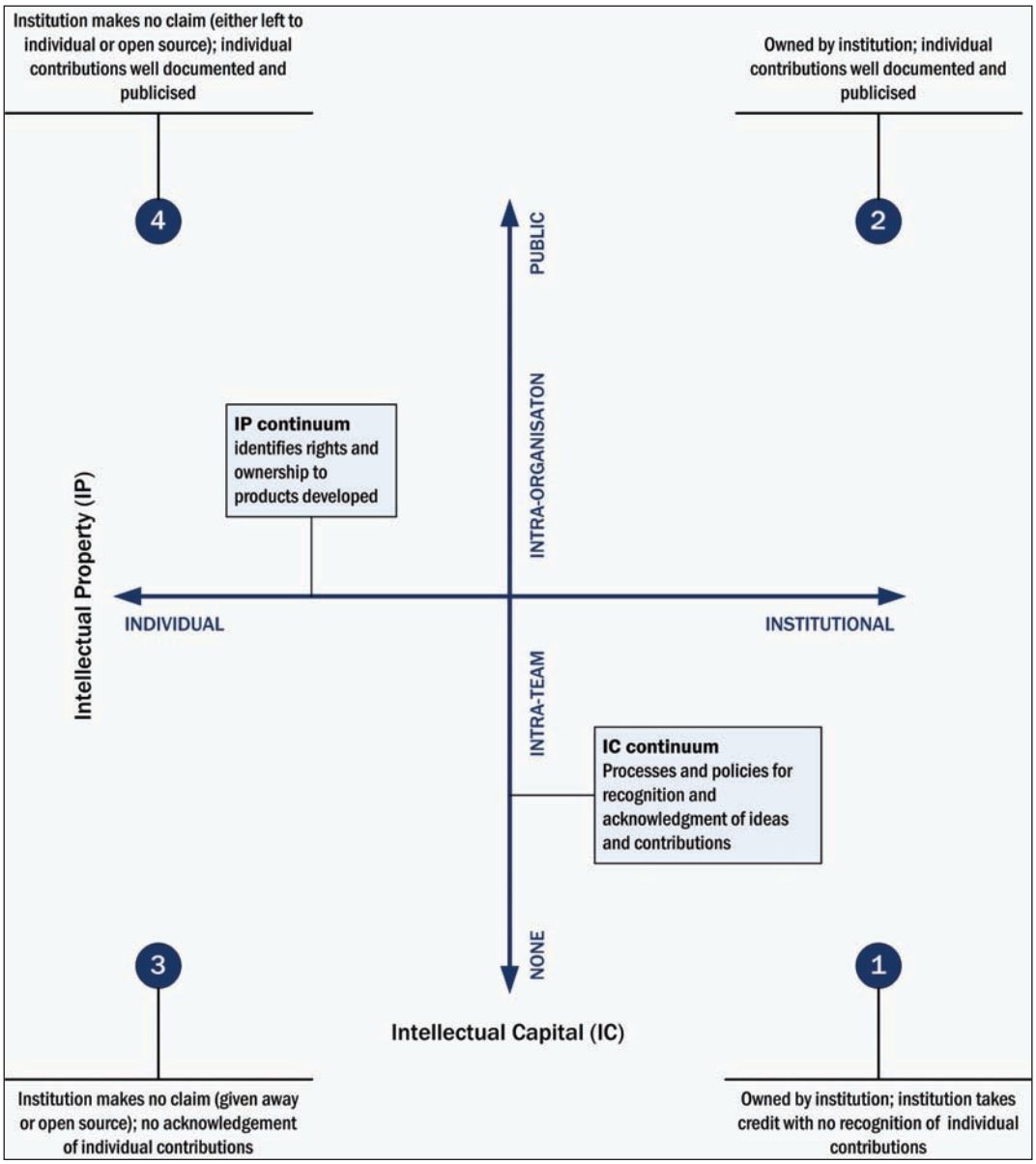
now develop a framework that can be used to ensure proper recognition and reward for individual and collective ideas in such a setting.

Given the critical value of IC in software development (Florida, 2002), it is important that the processes used within educational software development are strengthened and formalized through the adoption of a strong project management framework. Project management is a key role in any project involving information and communication technologies and interactive multimedia software, and it requires specific skills and attributes. These include both the hard skills of contract negotiation, budgeting, scheduling, project definition, and scoping as well as the soft skills of human relations, team building, and facilitation (Burdman, 2000; Schwalbe, 2000). Successful teams work well together because they have clear roles and relationships and because the terms of engagement within the team and with external parties are well defined, understood, and agreed by all. This provides a solid platform for the explicit incorporation of IC and IP policies into project documentation so that such issues can be considered early on, preferably during the project scoping phase.

A process and framework are required to recognize knowledge as it is created so that it becomes explicit. Without doing so, knowledge remains tacit and cannot be rewarded or acknowledged, that is, credited to the appropriate team members in the future. Extending this concept, knowledge

that is explicit within the team can remain tacit beyond team boundaries if no process is in place to ensure appropriate recognition of contribution. It is, therefore, necessary for teams to negotiate clear, up-front delineation of roles, responsibility, and ownership of both tangible and intangible outputs from the project. This does not prejudge what that ownership might be, merely that the agreement takes place before the project commences. It is important to consider how IC/IP generated during the project's life will be disseminated, in what form, and by whom. Such a clear articulation of roles and responsibilities has the benefit of helping to make the process of dissemination more

Figure 3. IP/IC management framework



visible. By doing so, it is hopefully the case that team members will recognize the significance of the different sources of acknowledgment. This in turn will result in up-front agreement on potential opportunities for dissemination of original ideas among the team.

A seldom discussed aspect of the manner in which ideas might be disseminated (and credit obtained) is the potential synergy between individual team members. For example, among academic staff involved in the project, there is a possibility for cross-disciplinary publications.

This framework, shown in Figure 3, maps out two axes: the horizontal axis representing formal ownership of the tangible IP, the vertical axis representing a continuum of recognition for the IC generated during a project, ranging from no acknowledgment of individual effort and contribution to a full public acknowledgment. Intermediate steps include recognition at the team and institutional level.

Enacting the IC/IP Framework

Our discussion so far has shown that, regardless of the nature of the IP ownership, academics and professionals working in software development teams need appropriate recognition for their contributions, but certain factors can prevent this from happening. The challenge, therefore, is to identify a set of project attributes that can be used to inform project management practices such that institutions are cognizant of the need for appropriate recognition. In the following section, we identify seven key attributes of, or processes within, a successful project. The model is developed from a review of the authors' own experiences of software development teams where problems had occurred. This review led to the identification of which weaknesses in the process had resulted in these problems (Williamson et al., 2003). By ensuring that these seven attributes are recognized and actively negotiated by newly forming teams and enacted throughout the life of the team, this model can assist projects in identifying and filling gaps in the structure of development teams, hence future risk can be mitigated.

In essence, the IC embedded in the members of the project team is articulated in terms of the various IP contributions made by these team members. However, if the focus is exclusively on the tangible products of the process (e.g., software and papers) through only considering the IP, then the worth of ideas (IC) can be underplayed, and their potential may not be realized. We are suggesting that explicit application of these seven guidelines can ensure more successful project outcomes and positive professional outcomes for all members of the team.

The nature of an effective community of practice for software development teams is discussed in terms of Figure 3. The two major axes and the four examples in Figure 3 are used to frame the seven attributes.

Intellectual Property: Individual Affirmation to Institutional Affirmation

Have an IP Acknowledgement Strategy

Highly successful projects exhibit a strong team dynamic which arises when the expertise and knowledge of individual team members can be communicated and shared with others. Part of this process involves ensuring that ideas are fairly acknowledged within and outside the team, whether by portfolio (graphic artists), publication (academics), or product (project managers and programmers).

Have an IP Review Strategy

It does not matter for the purposes of academic critical review whether the subject of study is a written paper, a software product, or a portfolio. Contribution from individual team members needs to be acknowledged through an inclusive authorship policy which is regularly revisited in

team meetings. This process can strengthen collegiality and reinforce mutual valuing between team members.

Have a Strategy to Separate IP from IC

The IP might be owned by an organization or institution, but the IC remains with the individuals in the team. Formal acknowledgment of where the ownership of IP lies is important and needs to be negotiated ahead of the commencement of the project. In many higher education institutions, this has become standard practice and involves retaining a competitive advantage and protecting the resources produced by employees of the organization. There are risks associated with key project contributors leaving (for example, a lead programmer) and either taking intellectual property with them or holding a software development team or institution to ransom by withholding access to code or other resources. In some organizations, the IC also remains with the organization via means of a nondisclosure agreement. Communities of practice might consider using a confidentiality agreement as part of a contract or offer of employment in order to keep this issue open and transparent.

Longevity Strategy: Ideas Remain

When a person leaves a team, they cede their IC to the project team or institution, and that contribution should continue to be recognized and acknowledged in project documentation, appropriate publications, and authorship in any finished product. In some projects, this may also involve ceding formal IP to the project (e.g., in the case of commercialization).

Intellectual Capital: No Formal Acknowledgment to Public Acknowledgment

Recognize the Emergent Nature of the Software Development Process and Its Impact on IC/IP for all Team Members

As software becomes more complex, it becomes less and less likely that the original academic imperative that led to the idea for the product will be instantiated in a form initially envisaged by the academic or the organizational unit that initiated the project. The development process and the end result will be strongly influenced by a wide range of individual and group contributions to the process and the product.

Ideas are Perishable

Software has a shelf-life, hence the IC that led to that product is also of limited use. The idea will become superseded and outdated as new ideas and new technologies emerge. For example, there are any number of commercial or free customizable online survey instruments (such as Survey Monkey, http://www.surveymonkey.com) that now exist. Learning Evaluation Online (LEO) was an early system that explored how customizable educational surveys could be developed online using an entirely Web-based interface (Kennedy & Ip, 1998). At the time this was an innovative approach, but it has since been superseded by more robust software. Thus, the IC for LEO has long since expired. The idea behind LEO has been taken up by others and reproduced using different software code. The code is the instantiation of the idea and is the only part of the project subject to IP rights.

Table 2. Implications for IP and IC

#	Scenario	Implications for IP and IC
1	IC and IP is owned by institution; institution takes credit with no contribution of individual creativity and effort.	This is a very poor scenario for developing the IC of an institution. Without affirmation, individuals will seek employment elsewhere and take their IC with them.
2	IP and IC are owned by the institution; individual contributions are well documented and publicized.	This is the scenario in a number of institutions worldwide, particularly those involved in distance education. This scenario is problematic when commercial aspects enter the situation as in the case of patents.
3	Institution makes no claim (software is given away or open source); no acknowledgment of individual contributions.	This is often the result of small student projects (although many institutions claim the IP of all undergraduate student work, not postgraduate) undertaken during a course of study). Most software of this type has a very limited life although there are some exceptions (Gunn, 1995).
4	Institution makes no claim (either left to the individual or open source community); individual contributions are well documented and publicized.	This applies to postgraduate work in universities. In many institutions, postgraduate (especially doctoral) students own their IP, and it is up to the student and supervisor to disseminate the details of the project. This aspect is changing as universities try to gain a competitive advantage, and many postgraduate students working in a large research department would do well to consider how the results of their studies might be retained, negotiating with the university in the early phases of the project. For example, some student projects (see **moodle.org** and moodle.com) have become very high profile products (Dougiamas & Taylor, 2003).

Public Acknowledgment of IP/IC Requires the Source Material to be in the Public Domain

Acknowledgment of unpublished work or work not publicly available is not sufficient to acknowledge IC and IP issues in a publication. In the case of academics where affirmation and professional career progress is at least partially a result of publication in accredited arenas such as books and journals, this is clearly not sustainable. Graphic artists, on the other hand, have their portfolios of work with iterations of visual designs that they take with them to the next project or job; and programmers have compilations of code: for these professionals, the publication is less important or substantive in career development. A key issue for an institution is providing the process by which academic publications can be developed without compromising the IP of the individual or trade advantages in the marketplace.

In summary, the implications for the four scenarios in Figure 3 are shown in Table 2.

THE FUTURE: APPLICATION OF THE SEVEN ATTRIBUTES OF THE IC/IP FRAMEWORK

In order to see how these attributes can be enacted in practice, the example of a major Australian multimedia project, *An@tomedia*, will be used. *An@tomedia* was designed to support problem-based learning (PBL) of anatomy in the Faculty of Medicine at the University of Melbourne (http://www.anatomedia.com). A number of academic evaluations on the role of *An@tomedia* in this PBL learning environment have been published (e.g.,

Table 3. Example of the application of the seven attributes of the IP/IP framework

#	Attribute	Enactment
1	Have an IP acknowledgment strategy	The acknowledgment of IP was never an issue within the project group. Individual contributions were always acknowledged by the core development team, becoming part of the documentation of the project. The extensive documentation ensured that no one was left off the credits on the *An@tomedia* Web site.
2	Have an IP review strategy	The existence of "prior art" was established in the early phases of the project. While the final product did not resemble the initial designs, it was always clear in the project meetings that the team was involved in the instantiation of the educational vision of the project leader (one of the core SMEs).
3	Have a strategy to separate IP from IC	The strategy to separate IP from IC was undertaken by the four principal authors as *An@atomedia* was moved from an interesting project to a commercial product. The documentation resulting from meetings included discussions of commercialization of *An@atomedia* and the associated need to separate IP from IC. The strategy adopted involved consultations with the university's legal advisors and the project team. The IP for the sale and commercial rights to *An@tomedia* were ceded to the four key authors by the other team members; however, the IC remained with members of the project team to use as they required.
4	Longevity strategy: Ideas remain	The credits list contains a list of all members who contributed to the project over a period of many years, including those individuals who either retired (in one case) or moved to other institutions (a number of people). It is possible for all members of the project team to include evidence of contributions to *An@atomedia* by reference to either the Web site or the CD-Roms (the form in which *An@tomedia* is published and sold).
5	Recognize the emergent nature of the software development process and its impact on IC/IP for all team members	The development of *An@tomedia* occurred over a considerable period of time. The genesis of some the clinical approaches adopted in the project were developed by the project leader and occurred well before *An@tomedia* commenced (Eizenberg, 1988, 1991). The use of technology followed as a consequence of the need to develop more effective and engaging approaches to the teaching of anatomy (Driver & Eizenberg, 1995). As the final design of the software emerged, it was always clear in meetings and associated documentation that other members of the project team were involved in the instantiation of prior concepts and developments in new and innovative ways, but the underlying concept derived from the earlier work in paper-based media.
6	Ideas are perishable	*An@atomedia* received a number of awards for innovation and excellence after the first release (see **http://www.anatomedia.com/credits.shtml**). However, as people come and go from the project, the initial ideas will be superseded or altered to reflect teaching evaluations, changes in the medical curriculum, and improvements in technology. Solutions developed in 1999 or 2000 may not be suitable in 2005. What was once a good idea may not be appropriate in the future, but the three major methods of affirmation remain—publication, portfolio, and vitae for all contributions.
7	Public acknowledgment of IP/IC requires the source material to be in the public domain	The *An@atomedia* Web site provides definitive acknowledgment of the specific contributions of individuals, including the evaluators, programmers, educational consultants, photographers, medical consultants, project managers, dissectors, illustrators, interface and graphic designers, and research assistants, to name a few.

Kennedy, Eizenberg & Kennedy, 2000; Kennedy, Kennedy & Eizenberg, 2001).

The software has been successfully commercialized by the four subject matter experts (core SMEs or core authors) after other members of the development team ceded any personal commercial claims to the group by means of a legal document to that effect.

Affirmation and acknowledgment involving publishing for academic members (Kennedy et al., 2000; Kennedy et al., 2001), contributions to portfolios for non-academic members, and public acknowledgments in the *An@tomedia* Web site for every person who contributed in any significant way to the project were not affected by this written agreement. The public affirmation (particularly important for non-academic members of the project team) is illustrated by the observation made by a reviewer of *An@tomedia* in *The Lancet* (Marušić, 2004) where she mentions the extensive list of credits for all the members of the team (over 60) involved with the project. This process was accomplished quite simply because matters of IP had been previously discussed in the course of project meetings, and the "prior art" that existed and underpinned the educational approach was well known to all project members. Table 3 summarizes the way in which the seven attributes worked in this project.

CONCLUSION

While formalized tools exist for capturing IP generated during a project, most software development teams lack formal explicit processes for ensuring that the IC generated is accurately and adequately apportioned. This article has raised issues relating to how software development project teams are recognized for their contribution and a simple framework for measuring recognition of contribution has been presented. Seven key project attributes or processes have been identified to assist project teams develop an awareness of how project roles and structures can be negotiated so that tacit ideas and knowledge generated can become explicit. Such a model must recognize that the requirements for, and process of, recognition will differ within different multiskilled teams. The application of the framework to one major multimedia project has been discussed.

REFERENCES

Alexander, S., McKenzie, J., & Geissinger, H. (1998). *An evaluation of information technology projects in university learning.* Canberra: Australian Government Publishing Services, Department of Employment, Education and Training and Youth Affairs.

Bannan-Ritland, B. (n.d.). The role of design in research: The integrative learning design framework. *Educational Researcher, 32*(1), 21-24.

Burdman, J. (2000). *Collaborative Web development: Strategies and best practices for Web teams.* Reading, MA: Addison-Wesley.

Davidson, C., & Voss, P. (2001). *Knowledge management: An introduction to creating competitive advantage from intellectual material*. Auckland, NZ: Tandem.

Dick, W., & Carey, L. (1990). *The systematic design of instruction*. Glenview, IL: Foresman/ Little.

Dougiamas, M., & Taylor, P. (2003) Moodle: Using learning communities to create an open source course management system. In D. Lassner & C. McNaught (Eds), *ED-MEDIA 2003, Proceedings of the 15th Annual World Conference on Educational Multimedia, Hypermedia and Telecommunications* (pp. 171-178). Honolulu, Hawaii. Norfolk VA: Association for the Advancement of Computers in Education.

Driver, C., & Eizenberg, N. (1995). Constructing and deconstructing the human body: Applying

interactive multimedia in the learning of anatomy. In J. M. Pearce, A. Ellis, C. McNaught, & G. Hart (Eds.), *Learning with technology: ASCILITE 95. Proceedings of the 12th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education* (pp. 586-587). The University of Melbourne: The Science Multimedia Teaching Unit.

Eizenberg, N. (1988). Approaches to learning anatomy: Developing a program for pre-clinical medical students. In P. Ramsden (Ed.), *Improving learning: New perspectives* (pp. 178-198). London: Kogan Page.

Eizenberg, N. (1991). Action research in medical education: Improving teaching via investigating learning. In O. Zuber-Skerrit (Ed.), *Action research for change and development* (pp. 179-206). Avebury: Aldershot.

Florida, R. L. (2002). *The rise of the creative class: And how it's transforming work, leisure, community and everyday life*. New York: Basic Books.

Gunn, C. (1995). Useability and beyond: Evaluating educational effectiveness of computer-based learning. In G. Gibbs (Ed.), *Improving student learning through assessment and evaluation* (pp. 168-190). Oxford, UK: Oxford Centre for Staff Development.

Jacobson, I., Booch, G., & Rumbaugh, J. (1998). *The unified software development process*. Reading, MA: Addison-Wesley Longman.

Kennedy, D. M., Eizenberg, N., & Kennedy, G. (2000). An evaluation of the use of multiple perspectives in the design of computer-facilitated learning. *Australian Journal of Educational Technology, 16*(1), 13-25. Retrieved May 5, 2005, from *http://www.ascilite.org.au/ajet/ajet16/kennedy.html*

Kennedy, D. M., & Ip, A. (1998). *Learning Evaluation Online (LEO): A customizable Web-based evaluation tool*. In C. Alvegard (Ed.), *Computer aided learning and instruction in science and engineering. CALISCE'98 proceedings* (pp. 255-262). Goteborg: Chalmers University of Technology.

Kennedy, G. E., Kennedy, D. M., & Eizenberg, N. (2001). Integrating computer-facilitated learning resources into problem-based learning curricula. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 3*(1). Retrieved May 5, 2005, from *http://imej.wfu.edu/articles/2001/1/02/index.asp*

Macmillan, F. (2000). Intellectual property issues. In C. McNaught, P. Phillips, D. Rossiter, & J. Winn (Eds.), *Developing a framework for a usable and useful inventory of computer-facilitated learning and support materials in Australian universities. Evaluations and Investigations Program report 99/11* (pp. 189-205). Canberra: Higher Education Division Department of Employment, Education, Training and Youth Affairs.

Marušić, A. (2004). Media reviews: Interactive anatomy. *The Lancet, 363*(9404), 254.

Phillips, R. A. (1997). *A developer's handbook to interactive multimedia: A practical guide for educational applications*. London: Kogan Page.

Schwalbe, K. (2000). *Information technology project management*. Cambridge, MA: Course Technology.

Stewart, T. (1999). *Intellectual capital: The new wealth of organizations*. New York: Currency.

United Kingdom Patent Office. (n.d.). *What is intellectual property or IP?* Retrieved May 5, 2005, from *http://www.intellectual-property.gov.uk/std/faq/question 1.htm*

Wenger, E. (1998). *Communities of practice: Learning, meaning and identity*. Cambridge, UK: Cambridge University Press.

White, J. A. D. (1997). Misuse or fair use: That is the software copyright question. *Berkeley Technology Law Journal, 12*(2). Retrieved May 5, 2005, from *http://www.law.berkeley.edu/journals/btlj/articles/vol12/White/html/reader.html*

Williamson, A., Kennedy, D. M., McNaught, C., & DeSouza, R. (2003). Issues of intellectual capital and intellectual property in educational software development teams. *Australian Journal of Educational Technology, 19*(3), 339-355. Retrieved May 5, 2005, from *http://www.ascilite.org.au/ajet/ajet19/williamson.html*

Xia, W., & Lee, G. (2004). Grasping the complexity of IS development projects. *Communications of the ACM, 47*(5), 69-74.

KEY TERMS

Educational Software Development Teams: Software development teams in university settings are multifaceted and multiskilled, requiring the skills of project managers, subject matter experts, educational designers, programmers, graphic designers, interface designers, IT support staff, editors, and evaluators. In many cases, one person can assume more than one role.

Intangible Assets: Higher education institutions are traditionally based upon ideas, one form of intangible assets. However, the modern university frequently seeks to differentiate itself from its competition. In this article, intangible assets include an investment or outcome enjoyed by the institution in knowledge-based resources and processes. Typically, these intangible assets are termed soft assets because they are not either infrastructure or equipment. Some examples are training programs, improvements to organizational communication flows, or new quality assurance systems.

Intellectual Capital (IC): IC is the sum of the individual imagination that, when aggregated, becomes everything everybody in an organization or team knows and which provides them with some advantage over their competitors. Organizational IC comes from the interplay of structural capital, which augments the value of human capital, leading to an increase in customer/supplier capital.

Intellectual Property (IP): IP is a formal measurable subset if IC; the tangible product that results from the idea and is represented and recognized through patents, trademarks, designs, and copyright (which includes software and multimedia). IP can also be extended to cover a much broader and often more intangible grouping that extends to trade secrets, plant varieties, geographical indications, and performers' rights.

Prior Art: In this article, the concept of "prior art" is used in a similar manner to that adopted by the patent office. However, higher education has a long tradition of valuing ideas, not just economic value (as in patent laws). The "prior art" in this instance refers to the intangible ideas (instantiated in the earlier publications) prior to the commencement of a project, such as the An@tomedia project. It was the basis of these earlier ideas that formed the nucleus of the design philosophy and influenced the manner in which the developers reached agreement on design decisions.

Project Framework: Negotiation of a model within the project to ensure that the contributions of all individuals (and their IC) are able to be appropriately recognized.

Recognition: Different professional groups look for and require different forms of recognition for their professional development. Where academic staff focus on publication, designers need to develop a portfolio of work, and software developers receive kudos and build a reputation based on a product that has been developed and the code therein.

This work was previously published in Encyclopedia of Communities of Practice in Information and Knowledge Management, edited by E. Coakes; S. Clarke, pp. 364-374, copyright 2006 by Idea Group Reference (an imprint of IGI Global).

Chapter 2.18
Developing Knowledge Management Systems from a Knowledge-Based and Multi-Agent Approach

Aurora Vizcaíno
Alarcos Research Group, University of Castilla-la Mancha, Spain

Juan Pablo Soto
Alarcos Research Group, University of Castilla-la Mancha, Spain

Javier Portillo-Rodríguez
Alarcos Research Group, University of Castilla-la Mancha, Spain

Mario Piattini
Alarcos Research Group, University of Castilla-la Mancha, Spain

ABSTRACT

Developing knowledge management systems is a complicated task since it is necessary to take into account how the knowledge is generated, how it can be distributed in order to reuse it, and other aspects related to the knowledge flows. On the other hand, many technical aspects should also be considered such as what knowledge representation or retrieval technique is going to be used. To find a balance between both aspects is important if we want to develop a successful system. However, developers often focus on technical aspects, giving less importance to knowledge issues. In order to avoid this, we have developed a model to help computer science engineers to develop these kinds of systems. In our proposal we first define a knowledge life cycle model that, according to literature and our experience, ponders all the stages that a knowledge management system should give support to. Later, we describe the technology (software agents) that we recommend

to support the activities of each stage. The article explains why we consider that software agents are suitable for this end and how they can work in order to reach their goals. Moreover, a prototype that uses these agents is also described.

INTRODUCTION

In the last decades, knowledge management (KM) has captured enterprises' attention as one of the most promising ways to reach success in this information era (Malone, 2002). A shorter life cycle of products, globalization, and strategic alliances between companies demand a deeper and more systematic organizational knowledge management. Consequently, one way to assess an organization's performance is to determine how well it manages its critical knowledge.

In order to assist organizations to manage their knowledge, systems have been designed. These are called knowledge management systems (KMS), defined by Alavi and Leidner (2001) as IT-based systems developed to support/enhance the processes of knowledge creation, storage/retrieval, transfer, and application.

However, developing KMS is a difficult task; since knowledge *per se* is intensively domain dependent whereas KMS often are context specific applications. Thus, reusability is a complex issue. On the other hand, the lack of sophisticated methodologies or theories for the extraction of reusable knowledge and reusable knowledge patterns has proven to be extremely costly, time consuming, and error prone (Gkotsis, Evangelou, Karacapilidis & Tzagarakis, 2006). Moreover, there are several approaches towards KMS developing. For instance, the process/task based approach focuses on the use of knowledge by participants in a project, or the infrastructure/generic system based approach focuses on building a base system to capture and distribute knowledge for use throughout the organization (Jennex, 2005). On the other hand, before developing this kind of system it is advisable to study and understand how the transfer of knowledge is carried out by people in real life. However, when developing KMS, developers often focus on the technology without taking into account the fundamental knowledge problems that KMS are likely to support (Hahn & Subramani, 2000).

Different techniques have been used to implement KMS. One of them, which is proving to be quite useful, is that of intelligent agents (van Elst, Dignum & Abecker, 2003). Software agent technology can monitor and coordinate events or meetings and disseminate information (Wooldridge & Jennings, 1995). Furthermore, agents are proactive in the sense that they can take the initiative and achieve their own goals. The autonomous behavior of the agents is critical to the goal of this research since it can reduce the amount of work that employees have to perform when using a KM system. Another important issue is that agents can learn from their own experience. Consequently, agent systems are expected to become more efficient with time since the agents learn from their previous mistakes and successes (Maes, 1994).

Because of these advantages, different agent-based architectures have been proposed to support activities related to KM (Gandon, 2000). Some architectures have even been designed to help in the development of KMS. However, most of them focus on a particular domain and can only be used under specific circumstances. What is more, they do not take into account the cycles of knowledge in order to use knowledge management in the system itself. For these reasons, in this article we propose a generic model for developing KMS. Therefore, in the next section we describe the model and the software agents that we propose to support it. In the following section, we explain how the agents are structured and how they have been modeled using the INGENIAS methodology. Later, the next section describes a prototype that we are implementing by using the agents proposed in the model. The following section summarizes

related works carried out with agents. Finally, conclusions and future work are outlined in the last section.

A MULTI-AGENT MODEL TO DEVELOP KNOWLEDGE MANAGEMENT SYSTEMS

A successful KMS should perform the functions of knowledge creation, storage/retrieval, transfer, and application (Jennex & Olfman, 2006). Taking this fact into account and after reviewing several knowledge life cycles and models (see Table 1) and seeing what stages most authors considered, we decided to define a knowledge life cycle that indicates what process a KMS should support (see Figure 1). This is a focus different to the previous one based on describing the knowledge cycle in human beings and/or in companies.

The stages of our proposal are acquisition, storage, use, transfer, and evaluation. The first three stages are considered in most knowledge life cycles (see Table 1). We have added transfer (also considered in several cycles) and evolution. Transfer is added because a KMS should disseminate knowledge to those people that can need it. Evolution is added because knowledge should always be updated otherwise it would not be used.

Figure 1. Knowledge life cycle model proposed

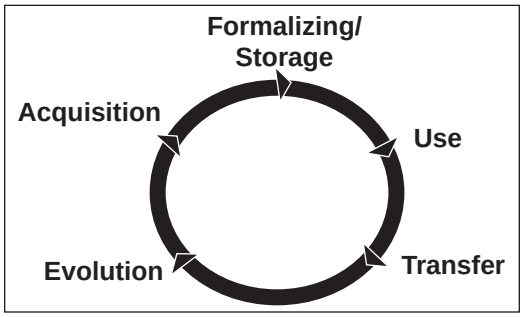


Table 1. Knowledge life cycle

Model	Stage1	Stage2	Stage3	Stage4	Stage5	Stage6	Stage7
Nonaka and Takeuchi (Nonaka & Takeychi, 1995)	Socialization	Externalization	Combination	Internalization			
Wiig (Wiig, 1997)	Creation	Storing/ gathering	Use	Leverage	Sharing		
Davenport and Prusak (Davenport & Prusak, 1998)	Generation	Codify/ Coordinate	Transfer	Roles and Skills			
Tiwana (Tiwana, 2000)	Acquire	Sharing	Use				
Alavi and Leidner (Alavi & Leidner, 2001)	Creation	Storage/ Retrieval	Transfer	Application			
Rus and Lindvall (Rus & Lindvall, 2002)	Creation/ Acquisition	Organization/ Storage	Distribution	Application			
Nissen (Davenport, 1998)	Creation	Organization	Formalize	Distribute	Application	Evolve	
Ward and Aurum (Ward & Aurum, 2004)	Creation	Distribution	Organization	Adaptation	Identification	distribution	Application
Dickinson (Dickinson, 2000)	Identification	Acquisition	Development	Distribution	Use	Preservation	

In the following paragraphs each stage of the model is described. At the same time and with the goal of illustrating that it is possible to support each stage by using current technology, we are going to explain how a software agent could be implemented for a KMS.

a. *Knowledge acquisition* is a key component of a KMS architecture. This stage includes the elicitation, collection, and analysis of knowledge (Rhem, 2006). During this process, it is vital to determine where in the organization the knowledge exists and how to capture it. The definition of the knowledge to be acquired can be assisted by classifying types of knowledge and knowledge sources (Dickinson, 2000). To support this stage we propose to use an agent called a *Captor Agent*. The Captor Agent is responsible for collecting the information (data, models, experience, etc.) from the different knowledge sources. It executes a proactive monitoring process to identify the information and experiences generated during the interaction between the user and the system or groupware tools (e-mail, consulted Web pages, chats, etc.). In order to accomplish this, the Captor Agent can use different techniques to acquire knowledge since there are several tools and techniques that consolidate and transform corporate data into information (Houari & Homayoun Far, 2004). They contain:
    - Front-end system (i.e., decision support system [DSS], executive information system [EIS], and online analytical processing [OLAP]).
    - Back-end system: data warehouse, data mart, and data mining (Giannella, Bhargava et al., 2004).

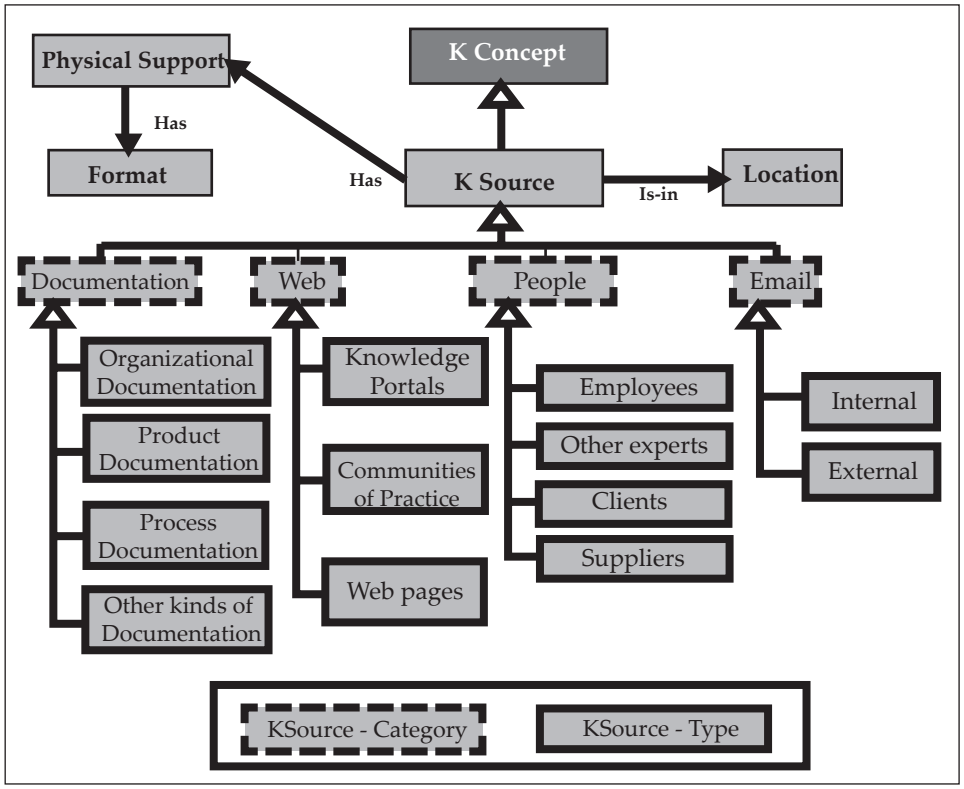
Agents can also apply classical techniques used by experts to acquire knowledge such as structured interviews, questionnaires, goal trees, decision networks, repertory grids, or conceptual maps (Rhem, 2006). More sophisticated techniques such as WebParser (Camacho, Aler & Cuadrado, 2004) to obtain information from the Web, document classification (Novak, Wurst, Fleschmann & Strauss, 2003), mailing list management (Moreale & Watt, 2003), or data mining and neuronal nets can be also used.

Once the knowledge has been obtained, the Captor Agent can classify it by using ontologies according to its type and the knowledge source from it was obtained (see Figure 2). This ontology is based on Rodriguez's ontologies for representing knowledge topics and knowledge sources (Rodríguez, Martínez, Favela, Vizcaíno & Piattini, 2004).

The ontology has four knowledge source categories: Documentation, which can be subdivided into documentation related to the organization's philosophy, documentation which describes the product/s which the company works with, documentation that describes the process that the company carries out, and other types of documentation that an organization has but that cannot be classified into any of the previous subgroups. Another important source where the Captor Agent finds information is the Web, which can also be divided into other subcategories such as portals, communities of practice, and so forth. The main knowledge source in a company is, without any doubt, people. Depending on the type of company, people may be classified as clients, employees, and so forth. The last knowledge source that the Captor Agent can use is e-mail that can be classified as internal mail (mail sent between employees), and external mail (e-mails sent to other people outside the organization).

One advantage of this approach is that the Captor Agent can work in any domain since by changing these ontologies the Captor knows what key knowledge should be found and where it might be.

Figure 2. Knowledge source ontology



b. *Knowledge formalizing/storing* is the stage that groups all the activities that focus on organizing, structuring, representing, and codifying the knowledge with the purpose of facilitating its use (Davenport & Prusak, 1998). To help carry out these tasks we propose a *Constructor Agent*. This agent is in charge of giving an appropriate electronic format to the experiences obtained so that they can be stored in a knowledge base to aid retrieval. Storing knowledge helps to reduce dependency on key employees because at least some of their expert knowledge has been retained or made explicit. In addition, when knowledge is stored, it is made available to all employees, providing them with a reference as to how processes must be performed, and how they have been performed in the past. Moreover, the Constructor Agent compares the new information with old knowledge that has been stored previously and decides whether to delete it and add new knowledge or to combine both of them. In this way, the combination process of the SECI (Nonaka, 1994) model is carried out, producing new knowledge resulting in the merging of explicit knowledge plus new explicit knowledge.

Different techniques exist to store knowledge and frequently the technique used is narrowly related to the retrieval method used. Therefore, if a case-based reasoning is going to be used, the knowledge will be stored as "cases." Other techniques are knowledge objects, frames, predicate logic, or fuzzy logic. In the case of using ontolo-

gies to classify the knowledge, methodologies to represent the knowledge can be used. Examples of these methodologies are Ontolingua (Gruber, 1993) or Representation Formalism for Software Engineering Ontologies (REFSENO) (Tautz & Von Wangenheim, 1998).

c. *Knowledge Use* is one of the main stages, since knowledge is helpful when it is used and/or reused. The main enemy of knowledge reuse is ignorance. Employers often complain because employees do not consult knowledge sources and do not take advantage of the knowledge capital that the company has. KMS should offer the possibility of searching for information; they can even give recommendations or suggestions with the goal of helping users to perform their tasks by reusing lessons already learnt, as well as previous experiences. In our model the agent in charge of this activity is the Searcher Agent, which searches in the knowledge base for the needed knowledge. Different techniques are currently used to search for knowledge. Many of them are based on the use of the position and frequency of keywords (Mohammadian & Jentzsch, 2004) or on information retrieval techniques (Frakes & Baeza-Yates, 1992; Liang & Huang, 2000). Other authors such as Sung Kim (2004) mix several techniques, data mining, and case-based reasoning to develop a recommender system.

d. *Knowledge Transfer* is the most investigated stage in knowledge management (Peachey, Hall & Cegielski, 2005). This stage is in charge of transferring tacit and explicit knowledge. Tacit knowledge can be transferred if it has been previously stored in shared means, for example, repositories, organizational memories, databases, and so forth. The transfer stage can be carried out by using mechanisms to inform people about the new knowledge that has been added. For this stage we propose a *Disseminator Agent*, which must detect the group of people or communities who generate and use similar information; for example, in the software domain, the people who maintain the same product or those who use the same programming language. Therefore, this agent fosters the idea of a community of practice in which each person shares knowledge and learns thanks to the knowledge of the other community members (Wenger, 1998). An appropriate knowledge management linked to communities of practice helps to improve the organization's performance (Lesser & Storck, 2001). Disseminated information may be of different types; it may be information linked to the company's philosophy or specific information about a determined process. Finally, the Disseminator Agent needs to know exactly what kind of work each member of the organization is in charge of and the knowledge flows linked to their jobs. In order to do this, the disseminator agent contacts with a new type of agent called the *personal agent* which is in charge of determining the users' profiles (it will be described in next section). Comparing this stage with the SECI model we can say that the disseminator agent fosters the socialization process since it puts people who demand similar knowledge in touch and once in contact they can share their experience, thus increasing their tacit knowledge.

e. *Knowledge Evolution*. This stage is responsible for monitoring the knowledge that evolves daily. To carry out this activity we propose a *Maintenance Agent.* The main purpose of this agent is to keep the knowledge stored in the knowledge base updated. Therefore, information that is not often used is considered by the Maintenance Agent as information to be possibly eliminated.

MULTI-AGENTS AGENCIES

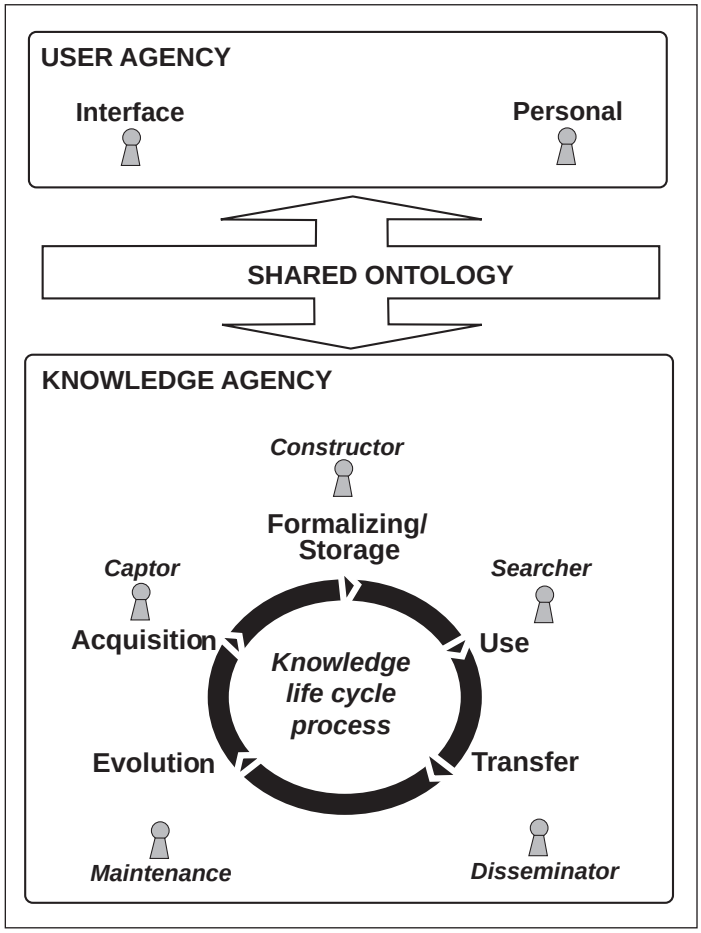
Once the model and the agents that we propose to give support to the different stages have been described, we are going to explain how the agents are structured into two agencies. Therefore, we group all the agents closely in charge of managing knowledge and supporting the different stages of the model proposed in one agency. Auxiliary agents are in another agency (see Figure 3).

Therefore, the *Knowledge Agency* is in charge of giving support to the KM process. It consists of the Constructor Agent, the Captor Agent, the Searcher Agent, the Disseminator Agent, and the Maintenance Agent.

On the other hand, the *User Agency* is formed of the Personal Agent and the Interface Agent. The Personal Agent monitors users' tasks to obtain their preferences and needs. In order to implement the Personal Agent, user modeling techniques can be used. User modeling implies obtaining certain knowledge about the user. This knowledge describes what the user "likes" or what the user "knows" (Chin, 1986).

The *Interface Agent* is the mediator between the users and the agents. Thus, when an agent

Figure 3. Agents distribution



wants to communicate a message to the user, the agent sends the message to the Interface Agent which shows it to the user.

Another component is the *Shared Ontology* which provides a conceptualization of the knowledge domain. The Shared Ontology is used for the consistent communication of the agencies.

In order to carry out the analysis and design of the agents involved we have followed a methodology called INGENIAS (Pavón & Gómez-Sanz, 2003) which provides metamodels to define multi-agent systems, and support tools to generate them. Using metamodels facilitates the development of systems enormously, since they are oriented towards visual representations of concrete aspects of the system.

Below, we are going to show the different agent meta-model diagrams which describe the roles and tasks of each agent.

Figure 4 shows that the goal of the Captor Agent is to obtain information that should be stored. Its role is "filter" since it must decide what information should be transformed into knowledge, the purpose being to use this in future projects. In the following lines, we describe each of the tasks carried out by this agent.

- **IdentifyIS:** This task consists of identifying available knowledge sources in the system.
- **CaptureInfo:** The agent must also capture information.
- **SendToConstructor:** Once the suitability of storing the information has been analyzed, the Captor sends it to the Constructor Agent (described in Figure 5) whose roles are sculptor and treasurer since it is in charge of giving an appropriate electronic format to the information (sculptor) and of storing it in the knowledge base (treasurer). The tasks developed by Constructor Agent are:
  - **CompareInfo:** The agent is in charge of comparing the new information with the previously stored knowledge.
  - **CombineInfo:** The agent is also in charge of combining the new information with the previously stored knowledge.
  - **ClassifyInformation:** Another task is to classify the information received by the Captor Agent (for instance: models, structures, files, diagrams, etc.).
  - **SendToDisseminator:** This is a critical task which consists of sending knowledge to the Disseminator Agent.
  - **SaveKnowledge:** One of the most important tasks is to store the new knowledge into the knowledge base.

The Disseminator Agent, whose role is PostOfficeEmplee, as it behaves the "postman" of

Figure 4. Captor agent diagram

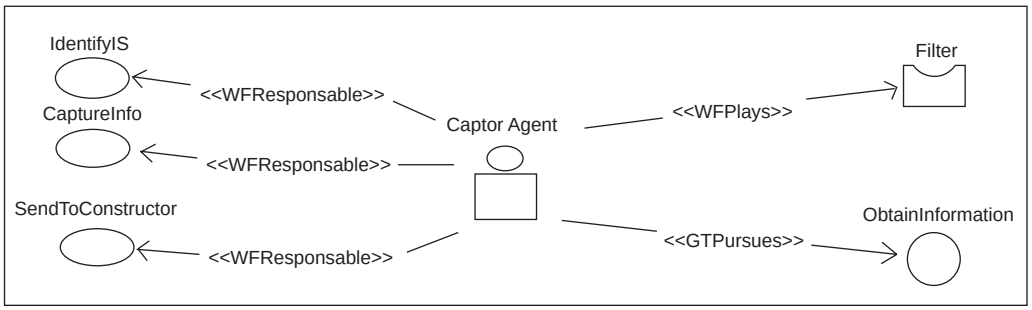


Figure 5. Constructor agent diagram

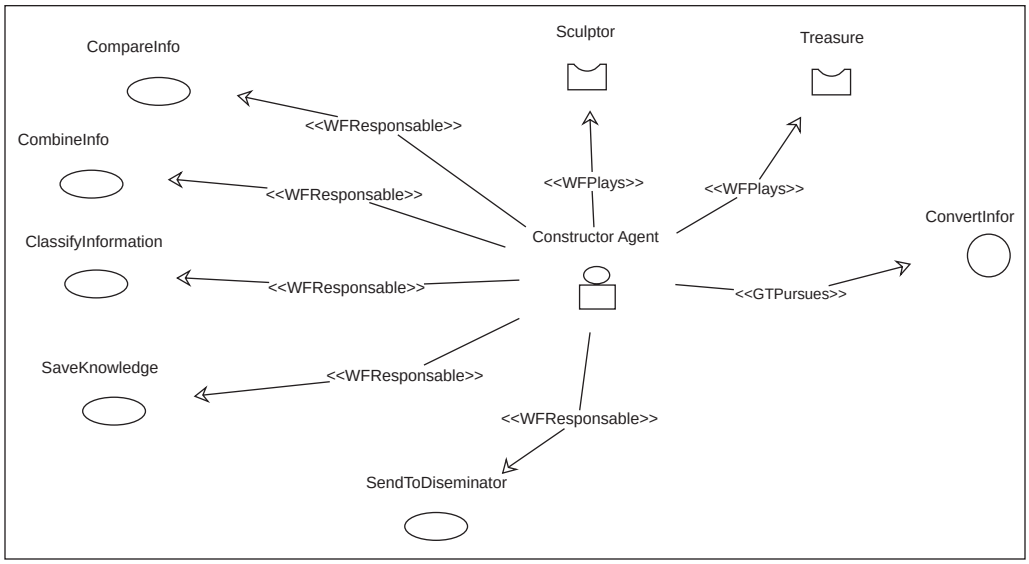
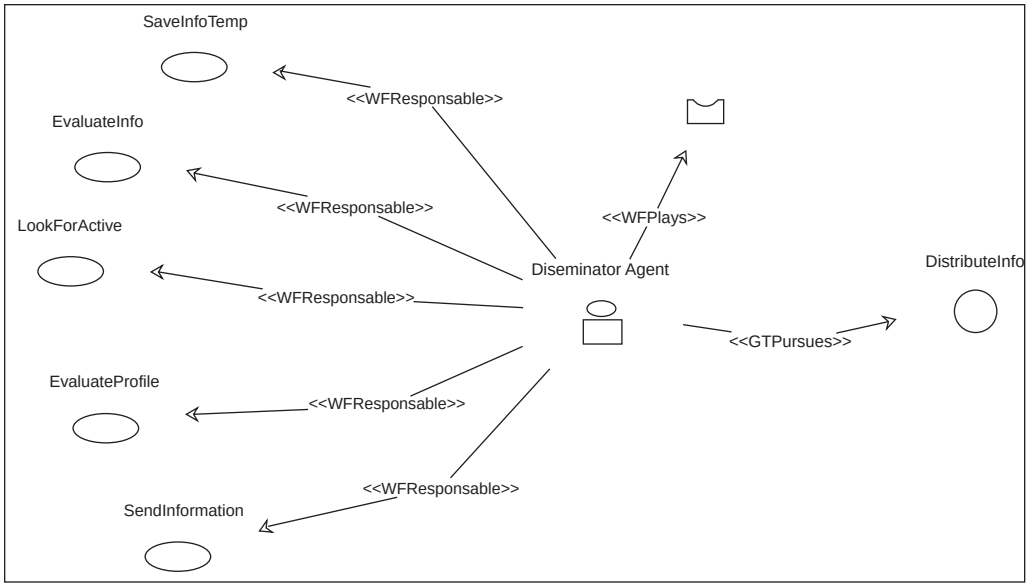


Figure 6. Disseminator agent diagram



the architecture, (see Figure 6) is composed of the next tasks:

- **SaveInfoTemp:** The Disseminator Agent stores temporally the new knowledge received by the Constructor Agent.
- **EvaluateProfiles:** Once identified one user profile, the Disseminator Agent evaluates it in order to determine user's needs.
- **LookForActivePersonalAgents:** Personal Agents can be distributed into different nodes, so it must identify all active Personal Agents available in the system.

- **SendInformation:** This is a critical task which consists of distributing the information to those people that can need it (really, the information is sent to their interface agents).
- **EvaluateInfo:** This task is focused on evaluating received information to be able to relate it with different user's profiles.

Another agent that supports the knowledge life cycle is the Searcher Agent. The goal of this agent is to foster the internalization process of the SECI model, since the employees have the opportunity of acquiring new knowledge by using the information that this agent suggests. The Searcher Agent diagram (Figure 7) is composed of the next tasks:

- **LookForInfo:** This agent is in charge of searching the information required by the users.
- **ClassifyInfo:** This agent also classifies the information found in the knowledge base.
- **SendInfoToI:** Finally, the agent sends the knowledge found in the knowledgebase to an Interface Agent.

Figure 7. Searcher agent diagram

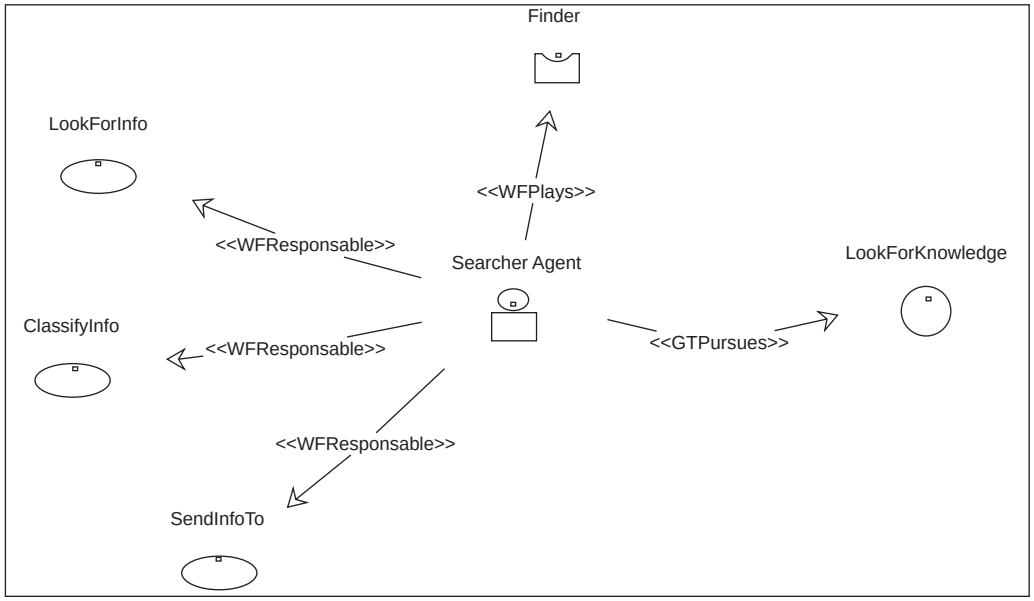


Figure 8. Maintenance agent diagram

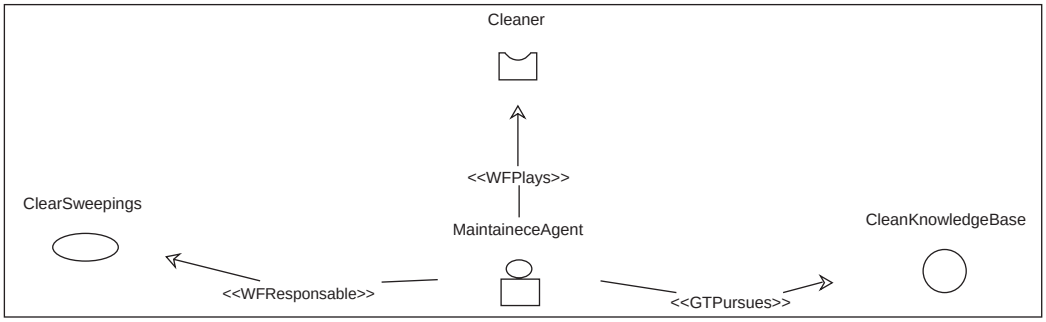


Figure 9. Personal agent diagram

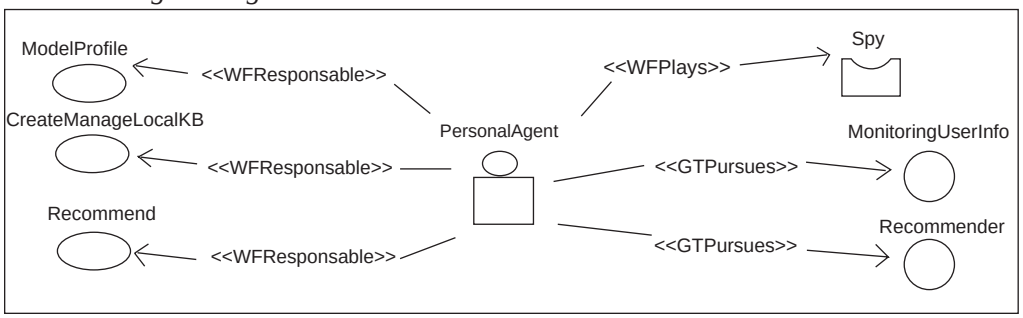
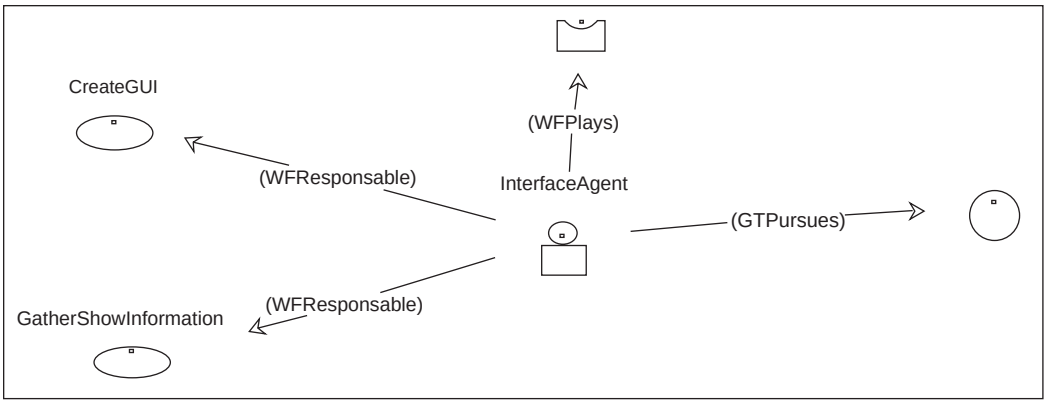


Figure 10. Interface agent diagram



The last type of agent of the Knowledge Agency is the Maintenance Agent (Figure 8). The main purpose of this agent is to keep the knowledge stored in the knowledge base updated. Therefore, its task dealt mainly with deleting obsolete information.

Now, the two types of agents of the User agency are described. Figure 9 shows the Personal Agent diagram whose role is called "spy" since the agent must monitor users' activities in order to obtain their profiles. Therefore, its goals are monitoring users' tasks and recommending information.

In order to attain these goals it should carry out the following tasks:

- **Modeling the users' profiles:** By observing the users' preferences, activities, information consulted, and so forth.
- **CreateManageLocalKnowledgeBase:** Creating and managing a "local knowledge base" where the relevant information for the user can be stored.
- **Recommending knowledge or knowledge sources:** This agent tries to guess what knowledge would be relevant for the user. To accomplish this, this agent communicates with the Searcher Agent and with the Interface agent.

On the other hand, the Interface Agent is an intermediary between the users and the rest of agents. Figure 10 shows that its main tasks are creating GUI and showing information to the users

These tasks are defined in order to attain the goal of showing important information to the

user, named in the diagram ShowInformation, so we have to create an user interface and put the received information from others agents in a nice way to the user.

A PROTOTYPE SYSTEM

In order to test our model we are developing a KMS to be used in software maintenance companies. So far, the prototype recommends what information sources maintainers should consult to solve a particular problem. Before constructing the prototype, the knowledge flows that take place in software maintenance companies were studied (Rodríguez, Martínez, Vizcaíno, Favela & Piattini, 2005). To illustrate how the prototype works let us describe a scenario.

Scenario

A software maintenance engineer selects the project to be work on. Then, the employee starts to work on an activity (for instance a maintenance request). At the same time, the Personal Agent is monitoring the engineer's movements and is logging in what project and activity the engineer is working on. So, the Personal Agent sends the Searcher Agent a message asking for knowledge related to the activity that the employee is carrying out. Depending on the activity, the Searcher Agent can use two retrieval techniques, position and frequency of keywords in the case of needing to give information about a topic, or case-based reasoning in the case of having to propose a solution to a problem. When the Searcher Agent finds suitable information, the agent sends it to the Interface Agent, which is in charge of communicating to the employee that certain information exists which can be useful for the employee's work. The employee will decide if to consult this information. Figure 11 despites the diagram of this part.

Once the employee finishes the work, the Captor Agent checks whether a new case can be constructed (in case the employee had found a solution to a problem) or whether a new knowledge source has been used. In both cases the Captor sends the new knowledge to the Constructor Agent which is in charge of storing this in the knowledge base or adding new concepts to the knowledge source ontology according to the circumstance that have taken place.

The collaboration between the Captor and the Constructor Agent is depicts in Figure 12, which is an interaction model diagram that the INGENIAS methodology utilizes. These diagrams are very useful to see, at first glance, as agents interact.

Some Implementation Aspects

The platform that we are using to develop the architecture is java agent development framework (JADE) since it is FIPA compliant and is currently one of the most widely used. Moreover, JADE has been successfully used in the development of other systems in the domain of knowledge management (Bergenti, Poggi & Rimassa, 2000; Gandon, 2000).

RELATED WORK

Traditional KM systems have received certain criticism, since employees are often overloaded with extra work as they have to introduce information into the KMS and worry about updating this information. One proposal to avoid this extra burden was to add software agents to perform this task in place of the employees. Later, intelligent agent technology was also applied to other different activities, bringing several benefits to the knowledge management process.

The benefits of applying agent technology to knowledge management include distributed system architecture, easy interaction, resource

Figure 11. Scenario diagram

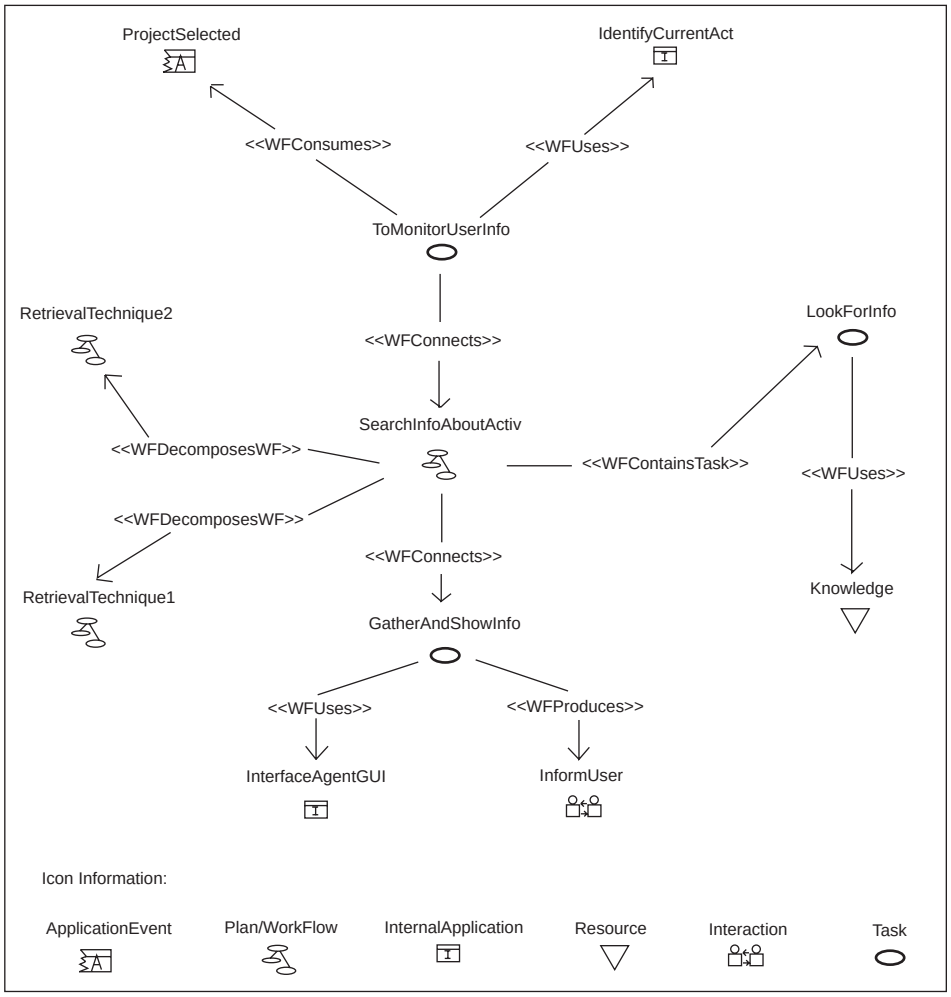
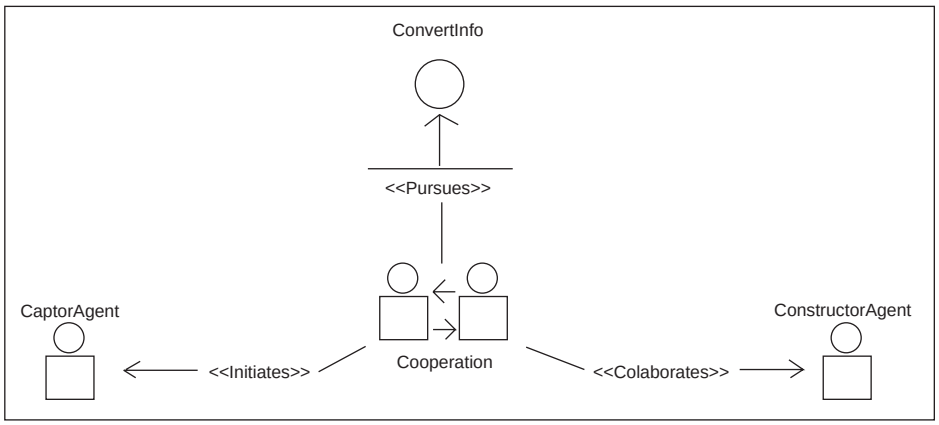


Figure 12. Cooperation between captor and constructor agent



management, reactivity to changes, interoperation between heterogeneous systems, and intelligent decision making. The set of knowledge management tasks or applications in which an agent can assist is very wide, for instance:

- To manage organizational memory, an example being the CoMMA project, (Gandon, 2000) (Corporate Memory Management through Agents), which combines emergent technologies, allowing users to exploit an organizational memory.
- To support cooperative activities. For instance Wang, Reidar, and Chunnian (1999) propose a multi-agent architecture to provide support to cooperative activities.
- To recommend. For instance Sung Kim (2004) describes a system to customize recommendations.
- To find experts. Some systems are used to help people find experts who can assist them in their daily work.
- To share knowledge. For instance Mercer and Greenberg (2001) propose a multi-agent system for knowledge sharing in a system designed to advise good programming practice.
- To manage mailing lists, or document classification (Moreale & Watt, 2003).

These and other existing systems were often developed without considering how knowledge flows and what stages may foster these flows. Because of this, they often support only one knowledge task, without taking into account that knowledge management implies giving support to different process and activities. On the other hand, KM systems often focus on the technology, without taking into account fundamental problems that these kinds of systems are likely to support (Hahn & Subramani, 2000).

CONCLUSION

The main contributions of this article are the design of knowledge cycle for developing KMS where the main functions that this kind of systems must support are described. Moreover, a multi-agent architecture is outlined to help KMS developers to implement these kinds of systems. The advantages of these contributions are:

- The model provides support to different activities: knowledge creation, storage/retrieval, transfer, and application. All are activities which, according to the authors who specialize in evaluating KMS, should support this kind of system.
- The architecture is based on a KM life cycle that we have proposed for this end. Therefore, we try to avoid the lack of other architectures that are focused on the technology and forget the knowledge aspects.
- The architecture makes use of intelligent agents. This is a technique that have proved to be very convenient in knowledge management activities since it avoids one of the problems of some KMS such as overloading the employees with extra work instead of helping them during their daily work. Agents can carry out many tasks on behalf of users. Moreover, they act when they consider that it is necessary to do so without needing users' instructions. Another advantage of using agents is that they can collaborate with other agents already implemented to carry out concrete knowledge tasks; for instance, obtaining information from the Internet or from e-mail. Thus, the development of KMS would be easier since only the basic agents of our model would have to be implemented and these could collaborate with other agents that have already been tested.

On the other hand, we are modeling the agents in a systematic way by using INGENIAS methodology whose metamodels help future developers to understand how the different agents work.

As future work we aim to compare the implementation of a KMS based on our proposal with developments using other architectures. Without any doubt this evaluation will help us to improve our proposal. On the other hand, we are also working on extending the model documentation with a more wide and detailed description of the possible techniques that could be used to implement each type of agent according to the main needs that organizations usually demand.

From a technological point of view, we are also studying JADEx in order to see how easy it would be to migrate to this new platform.

ACKNOWLEDGMENT

An earlier version of this article was presented at the 40th Hawaii International Conference on System Sciences (HICSS-2007), 03-06 January 2007, Waikoloa, Big Island, Hawaii.

This work is partially supported by the ENIGMAS (PIB-05-058), and MECENAS (PBI06-0024) project, Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, both in Spain. This is also supported by the ESFINGE project (TIN2006-15175-C05-05) Ministerio de Educación y Ciencia (Dirección General de Investigación)/Fondos Europeos de Desarrollo Regional (FEDER) in Spain.

REFERENCES

Alavi, M., & Leidner, D. E. (2001). Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly, 25*(1), 107-136.

Bergenti, F., Poggi, A., & Rimassa, G. (2000). *Agent architecture and interaction protocols for corporate memory management systems.* Paper presented at the ECAI'2000, Workshop on Knowledge Management and Organizational Memories.

Camacho, D., Aler, R., & Cuadrado, J. (2004). Rule-based parsing for Web data extraction. In M. Mohammadian (Ed.), *Intelligent Agents for data mining and information retrieval* (pp. 65-87). Hershey, PA: Idea Group Publishing.

Chin, D. (1986). User modelling in UC: The UNIX consultant. *Human factors in computing systems.*

Davenport , P. (1998). *Working knowledge.*

Davenport, T. H., & Prusak, L. (1998). *Working knowledge: How organizations manage what they know.* Boston: Harvard Business School Press.

Dickinson, A. (2000). Enhancing knowledge management in enterprises (ENKE) IST project, IST-2000-29482. Retrieved April 27, 2007, from http://www.ist-enke.com

Frakes, W. B., & Baeza-Yates, R. (1992). *Information retrieval data structures and algorithms.* Englewood Cliffs, NJ: Prentice Hall.

Gandon, F. (2000). *A multi-agent architecture for distributed corporate memories.* Paper presented at the Third International Symposium From Agent Theory to Agent Implementation in European Meeting on Cybenertics and Systems Research.

Giannella, C., Bhargava, R. & K. H. (2004). *Multi-agent systems and distributed data mining.* Paper presented at the Cooperative Information Agents VIII: 8th International Workshop (CIA'04), Erfurt, Germany. Springer-Verlag.

Gkotsis, G., Evangelou, C., Karacapilidis, N., & Tzagarakis, M. (2006). *Building collaborative knowledge-based systems: A Web engineering approach.* Paper presented at the WWW/Internet, Iadis International Conference, Murcia.

Gruber, T. R. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition, 5*(2), 199-220.

Hahn, J., & Subramani, M. (2000). *A framework of knowledge management systems: Issues and challenges for theory and practice*. Paper presented at the 21st International Conference in Informatin Systems (ICIS'00) (pp 302-312), Brisbane.

Houari, N., & Homayoun Far, B. (2004). *Application of intelligent agent technology for knowledge management integration.* Paper presented at the Third IEEE International Conference on Cognitive Informatics (ICCI'04).

Jennex, M. E. (2005). *The issue of system use in knowledge management systems.* Paper presented at the 38th Hawaii International Conference on System Sciences.

Jennex, M. E., & Olfman, L. (2006). A model of knowledge management success. *Journal of Knowledge Management, 2*(3), 51-68.

Lesser, E. L., & Storck, J. (2001). Communities of practice and organizational performance. *IBM Systems Journal, 40*(4), 831-841.

Liang, T. P., & Huang, J. S. (2000). A framework for applying intelligent agents to support electronic trading. *Decision Support Systems, 28*(4), 305-317.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM, 37*(7), 31-40.

Malone, D. (2002). Knowledge management: A model for organizational learning. *International Journal of Accounting Information Systems, 3,* 111-123.

Mercer, S., & Greenberg, S. (2001). *A multi-agent architecture for knowledge sharing.* Paper presented at the Sixteenth European Meeting on Cybernetic and Systems Research, Viena.

Mohammadian, M., & Jentzsch, R. (2004). Computational intelligence techniques driven intelligent agents for Web data mining and information retrieval. In M. Mohammadian (Ed.), *Intelligent agents for data mining and information retrieval.* Hershey, PA: Idea Group Publishing.

Moreale, E., & Watt, S. (2003). An agent-based approach to mailing list knowledge management. *Agent-mediated knowledge management.*

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science, 5*(1), 14-37.

Nonaka, I., & Takeychi, H. (1995). *The knowledge-creation company: How Hapanese companies create the dynamics of innovation*. New York: Oxford University Press.

Novak, J., Wurst, M., Fleschmann, M., & Strauss, W. (2003). Discovering, visualizing and sharing knowledge through personalized learning knowledge maps. *Agent-mediated knowledge management.* Standford.

Pavón, J., & Gómez-Sanz, J. J. (2003). *Agent oriented software engineering with INGENIAS*. Paper presented at the Multi-Agent Systems and Applications III: CEEMAS 2003 (LNAI), Prage, Czech Republic. Springer.

Peachey, T., Hall, D., & Cegielski, C. (2005). Knowledge management and the leading information systems journals: An analysis of trends and gaps in published research. *International Journal of Knowledge Management, 1*(3), 55-69.

Rhem, A. J. (2006). *UML for developing knowledge management systems.* New York: Auerbach Publications.

Rodríguez, O., Martínez, A. I., Favela, J., Vizcaíno, A., & Piattini, M. (2004). *Understanding and supporting knowledge flows in a community of software developers.* Paper presented at the 10th International Workshop on Groupware

(CRIWG 2004) LNCS 3198, San Carlos, Costa Rica. Springer.

Rodríguez, O. M., Martínez, A. I., Vizcaíno, A., Favela, J., & Piattini, M. (2005). Identifying knowledge flows in communities of practice. In E. Coakes & S. A. Clarke (Eds.), *Encyclopedia of communities of practice in information and knowledge management.* Hershey, PA: Idea Group Publishing.

Rus, I., & Lindvall, M. (2002, May/June). Knowledge management in software engineering. *IEEE Software,* 26-38.

Sung Kim, J. (2004). Customized recommendation mechanism based on Web data mining and case-based reasoning. In M. Mohammadian (Ed.), *Intelligent agents for data mining and information retrieval*..

Tautz, C., & Von Wangenheim, C. G. (1998, October 20). *REFSENO: A representation formalism for software engineering ontologies* (Fraunhofer IESE-Report Nº 015.98/E, version 1.1).

Tiwana, A. (2000). *The knowledge management toolkit: Practical techniques for building a knowledge management system.* Prentice Hall.

van Elst, L., Dignum, V., & Abecker, A. (2003). Agent-mediated knowledge management. *Agent-mediated knowledge management.* Standford.

Wang, A., Reidar, C., & Chunnian, L. (1999). *A multi-agent architecture for cooperative software engineering.* Paper presented at the Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99), Germany.

Ward, J., & Aurum, A. (2004). *Knowledge management in software engineering: De-scribing the process.* Paper presented at the 15th Australian Software Engineering Conference (ASWEC 2004), Melbourne, Australia. EEE Computer Society Press.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity.* Cambridge, U.K.: Cambridge University Press.

Wiig, K. M. (1997). Knowledge management: Where did it come from and where will it go? *Expert Systems with Applications, 13,* 1-14.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review, 10*(2), 115-152.

This work was previously published in International Journal of Knowledge Management, Vol. 3, Issue 4, edited by M. Jennex, pp. 67-83, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.19
Human-Centered Design of a Semantically Enabled Knowledge Management System for Agile Software Engineering

Christian Höcht
Technical University of Kaiserslautern, Germany

Jörg Rech
Fraunhofer Institute for Experimental Software Engineering (IESE), Germany

ABSTRACT

Developing human-engineered systems is considered as a challenge that addresses a wide area of expertise; computer scientists as well as social scientists. These experts have to work together closely in teams in order to build intelligent systems to support agile software development. The methodology developed in the RISE project enables and supports the design of human-centered knowledge-sharing platforms, such as Wikis based on standards in the field of education science. The project "RISE" (Reuse In Software Engineering) is part of the research program "Software Engineering 2006" funded by the German Federal Ministry for Education and Research (BMBF). The goal was to improve the reuse of artifacts in software engineering, and brought together researchers from education science (The Department of Educational Sciences and Professional Development at the Technical University of Kaiserslautern) and computer science (Fraunhofer Institute for Experimental Software Engineering (IESE) and the German Research Center for Artificial Intelligence (DFKI)) with industrial partners (Empolis GmbH and brainbot technologies AG). This chapter gives an overview about the human-centered design of Wiki-based

knowledge and learning management systems in software engineering projects, and raises several requirements one should keep in mind when building human-centered systems to support knowledge and learning management.

INTRODUCTION

The development of complex software systems is based on company- and domain- specific knowledge that has to be constantly cultivated among the employees, because the resulting quality of manufactured software systems depends on what degree the needed knowledge is actually available (Decker, Ras, Rech, Klein, Reuschling, Höcht, & Kilian, 2005). It is not only that the technical platform should release software engineers as much as possible from time-consuming retrieval-processes. At the same time the platform acquires valuable pieces of information from users, who publish their problems and experiences during work. Out of this process, company-based knowledge may be built and refined.

However, it is just because of the various possibilities of searching and browsing through artifacts that users feel overwhelmed by the flood of information. Therefore, the increasing amount of information itself is not the problem, but an unfiltered and unrated access to it. In fact, the main goal is to ensure that software engineers can deal with their daily tasks without burdening them with additional work. A systematic selection and presentation of content helps to avoid the feeling of being swamped with artifacts and so keeps, and even improves, the employees' motivation. For example, on the one hand, it is a good idea to offer plugins that provide visual representations of concepts, but on the other hand, offering totally unrestricted overviews is often too complex to be processed by the users (Tonkin, 2005).

Ballstaedt (1997) mentions four main types of artifacts considering textual content. As Wikis mainly consist of text, this typology is helpful to

Figure 1. Different types of content

	Description	Example	Challenges in Knowledge Management related to Software Engineering
expository text	Expository text describes facts, explains the context, and thus provides conceptual knowledge.	dictionary entry	Information related to software products, solutions, or methods might expire very quickly, and would thus require much effort to maintain it.
narrative text	Narrative text reports actions or plots and events. It informs about specific situations, motives, decisions, acts, and their consequences.	blog entry	Narrative text might be rather helpful for some employees, but probably just a waste of time for others, because the individual usefulness of a specific narrative text is hard to evaluate.
instructional text	Instructional text provides procedural knowledge and enables people to do or not to do something.	help on Installation or maintenance	Providing useful instructional text assumes that the author respects different levels of knowledge and writes adequate content, keeping a specific target group in mind.
additional didactic text	Additional text that supports specific learning activities.	advance organizer	Additional didactic content requires deeper knowledge about reading and learning strategies and depends on rather static content like printed material.

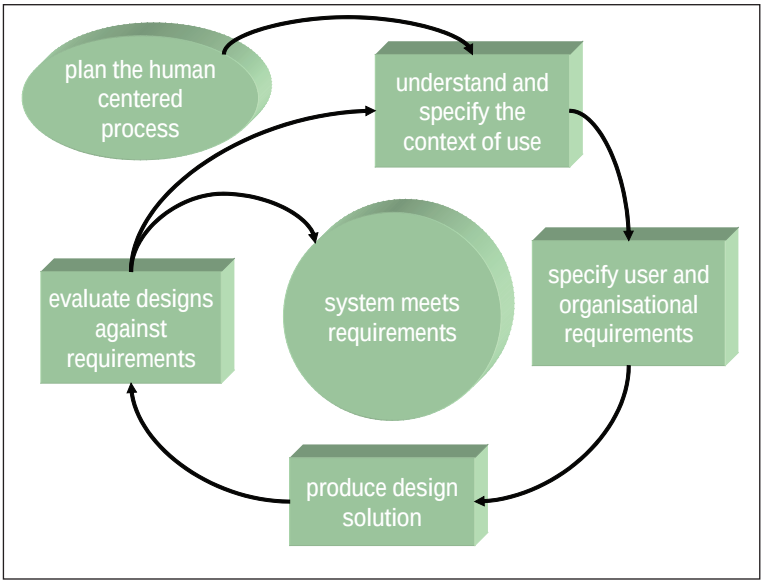
distinguish between different textual instances, as seen in Figure 1.

From the pedagogical point of view, artifacts in a knowledge management (KM) system are considered as not more than a faint representation of knowledge. In order to communicate knowledge to others, its complexity has to be reduced. This can be done, on the one hand, by picking out only those aspects that are necessary to deal with the artifact. The main idea of that principle is that these selected aspects allow one to indicate the underlying complexity. This can be done by letting the users assign adequate metadata to the given concept. Several methods to support the users by means of an intelligent application are discussed later in the chapter. But it has to be critically remarked that the validity of this assumption depends on the type of knowledge that has to be communicated and the given types of content.

This approach itself, which concentrates on essential aspects concerning a concept, does not yet allow other users to adopt the given artifact. It is rather necessary to restructure the chosen aspects in a different way. This method considers the previous knowledge as well as the interests of the targeted users, provided by semantics based on metadata, concept structures, and user profiles. These requirements can be addressed by means of a semantically enabled Wiki that, for example, arranges artifacts in a flexible way depending on the users' needs (e.g., artifacts written/read by the user and given interests or tasks of the user). Besides that, searching the whole system for certain artifacts should present results in a nested or context-based way (e.g., the artifact describing "Pair Programming" should be arranged under the parent concept "Extreme Programming"). This solution is based on underlying ontologies, as discussed later in the chapter. After the whole process of selecting certain aspects and restructuring them, users may be provided with complete, self-contained and motivating knowledge.

Based on the principles of a human-centered design process according to ISO 13407, expected users take an active part in the development process in order to get a clear idea of what might be specific requirements (ISO 13407, 1999). The early

Figure 2. Human-centered design processes for interactive systems (Source: ISO 13407, 1999)



phase of the German research project, "RISE," therefore was dominated by context-analyses within the participating companies. Qualitative-centered studies delivered deep insight into organizational aspects and users' special needs. Two focus groups dealt with the drawbacks and opportunities of using conventional Wiki systems that were formerly introduced by the associated companies. Not only the focus groups, but also further analyses of existing artifacts allowed the allocation of functions among the technical system and respectively, the users themselves.

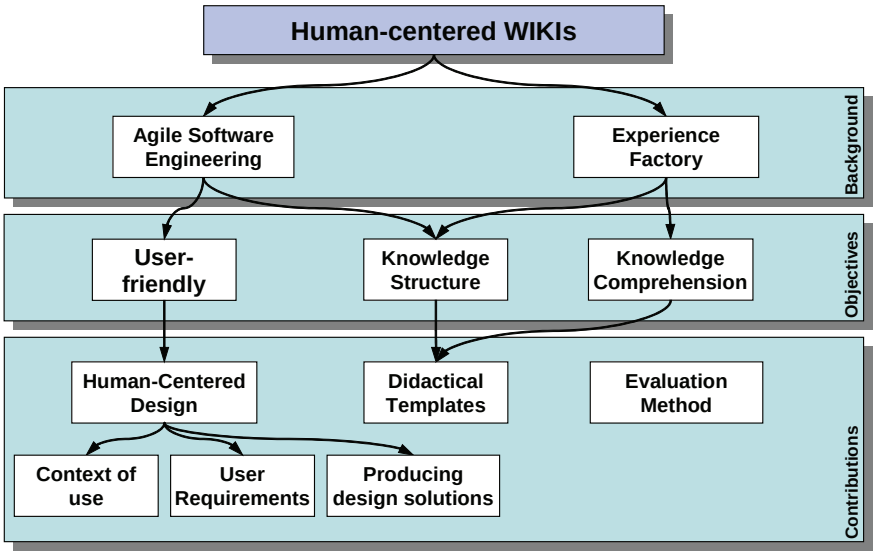
For example, the analysis of individually built file structures showed that an automatically enabled structuring of existing artifacts might be critical because people tend to use structures in different ways (hierarchical, time-based, priority-based, people-based, etc.) and with different granularity. This includes accessing existing artifacts as well as creating new ones. Teevan et al. discovered in a comparable study, for example, that people even use hierarchical organized structures if they do not maintain their individual structures hierarchically (Teevan, Alvarado, Ackerman, & Karger, 2004). This example shows, rather clearly, how important it is to allocate functions between users and intelligent systems. But it also gets clear that definite answers could not be found until final tests of the application.

As depicted in Figure 3, we elaborate in the next section on the background regarding this chapter. This includes information on agile software engineering and software reuse based on the experience factory approach. Thereafter, in "Human-Centered Design in Software Developing SMEs," we describe the relevant objectives of the RISE project, as well as our contribution to human-centered design of Wikis in software engineering. Finally, we describe our evaluation method and results from the project evaluation.

BACKGROUND

Centering the user during the software development process is especially important in engineering knowledge and learning management (KLM) as well as social software systems (e.g., Web 2.0)

Figure 3. Outline and structure of the following chapter



where users freely publish and consume information. Their efficient as well as effective interaction with the technical system gets more and more crucial for the overall success of the software product. The integration of users and users groups into the software systems is steadily increasing. This is also true in development methods such as extreme programming (i.e., an agile method), where the team members and the customer of a software system are deeply integrated to develop the envisioned software system. This section gives an overview about agile software engineering, as well as existing knowledge management approaches (i.e., the experience factory) that build the basis for collaborative human-centered software development.

Software Engineering and Reuse

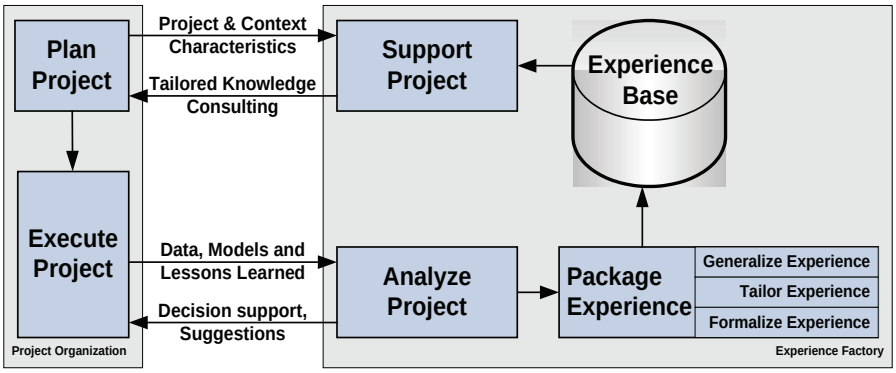
Software engineering (SE) as a field in computer science is concerned with the systematic development of high-quality software systems. During the planning, definition, development, and maintenance of software systems, the people involved generate and require any information and knowledge to support them in their work or to back up their decisions.

This reuse of existing knowledge and experience is one of the fundamental parts in many sciences. Engineers often use existing components and apply established processes to construct complex systems. Without the reuse of well-proven components, methods, or tools we had to rebuild and relearn them again and again.

In the last thirty years, the fields software reuse and experience management (EM) are increasingly gaining importance. The roots of EM lie in experimental software engineering ("Experience Factory"), in artificial intelligence ("Case-Based Reasoning") and in knowledge management. EM is comprised of the dimensions methodology, technical realization, organization, and management. It includes technologies, methods, and tools for identifying, collecting, documenting, packaging, storing, generalizing, reusing, adapting, and evaluating experience knowledge, as well as for development, improvement, and execution of all knowledge-related processes.

The Experience Factory (EF) is an infrastructure designed to support experience management (i.e., the reuse of products, processes, and experiences from projects) in software organizations (Basili, Caldiera, & Rombach, 1994). It supports the collection, pre-processing, and dissemination of experiences of organizational learning, and represents the physical or at least logical separation of the project and experience organization, as shown in Figure 4. This separation is meant

Figure 4. The Experience Factory



to relieve the project teams from the burden to find, adapt, and reuse knowledge from previous projects, as well as to support them to collect, analyze, and package valuable new experiences that might be reused in later projects.

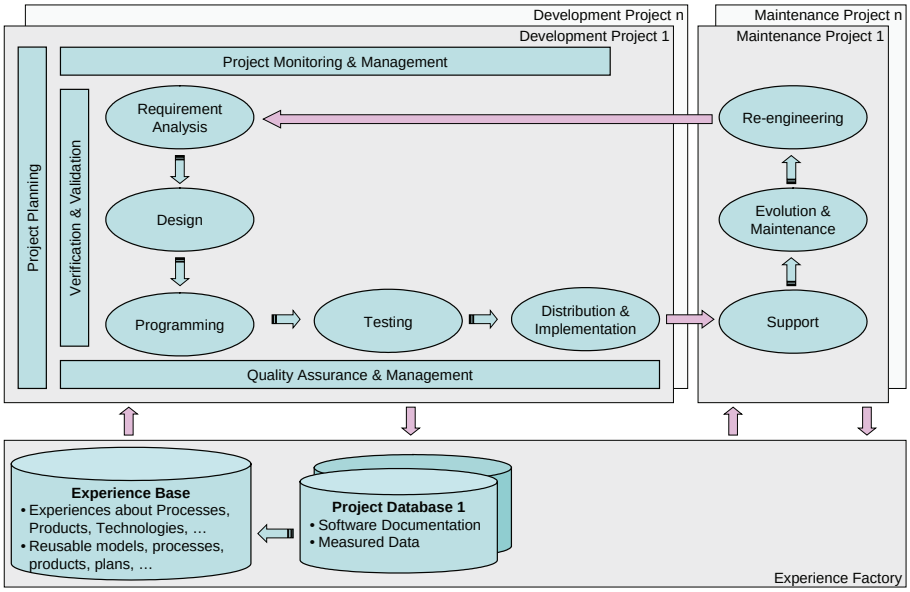
For example, if we begin a project ("Plan project"), we might use the experience factory to search for reusable knowledge in the form of architectures, design patterns, or process models based upon our project context. In the execution phase ("Execute project"), the EF is used to retrieve knowledge "on demand" (e.g., to support decisions or reuse source code) and at the projects end, it is analyzed (i.e., a post-mortem analysis) in order to extract reusable knowledge, for example, in form of project data, architectures, quality models, or other experiences that might be useful in other projects.

Traditional, Process-Oriented Software Development and Reuse

Since its beginning, several research directions developed and matured in field SE. Figure 5 shows the software development reference model integrating important phases in a software lifecycle.

- *Project engineering* is concerned with the acquisition, definition, management, monitoring, and controlling of software development projects, as well as the management of risks emerging during project execution.
- Methods from *requirements engineering* are developed to support the formal and unambiguous elicitation of software requirements from the customers, to improve the usability of the systems, and to establish a binding and unambiguous definition of the resulting systems during and after software project definition.
- The research for *software design and architecture* advances techniques for the development, management, and analysis of (formal) descriptions of abstract representations of the software system, as well as required tools and notations (e.g., UML).
- Techniques to support the professional *programming* of software are advanced to

Figure 5. Software development reference model



develop highly maintainable, efficient, and effective source code.
- *Verification and validation* is concerned with the planning, development, and execution of (automated) tests and inspections (formal and informal) in order to discover defects or estimate the quality of parts of the software.

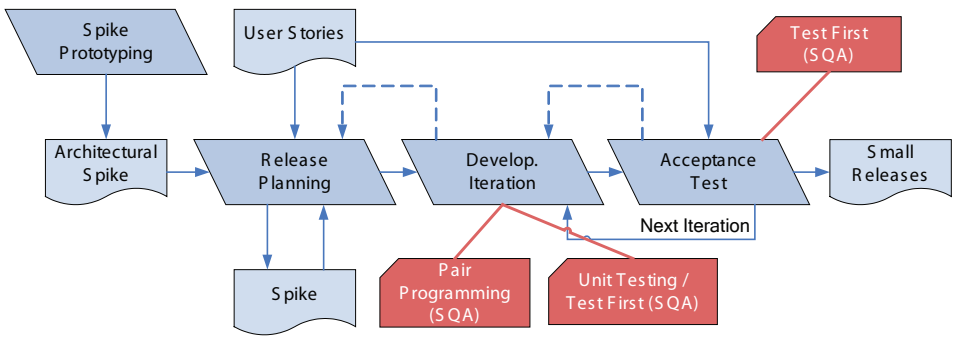
Research for implementation and distribution is responsible for the development of methods for the introduction at the customer's site, support during operation, and integration in existing IT infrastructure. After delivery to the customer, software systems typically switch into one of the following phases:

- In the *software evolution* phase, the focus of research lies on methods in order to add new and perfect existing functions of the system.
- Similarly, in the parallel phase, *software maintenance* techniques are developed for the adaptation to environmental changes, prevention of foreseeable problems, and correction of noticed defects.
- If the environment changes dramatically or further enhancements are impossible, the system either dies or enters a *reengineering* phase. Here, techniques for software understanding and reverse engineering of software design are used to port or migrate a system to a new technology (e.g., from Ada to Java or from a monolithic to a client/server architecture) and obtain a maintainable system.

Agile Software Development and Reuse

Beside the traditional or process-oriented software development another trend arose during the last years. *Agile software development* methods impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. Agile methods have in common that small releases of the software system are developed in short iteration in order to give the customer a running system with a subset of the functionality needed. Therefore, the development phase is split into several activities that are followed by small maintenance phases. In contrast to traditional, process-oriented SE where all requirements and use cases are elicited, the agile methods focuses on few essential requirements and incrementally develops a functional system in several short development iterations.

Figure 6. Agile software development (here the XP-Process)



Today, Extreme Programming (XP) (Beck, 2003) is the best-known agile software development approach. Figure 6 shows the general process model of XP that is closely connected to refactoring and basically its cradle (Beck & Fowler, 1999).

These agile methods (and especially extreme programming (XP)) are based upon twelve principles (Beck, 2003). We mention four of these principles, as they are relevant to our work. *The planning game* is the collective planning of releases and iterations in the agile development process and necessary to quickly determine the scope of the next release. Knowledge about the existence of existing code elements or subsystems relevant to the project can be used to plan the scope of the next release. *Small releases* are used to develop a large system by first putting a simple system into production and then releasing new versions in short cycles. The more an engineer can reuse, the faster his work is done and the quicker the customer gets feedback. *Simple design* means that systems are built as simply as possible, and complexity in the software system is removed, if at all possible. The more libraries are used and identified (via code retrieval), the less functionality has to be implemented in the real system. *Refactoring,* or the restructuring of the system without changing its behaviour, is necessary to remove qualitative defects that are introduced by quick and often unsystematic development. Decision support during refactoring helps the software engineer to improve the system.

Traditional software reuse initiatives and approaches that were developed for process-driven software development are inadequate for highly dynamic and agile processes where software cannot be developed for reuse and reuse cannot be planned in advance. Teams and organizations developing with agile methods need automated tools and techniques that support their work without consuming much time. Therefore, *agile software reuse* is a fairly new area where minimally invasive techniques are researched to support software engineers (Cinneide, Kushmerick, & Veale, 2004).

Typically, people in SME do not exactly know what additional knowledge and learning objects they need to support their work. The individual requirements of an employee in an SME will grow as they are working with a KLM system. Therefore, the agile approach to develop a software system; that is, incrementally developing the system with intense interaction with the user, is very important. The traditional approach of developing the system based on a final set of elicited requirements will probably result in an out-dated KLM system that does not fit the users needs.

HUMAN-CENTERED DESIGN IN SOFTWARE DEVELOPING SMES

Software engineering, in general, aims at engineering-like development, maintenance, adjustment, and advancement of complex software systems. One characteristic of software engineering compared with classical engineering lies in the prevalent immateriality and the complexity and abstractness of software resulting from it. For this reason Broy and Rombach (2002), for example, point out the difficulty to gain vital working experiences, and point at deficits in the field of further training. Although many companies apply (agile) methods of software engineering, the transfer of the underlying complex knowledge is still a crucial problem.

Challenges in Software Development

The development of complex software systems requires specific knowledge about technologies, tools, processes, standards, and so forth. The quality of the outcome depends strongly on how successful existing knowledge and expertise of the employees can be provided and actually applied.

Considering that challenging situation, the German research project, "RISE" (http://www.rise-it.info), aims at a holistic support of software engineering tasks. Software engineers should be enabled to deal with their tasks and thus act more professionally. Furthermore, the transfer of existing knowledge within software-related companies has to be optimized. This should be accomplished by systematic reuse of experiences, methods, and models in different application contexts.

The project "RISE" is part of the research program "Software Engineering 2006," funded by the German Federal Ministry for Education and Research (BMBF). Project partners are Dept. of Educational Sciences and Professional Development at the Technical University of Kaiserslautern, Empolis Arvato GmbH, brainbot technologies AG, Fraunhofer Institute for Experimental Software Engineering (IESE), and the German Research Center for Artificial Intelligence (DFKI).

The vision of "RISE" is to develop a usable knowledge-management application for software developers that makes fun and requires minimum effort. Software developers are gently supported during teamwork and their reuse of working experiences and knowledge. Consequently, the organization itself should profit from improved productivity increases.

Based on the already stated special problems and situations, research also focuses on how high the impact of an intranet-based platform could be, in order to ensure the individual capacity to do successful work through reuse of experiences, methods, and models. Thus RISE tries to find answers to the question, *what are the main requirements for a knowledge-management platform from the view of employees?* Inorder to keep complex knowledge manageable so that it can be used individually and updated during work.

It has to be clarified in what respect a technical platform may allow an individually focused view of the information needed (knowledge management) and might facilitate (usability) use of it.

The participating companies already used Wiki systems as our project started. They provided initial data for the empirical study in the early project phase. In our case, we had to deal with technically and functionally rather different Wiki systems, which can all be summarized under the wide term "social software." Such Web-based content-management systems allow users not only to access content, but also to add their own pieces of information or to edit already available artifacts with as little effort as possible.

The underlying rationale is based on a strongly decentralized way of editing content. Early results of the evaluation, however, showed that a sophisticated methodology is required to successfully implement semantic based "social software" at the intranets of software-firms, in order to ensure a broad acceptance. In summary, we observed that our industrial partners had the following problems with KLM systems:

- The organizations used Wiki systems that few people used due to missing structure, participation, and documentation guidelines.
- The people stored their information in their own data repositories, network drives, group directories, and so forth, with individual structures. This made the discovery and goal-oriented retrieval of important knowledge nearly impossible.
- The organizations used several other tools such as content management systems (CMS), e-mail, versioning systems, and chat tools (i.e., ICQ) that comprised valuable information, all unconnected and without a common structure or template support.

As observed, acceptance might decrease rapidly if the application system no longer represents a source of information that is usable, as described within the international standard ISO 9241-11: "*extend to which a product can be used by specified users to achieve specified goals*

with effectiveness, efficiency and satisfaction in a specified context of use" (ISO 9241-11, 1998). So, if the systems fail to be usable, neither contents are published or edited nor even read in the worst case. The communication amongst software engineers will not be intensified, as intended, but stagnates to a total deadlock.

Understanding Context-of-Use and Defining User Requirements

It seems to be obvious that employees are responsible for an individual development of a vital knowledge basis. Not only that, this requires special abilities, resources, and strategies. Moreover, employees increasingly have to gather information and subsequently learn new things besides their working day, which is dominated by different peaks.

One kind of peak is, of course, caused by strict deadlines that temporarily raise the working load to a maximum. Besides those hard and rather ordinary kinds of peaks, there exists even a more elementary one: the problem-orientation that many software engineers cling to. One main hypothesis of the project "RISE" is that this kind of peak not only may provoke benefits, but also causes huge problems that eagerly demand a solution. The term "problem-orientation" refers to the assumption that software development predominantly is a rather trivial process: developers are provided with certain requirements and try to consider them while realizing software systems. But from time to time, that trivial process gets disturbed by rather tricky problems developers get stuck in.

Those more or less hard problems represent a kind of peak that has fatal consequences for knowledge management in software companies: developers that run into that peak are highly motivated to solve the problem and often try to solve it on their own. Probably, they spend hours and hours with their problem and do not bother about existing and proved solutions, concepts, or methods. In that case, reuse might not exceed accidental or sporadic access to available knowledge. Considering that the knowledge collected by means of a content management system, it seems to be almost impossible to build a shared repository. The result would be a collection of various problems (hopefully including a solution) that might resemble each other more or less. The so called "Experience Factory" approach has to be mentioned, which is considered as one answer to that challenge: "the Experience Factory approach was initially designed for software organizations and takes into account the software discipline's experimental, evolutionary and nonrepetitive characteristics" (Basili, Lindvall, & Costa, 2001, p. 1).

In addition to that, existing knowledge does not only grow rapidly, but can be accessed ever more easily. However, the technical feasibility itself does not actually pay off: employees might be overwhelmed by the huge amount of information they can access which suggests to them, despite (and just because of) various and omnipresent search possibilities, the feeling of just being swamped with artifacts.

Basili et al. (2001) see that problem too, and demand that the company has to *unload* its experts: "Experts in the organization have useful experience, but sharing experience consumes experts' time. The organization needs to systematically elicit and store experts' experience and make it available in order to unload the experts" (Basili et al., 2001, p. 2).

One-sided approaches to knowledge management often focus on gathering as much information as possible from experienced employees in order to preserve it. From an educational point of view, however, it neither seems to be reasonable nor possible to store that potentially relevant knowledge with its unreduced complexity if knowledge has to enable the individual capacity to act professionally as a software developer (Renzl, 2004, p. 36).

In fact, it is rather important to support the individual capacity to act professionally of each

developer systematically. It is assumed that this individual ability will get more important in future. Employees will be faced with even more complex working conditions. Customized approaches to knowledge management for different kinds of enterprises should start just from the special abilities and needs of the employees, and thus help to prepare for that situation.

It is assumed that, by means of an adequate selection and presentation of content, the feeling of being swamped with artifacts can be reduced, and the software developers' motivation can be ensured or even be raised. This form of knowledge management centers the affected subjects. Thus, employees with huge expertise are no longer considered like a means to an end but, in fact, the technical knowledge management system.

- *First*, this application system acts as a flexible and customizable view (like a window) on the information needed during working-time. This system strongly relieves software engineers from time-consuming retrieval activities.
- *Second*, the system supports the collection and storage of vital working experience, which has to be subsequently formed and refined.

These are the two main factors of the RISE methodology, which were optimized during project time.

Producing Design Solutions and Evaluating Them Against Requirements

The survey that was dominated by qualitative social research methods revealed valuable insights about requirements of users and organizations. Two focus groups were dedicated to explore usage problems with the knowledge management platforms already introduced.

Especially, the collection of user requirements, related to Wikis as experience management systems for software engineering, important problem areas could be identified.

The gathered qualitative data is characterized by a very high validity, which means (given the background of user-centered design) that a common understanding of user requirements was developed and involved users agreed to high extent which user requirements should be realized in favor.

In order to classify the collected data, the Munich knowledge-management model has been used with its four categories: knowledge-representation, knowledge-usage, knowledge-communication, and knowledge-creation (Reinmann-Rothmeier, 2001):

- **Knowledge-representation:** Some employees tend to write down a minimum of information. It was observed that some of them document just as little as possible, so that they will be able to find out later what was meant by the given artifact. The main disadvantage of that behavior is that the saved data becomes hard or even almost impossible to understand for other employees accessing that content because they do not have the necessary knowledge to »decode« and to classify it. However, it seems to be a kind of game those developers are playing when doing some documentation: they rather puzzle together pieces of information than writing down supposedly redundant or superfluous information.
- **Knowledge-usage:** Crucial information about projects, products, or customers are distributed across various databases. This means that useful knowledge is hard to find and thus hard to be transferred in other contexts like new projects for example.
- **Knowledge-communication:** Employees who browse or search through Wiki con-

tent in order to find some special piece of information have to spend a lot of time. Moreover, editing content in most Wikis is rather uncomfortable: for example, adding a chart, a table or even a picture to Wiki pages requires special handling if it is even possible. As a consequence, information is being transferred by mail under pressure of time and not added to the Wiki site.

- **Knowledge-creation:** With increasing activity in a Wiki, it becomes harder for authors to integrate their piece of information into the existing structure. So, it becomes more and more likely that no new content is being added and already saved content gets orphaned.

Basically, using metadata is a good idea to keep heterogeneous and complex structures describable and thus technically manageable. But the problem is that users often fail to provide useful metadata, or simply do not bother about complex sets of metadata.

One very important hypothesis of the RISE project is that software-development teams that use a Wiki collaboratively to build knowledge repositories are actually about to **create a kind of semantic structure**, which users may not necessarily be aware of. This more or less refined structure, called "Wikitology," can be considered as a weakly formalized ontology. This has three main advantages:

- The *maintenance of metadata* is not considered as extra activity.
- *Ontologies* may always grow behind the content.
- Maintenance of ontologies is an expensive and time-consuming process. But, based on our assumptions, this work can be *automated more efficiently,* and thus releases employees from that job.

Another crucial approach of the RISE project is the **use of tags** as a very minimalistic but rather effective way of providing metadata. Any tags may be assigned individually to WIKI pages. So, each user is able to create his own view on any artifact by using his own depiction. Since in our approach all users are able to see the tags provided by the others, it is, of course, very probable that team members manage to create a shared set of tags as well.

Furthermore, **templates** for special types of content (bug report, use case, user story, etc.) help to provide a minimum of required information. Readers as "consumers" of the artifact can be satisfied, too. For example, the given structure helps them to browse faster through the content in order to find only some piece of information being of interest. Finally, this results in a higher acceptance of the system, which might also motivate employees to add their content to the repository.

Additionally, a **blog component** is added to the system. The blog contains not only news. In fact, the blog can be considered as a human-based *filter* on the Wiki content. For example, users might link to certain Wiki pages and provide some extra information along with the link. So, other users get useful assistance with the evaluation of artifacts.

EVALUATION OF SEMANTICALLY ENABLED KM SYSTEMS

The objective of the evaluation of a KM system, such as the Riki, is to show its effectiveness in a specific application context. The tailored instances of the system at different organizations are evaluated to identify the usefulness to the users, the examined applicability, the evolvability, as well as economic factors. The evaluation was split into two phases. The first phase, called *baseline*

evaluation, at the start of the project was used to elicit the context and current state of knowledge transfer. From the information we gathered in this phase, we designed and developed the Riki system considering people, processes, and available technologies. The second phase, called *delta evaluation,* at the end of the project helped to evaluate the change on the socio-technical knowledge management system. In both phases, we used the following three techniques to elicit valuable information from the organizations and potential end users:

- Goal-oriented, questionnaire-based *interviews* were used to query the previously listed questions with three to ten persons in two to four sessions. The collected answers were summarized and validated by the participants via e-mail.
- *Group discussions* were done at every company to collect any additional information, opinions, ideas, and so forth, that were not covered by the interviews. The discussion was started with a specific topic (e.g., Why did the old Wiki not work for you as a KM system?), and every person could state what they expected from an improved knowledge-management infrastructure.
- *Artifact analyses* were conducted to identify knowledge sources, the type of knowledge within, as well as how employees structure their documents and knowledge in existing storage systems (e.g., the hierarchy of directories in personal file systems or in pre-existing WIKI systems).

These evaluation techniques helped to cover the following three topics:

- **Technology:** Elicitation of the existence and characteristics of the technical infrastructure, existing KM systems, and other software systems that might be integrated into or used as the KM system (i.e., the Riki system). Furthermore, these technological systems potentially have valuable information that can be utilized in a KM system.
- **Methodology:** Elicitation of the applied methodology for production (e.g., software development) and knowledge management (esp. knowledge transfer processes). This gives further information about how the KM system should be integrated into the social system of the organization and where, when, and by whom knowledge is produced or consumed.
- **Knowledge:** Elicitation of the existing knowledge components available in the organization, as well as their characteristics and interrelation (e.g., for the development of an ontology). Furthermore, the typical structure of documents that might be didactically enriched.

A more detailed description, as well as some results of the two evaluation phases, is described in the following sections. They address persons who want to evaluate a KM system, such as the RIKI in an organization.

Baseline Evaluation

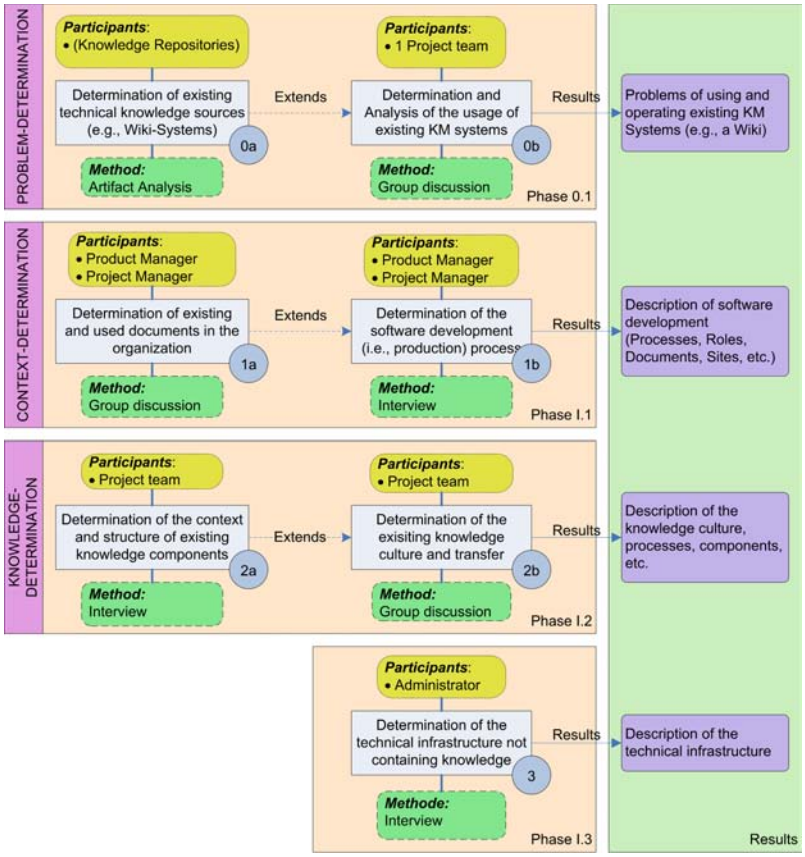
The baseline evaluation is concerned with the determination of the organizational context a social-technical knowledge management should be embedded in. The core goal of the baseline evaluation is to measure and analyze the current status of the implicitly or explicitly performed KM processes, the used knowledge carrying or KM systems, as well as the knowledge culture itself. Baseline evaluations are typically applied only once to get a consistent view of the KM in the organization before larger changes. This section describes the basic process for the evaluation of a KM system as well as a summary of our baseline evaluation.

Baseline Evaluation Process

The baseline evaluation process (and similarly the delta evaluation) is structured into the three subphases: problem-determination, context-determination and knowledge-determination. As depicted in Figure 7, the steps in these subphases define a process that results in several documents (e.g., a problem description) usable for the specification of a socio-technical KM system integrated into the surrounding organizational context:

- **Problem-determination:** The problem determination serves to identify existing knowledge sources (e.g., in case of our partner, "empolis," a Wiki system named MASE) as well as emerged problems and challenges with it. Step 0a is used to determine existing information systems and technical knowledge sources using a systematic analysis method. By means of a group discussion in step 0b, the exchange of knowledge via existing technical (KM) systems in a typical project team is illuminated.
- **Context-determination:** This subphase is concerned with the determination of the context the socio-technical KM system is embedded in. The context determination produces information about the production processes, roles, documents, or sites that might be used in the KM system. The three steps in this subphase are concerned with

Figure 7. Plan for baseline and delta evaluation



the development processes that are to be supported by the KM initiative. Step 1a uses a group discussion technique to determine existing documents, templates, and other potentially reusable elements. To elicit or update the development process in use, we applied an interview with several product and project managers in step 1b.

- **Knowledge-determination:** The knowledge determination subphase targets the core information that should be made reusable via the KM system. The content, context, and structure of reusable elements is determined in step 2a using an interview. Based on this information, the knowledge transfer processes and the knowledge culture itself is analyzed in step 2b using a group discussion with several members of project teams.

Finally, step 3 is used to identify the technical infrastructure of tools and systems that are not used for knowledge management, but might be integrated and connected with the new or improved KM system (i.e., a new or improved Riki).

Results from the Baseline Evaluation

In the context of the RISE project, we applied step 0a to step 3 of the evaluation plan, as depicted in Figure 7. For the execution of the baseline evaluation one period, we required a time span of approximately 3 months, taking into account that not all employees are continuously available. Based on different vacation planning, holidays in summer, autumn, and Christmas, as well as other projects, one should take into account the time-consuming characteristics of such an evaluation.

During the group discussion concerning the usage of existing KM systems (cf. Figure 7, 0b) as well as the knowledge culture and transfer (cf. Figure 7, 2b), we determined the following status in the organization:

- The organization used a Wiki ("MoinMoin") that few people used, and contained partially documented concepts, ideas, comments, and larger documentations about planned as well as already developed software systems. Other information in the WIKI included customer information, dates, ToDo-lists, addresses, and absent lists.
- Important information about the software system itself is documented in the source code (in this case, Python and ePyDoc). To extract this information, one has to check out the whole software system from the versioning system and search for relevant information on file level.
- Important information about changes to the software system is documented in the used versioning system (in this case: Subversion). Changes at the software system are sent to all interested users by e-mail. To extract this information without the e-mails, one has to analyze every check-in into the system and appropriate comments.
- Internal information sources (i.e., repositories) with other project-relevant information are change tracking system, project folders, universal and private network drives, e-mails, and chat tools (e.g., ICQ), with information in files such as plain text or MS Word documents. Furthermore, task cards at a physical blackboard.
- External information sources are distributed over the whole Internet, but the employees had a focus on MSDN and Google if they required further information.

Positive characteristics: What goes well?

- The cooperation between two people closely working together, but who were distributed physically over two cities, worked very well using the old Wiki.
- The local teams worked on one floor and

had only short distances to their colleagues. The face-to-face communication was very good (i.e., everybody was in "shouting" distance).

Negative characteristics: What runs badly?

- The Wiki was not used anymore at the beginning of the evaluation and the feedback indicated that most people were not pleased with the structure of the knowledge base.
- Neither the old Wiki nor the documentation language for Python (ePyDoc) permitted the integration of pictures or graphics.
- The search for information in the Wiki and file system is term based (i.e., no stemming or Boolean operators). This was perceived as insufficient and demotivating by the users. Furthermore, the search within the Wiki was impeded by the use of camel case (e.g., "MyProjectDescription") in the page names.
- The discovery of relevant information is perceived as complicated as they are distributed over multiple repositories that all use different (or nonexistent) search mechanisms.
- Access to the Wiki from a text editor (e.g., emacs under Linux/Unix) or a shell (i.e., command line interface) is impossible and/or uncomfortable. Nevertheless, some developers are biased or required to use these, and are not willing to install or use other operating systems (or Web browsers such as Internet Explorer™).
- Neither were the authors (resp. other observers) informed about changes done to the content in the Wiki, nor were they informed about changes to the navigational or divisional structures (i.e., chapters and sections) of the content.
- It was not clear where to store information as they could be spread or duplicated in multiple repositories, for example, in the versioning system, change tracking system, or the code itself.

In summary, the knowledge transfer and management processes, as they were lived in the organization previously to the introduction of the Riki system, were determined by the baseline evaluation as follows:

- *Storage* of information is limited to few people in the organization and the documented information is only partially complete, consistent, or valid.
- *Reuse* of content is minimal, as the information is distributed over several sources with different search interfaces and techniques. Furthermore, the content of the documents have inconsistent structures, incomplete descriptions, or are simply outdated.
- *Workflow* for reuse of content and getting an *overview* is slow and typically demotivating, as multiple sources have to be searched manually and documents belonging together are not grouped or linked.
- *Sharing knowledge* is cumbersome, there are no templates, guidelines, or checklists to validate if the recorded information has some quality and might be easily reused by the colleagues.
- *Confidence* in the knowledge transfer system and *motivation* to share is low, as only few people are creating shareable documents and they are mostly not accurate or up-to-date. Nevertheless, the people would like to share their knowledge in a more persistent way.
- *Face-to-face communication* is strong, especially as most employees have short distances to their colleagues, are roughly of the same age, and see no need to hide their information (i.e., egghead's syndrome).

Delta Evaluation

The core goal of a delta evaluation is to measure and analyse the changes a KM system has inflicted on the affected organization. Delta evaluations can be applied multiple times during the lifecycle of a KM system (e.g., every year) to evaluate the effect on the organization.

Delta Evaluation Types

Since KM systems are usually used only irregularly in the beginning and not yet a firm part of the working process, the delta evaluation phase can be applied in three different types:

1. **Applicability:** The first type serves to examine how frequently the system is used, if it integrates into the socio-technical infrastructure, and if it helps the users during their daily work. In order to keep the effort of the evaluation down to a minimum, a lightweight evaluation is planned by reusing existing plans and other technology assessment or acceptance models.
2. **Usefulness:** In the second type, the results and experiences from the first type serve to sharpen requirements and improvement goals. The system has already demonstrated that it is applicable in the organization, and represents an integral component of the regular work routine. By means of structured evaluation processes, a formal evaluation plan is constructed to examine, in particular, the aspects usefulness, and ease-of-use for developers as well as management staff.
3. **Economy:** Finally, the third type serves to examine the economical aspects of the KM system. The system is already integrated into the regular work routine, and users share and reuse the knowledge within. An adaptive evaluation, reaction, and risk plan are developed to continuously monitor and improve the system. The focus of this type is the evaluation of the system and a forecast of the usefulness regarding the return on investment (ROI) and total cost of ownership (TCO).

To identify existing or potential problems, the entire system is continuously monitored during start-up phase. Logfiles help to drill-down into specific problems if the users observe a problem while using the system.

Results of the Delta Evaluation

In the context of the RISE project, we could only apply the first subphase and evaluate the applicability of the Riki system. Due to the short amount of time the system was used (two months) we could only get little insight into the usefulness of the system for the users in their daily routine. During the group discussion concerning the usage of the Riki (cf. Figure 7, 0b) as well as knowledge culture and transfer (cf. Figure 7, 2b), we noted the following characteristics the users liked very much about the Riki:

- Metadata elements that can be placed by every user (such as keywords in form of tags) can be very helpful in indexing the content of a Riki, and the users reacted very positively that they were able to *index every page* with their own metadata.
- The usage of metadata enabled the users to build up and use their own *individual ontology* (in form of individual tags) that is not bound to compromises or constraints from universal ontologies that might have been constructed in advance. Furthermore, metadata from the universal ontology (i.e., specified beforehand or as defined by other users) was partially used in their own ontology.
- *Searching* the information stored in the Riki is more accepted by the users when the metadata might be integrated into the search process. In the Riki, the results are *clustered*

by this metadata and the metadata might be used to *refine* the search query. The search technology exploits co-occurring metadata from multiple users and applies "collaborative filtering" techniques (i.e., "metadata x you search for is also called y").

- Annotated pages that were listed in search results reminded the users that they already read them or were highly valuable and should be read again. Similar to the Memex concept by Vannevar Bush (Bush, 1945) (cf. http://en.wikipedia.org/wiki/Vannevar_Bush), the metadata might even be used to record a reading sequence for oneself.
- The integrated view of the Riki system, including various artefacts as well as repositories, enables the user to send links that are not subject to change as, for example, mounted devices in the Windows file system.

In comparison to the status, as determined by the baseline evaluation, the usage of the Riki system had the following subjective effect on the organization:

- *More storage* of information into the Riki than before as barriers (technological and social) are reduced.
- *More reuse* of content from the Riki as users are more likely to share and search for content.
- *Faster workflow* for reuse of content and an *improved overview* due to the integrative view over multiple systems (e.g., file system, e-mail, etc.) in a central integrated repository.
- *Less barriers for sharing knowledge*, as it is easier and faster to enter information, but this typically results in a *low quality* of the content. Most users embrace the Wiki idea and record even preliminary information that is revised by oneself or others over time.
- *Higher confidence* in the system and *increased motivation* as content is easier to find and more people are participating in the sharing process.
- *Consistent face-to-face communication* even as more information is reused from the technical system.

Further and more quantitative results about the usefulness and economical aspects, especially of the more SE specific features of a Riki such as ontology-based templates for requirements, will be elicited in later evaluations.

CONCLUSION

Although the importance of usability is broadly and increasingly accepted, user-centered design processes can hardly be found in software companies in general. Small- and medium-sized companies often fail to apply user-centered design methods due to poorly established role models. Even if user-requirements are (more or less) specified, the problem that remains is to guarantee that they are strictly followed during production.

The whole research process during the project "RISE" was strongly dominated by qualitative evaluation activities. They have been preferred because they helped to specify the context of use as well as user and organizational requirements. In general, empirical data, gathered by means of qualitative research methods, is characterized by its high validity. Unfortunately, it is rather difficult to extract requirements out of this data or to check to what extent those "implicit" requirements have been considered during the development process of the software system.

Strictly following the model of a user-centered design of an interactive software system, that problem could not be bypassed. It is rather necessary to provide prototypes, as soon as possible, that contain as much functionality as necessary in order to check current implementation states against more or less "implicit" requirements.

In terms of a rather comprehensive meaning of *usability*, it was not only about developing user interfaces and improving them. In fact, usability engineering is considered as a vital concept in software development. So, it is not only about describing, designing, or producing design solutions for user interfaces. It is rather about understanding and facilitating human-engineered socio-technical systems.

ACKNOWLEDGMENT

Our work is part of the project RISE (Reuse in Software Engineering), funded by the German Ministry of education and science (BMBF) grant number 01ISC13D.

REFERENCES

Ballstaedt, S. (1997). *Wissensvermittlung.* Weinheim: Beltz, Psychologie-Verl.-Union.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Experience factory. In J. J. Marciniak (Ed.), *Encyclopedia of software engineering* (Vol. 1) (pp. 469-476). New York: John Wiley & Sons.

Basili, V., Lindvall; M., & Costa, P. (2001). *Implementing the experience factory concepts as a set of experience bases.* Retrieved May 12, 2006, from http://www.cebase.org:444/fc-md/ems_--_total_project/papers/SEKE01/seketalk18.pdf

Beck, K. (2003). *Extreme programming explained.* Boston: Addison-Wesley.

Beck, K., & Fowler, M. (1999). Bad smells in code. In G. Booch, I. Jacobson, & J. Rumbaugh (Eds.), *Refactoring: Improving the design of existing code* (1st ed.) (pp. 75-88). Addison-Wesley Object Technology Series.

Broy, M., & Rombach, D. (2002). Software engineering. Wurzeln, Stand und Perspektiven. *Informatik Spektrum, 25*(6), 438-451.

Bush, V. (1945). As we may think. *The Atlantic Online, 176*, 101-108, (Reprinted in 1996: *ACM Interactions, 3*(2), 35-46).

Cinneide, M. O., Kushmerick, N., & Veale, T. (2004). Automated support for agile software reuse. *ERCIM News, F* (pp. 22-23)

Decker, B., Ras, E., Rech, J., Klein, B., Reuschling, C., Höcht, C., & Kilian, L. (2005). A framework for agile reuse in software engineering using Wiki technology. *Wissensmanagement, 2005*, 411-414.

ISO 9241-11 (1998). *Ergonomic requirements for office work with visual display terminals (VDTs). Part 11: Guidance on usability.*

ISO 13407 (1999). *Human-centred design processes for interactive systems.*

Reinmann-Rothmeier, G. (2001). *Wissen managen* (Nr. 131). München: Inst. für Pädag. Psychologie und Empirische Pädag., Lehrstuhl Prof. Dr. Heinz Mandl.

Renzl, B (2004). Zentrale Aspekte des Wissensbegriffs – Kernelemente der Organisation von Wissen. In B. Wyssusek, M. Schwartz, & D. Ahrens (Eds.), *Wissensmanagement komplex*. Berlin: Schmidt.

Teevan, J., Alvarado, C., Ackerman, M. S., & Karger, D. R. (2004). *The perfect search engine is not enough: A study of orienteering behavior in directed search*. CHI 2004 Paper. Retrieved May 12, 2006, from http://haystack.lcs.mit.edu/papers/chi2004-perfectse.pdf

Tonkin, E. (2005). *Making the case for a Wiki*. Retrieved May 12, 2006, from http://www.ariadne.ac.uk/issue42/tonkin/intro.html

APPENDIX I: INTERNET SESSION: HCI AND WIKI

http://c2.com/cgi/wiki?WikiEngines
http://en.wikipedia.org/wiki/Wiki

WIKI systems are a way to share knowledge and collaborate in a distributed environment with other people from different cultures and with different goals. They are used in different contexts by different groups of people, and are typically developed by software programmers with a very technology-centered view and without knowledge about usage scenarios or real-use cases.

Interaction

Survey the information presented at the Web sites on WIKIs and HCI methods and theory. Prepare a brief presentation on the core concepts and history of WIKIs with a focus on their HCI. Alternatively, analyze if the current WIKI systems are built human centered or technology centered. What would you change?

APPENDIX II: USEFUL URLS

Website of the RISE project:
http://www.rise-it.info

More information about WIKI systems on Wikipedia:
http://en.wikipedia.org/wiki/Wiki and http://en.wikipedia.org/wiki/List_of_wikis

A wikibook about WIKI systems:
http://en.wikibooks.org/wiki/Wiki_Science

More information about semantic WIKIs on Wikipedia:
http://en.wikipedia.org/wiki/Semantic_Wiki

More information about the topic "Social Software" on Wikipedia:
http://en.wikipedia.org/wiki/Social_software
A collection of arguments why WIKIs work:
http://c2.com/cgi/wiki?WhyWikiWorks and **http://en.wikipedia.org/wiki/Wikipedia:Our_Replies_to_Our_Critics**

The International Symposium on WIKIs (WikiSym):
http://www.wikisym.org/

Multilingual online journal for qualitative research:
http://www.qualitative-research.net/fqs/fqs-eng.htm

APPENDIX III: FURTHER READINGS

Beyer, H., & Holtzblatt, K. (1998). *Contextual design*. San Francisco.: Morgan Kaufmann Publ.

Cooper, A. (1995). *About face*. Foster City, CA: IDG Books Worldwide.

Fensel, D. (2004). *Ontologies: A silver bullet for knowledge management and electronic commerce*. Berlin: Springer.

Lauesen, S. (2005). *User interface design*. Harlow: Pearson/Addison-Wesley.

Leuf, B., & Cunningham, W. (2006). *The Wiki way*. Boston: Addison-Wesley.

Nielsen, J. (2004). *Usability engineering.* San Diego, CA: Kaufmann.

Shneiderman, B., & Plaisant, C. (2005). *Designing the user interface*. Boston: Pearson.

Shneiderman, B. (2002). *Leonardo's laptop*. Cambridge, MA: MIT Press.

Staab, S. (2004). *Handbook on ontologies*. Berlin: Springer.

Possible Titles for Papers/Essays

- Pedagogics in Knowledge Management: Socio-Technical Methods and Tools in a Phase of Convergence
- Knowledge Management Systems and Human Computer Interaction

This work was previously published in Open Source for Knowledge and Learning Management: Strategies Beyond Tools, edited by M. Lytras and A. Naeve, pp. 122-149, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.20
Riki:
A System for Knowledge Transfer and Reuse in Software Engineering Projects

Jörg Rech
Fraunhofer Institute for Experimental Software Engineering (IESE), Germany

Eric Ras
Fraunhofer Institute for Experimental Software Engineering (IESE), Germany

Björn Decker
Fraunhofer Institute for Experimental Software Engineering (IESE), Germany

ABSTRACT

Many software organizations have a reputation for producing expensive, low-quality software systems. This results from the inherent complexity of software itself as well as the chaotic organization of developers building these systems. Therefore, we set a stage for software development based on social software for knowledge and learning management to support reuse in software engineering as well as knowledge sharing in and between projects. In the RISE (Reuse in Software Engineering) project, we worked with several German SMEs to develop a system for the reuse of software engineering products such as requirement documents. The methodology and technology developed in the RISE project makes it possible to share knowledge in the form of software artifacts, experiences, or best practices based on pedagogic approaches. This chapter gives an overview of the reuse of knowledge and so-called Learning Components in software engineering projects and raises several requirements one should keep in mind when building such systems to support knowledge transfer and reuse.

INTRODUCTION

The software industry develops complex systems that often have a reputation of being expensive and of low quality. One approach for coping with the increasing complexity of these systems is software reuse—the sharing of knowledge about software products, processes, people, and projects in an organization.

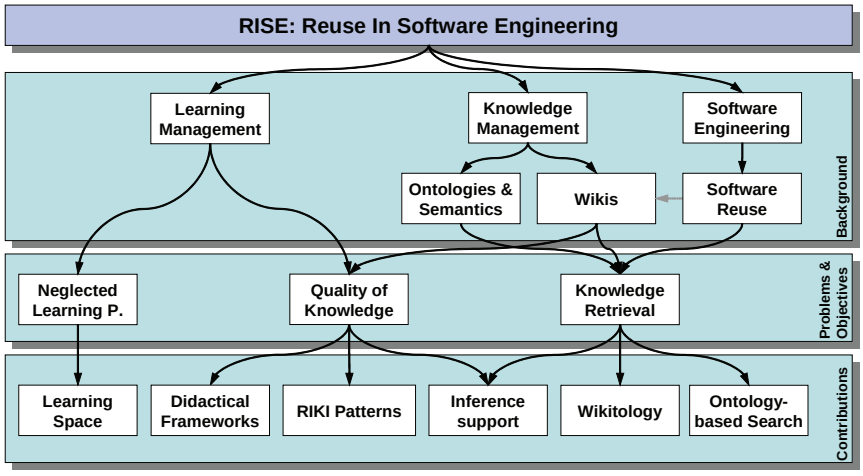
But the poor and often nonexistent documentation of this knowledge inhibits easy recording, collection, management, comprehension, and transfer. The knowledge, for example, in the form of requirement descriptions, architectural information, design decisions, or debugging experiences, needs a systematic, minimally invasive, methodological, and technological basis to strengthen its reuse and transfer in software organizations.

Knowledge management (KM) and *learning management* (LM) seem to have the potential for building this basis if they are used in synergy. However, the relationships between these two promising fields have not been fully understood and harnessed yet. On the one hand, learning as the comprehension of knowledge is considered to be a fundamental part of KM, as employees must internalize shared knowledge before they can use it to perform a specific task. So far, research within KM has addressed learning mostly as part of knowledge sharing processes, and focused on specific forms of informal learning (e.g., learning in a community of practice) or on providing access to learning resources or experts. On the other hand, LM includes techniques to preprocess and formalize knowledge, and might also benefit from other KM approaches. Especially, those approaches that support technical and organizational aspects in an organization can be used in addition to professional e-learning systems.

In this intersection between KM and LM, the Wiki technology (cf. http://en.wikipedia.org/wiki/Wiki) promises to be a lightweight basis to capture, organize, and distribute knowledge that is produced and used in organizations. Wikis are Web-based knowledge repositories where every user can freely submit or retrieve knowledge.

The *RISE* (*Reuse In Software Engineering*) project was started to support software engineers via the reuse of didactically enriched software artifacts from all software development phases in SMEs (small and medium enterprises). As the basis for knowledge transfer, we have developed a reuse-oriented Wiki (Riki) with fast and liberal access to deposit, mature, and reuse experiences made in software projects. Our Riki was extended by search technologies using case-based reasoning technology and ontologies to provide a formal and consistent framework for the description of knowledge and experiences. Ontologies and templates enrich the Riki content with semantics that

Figure 1. Outline and structure of the following chapter



enable us to didactically augment the knowledge within the Riki with additional information and documented experiences.

The RISE system sketches our approach for software reuse and tackles several problems of traditional KM and LM in learning software organizations. We use *semiautomatic indexing of pages* to improve retrieval and enable the semi-automatic creation and evolution of ontologies from Wikis (i.e., Wikitologies). The *cooperative adaptation* of knowledge to community needs and the *didactic augmentation* of content and interface are targeted to improve the usability of lightweight KM applications in agile environments.

In this chapter, we give an overview of the methodology and technology developed in the RISE project to build the reuse-oriented Wiki framework named Riki. As depicted in Figure 1, we describe in the *Background* the relevant background about knowledge and learning management, as well as software engineering and Wikis. This is followed by *Problems and Challenges* containing the targeted problems and objectives examined in the RISE project.

Our main contribution for knowledge transfer and reuse in software engineering is summarized in *Semantic-Based Reuse in SE: The RISE Approach*, which contains both the methodology in *RIME: The RISE Methodology for Knowledge Transfer and Reuse* and the technology in *Riki: The Reuse-Oriented Wiki*. Furthermore, several best practices, Wiki patterns, and lessons learned gathered during the project are listed in *RISE in Retrospective*. Finally, we finish this chapter with our conclusions.

BACKGROUND

This section is concerned with the background and related work in knowledge management with Wikis, problem-based and experiential e-learning in Wiki systems, and reuse for software development artifacts. It gives an overview of related Wikis for SE, for example TRAC, Snip Snap, and MASE.

Software Engineering and Reusable Knowledge

The discipline of *software engineering* (SE) was born in 1968 at the NATO conference in Garmisch-Partenkirchen, Germany (Naur & Randell, 1968; Simons, Parmee, & Coward, 2003). At the same conference, the methodical reuse of software components was motivated by Dough McIllroy (McIllroy, 1968) to improve the quality of large software systems by reusing small, high-quality components. The main concern of software engineering is the efficient and effective development of high-qualitative and very large software systems. The objective is to support software engineers to develop better software faster with tools and methods.

Software Reuse

The reuse of existing knowledge and experience is a fundamental practice in many sciences. Engineers often use existing components and apply established processes to construct complex systems. Without the reuse of well-proven components, methods, or tools, engineers have to rebuild and relearn these components, methods, or tools again and again.

Today, *reuse-oriented software engineering* covers the process of development and evolution of software systems by reusing existing software artifacts. The goal is to develop complex software systems in shorter periods of time or with higher quality by reusing proven, verified, and tested components from internal or external sources. Trough systematic reuse of these components and feedback about their application, their internal quality (e.g., reliability) is continuously improved. But reuse of components is only appropriate if the

cost of retrieving and adapting the component is either less costly or results in higher quality than a redeveloped component.

Since the 1980s, the systematic reuse and management of experiences, knowledge, products, and processes was refined and named *Experience Factory* (EF) (Basili, Caldiera, & Rombach, 1994). This field, also known as *Experience Management* (Jedlitschka, Althoff, Decker, Hartkopf, Nick, & Rech, 2002), or *Learning Software Organization* (LSO) (Ruhe & Bomarius, 1999), researches methods and techniques for the management, elicitation, and adaptation of reusable artifacts from SE projects. The *Component Factory* (CF) as a specialization of the EF is concerned with the reuse of software artifacts (Basili, Caldiera, & Cantone, 1992) and builds the framework in which further analysis and retrieval techniques are embedded.

In the beginning, only the reuse of source code was the focus of reuse-oriented software engineering. Today, the comprehensive reuse of all software artifacts and experiences from the software development process is increasing in popularity (Basili & Rombach, 1991). Besides source code artifacts such as requirements, design document, test cases, process models, quality models, and best practices (e.g., design patterns) are used to support the development and evolution of software systems. These artifacts are collected during development or reengineering processes and typically stored in special artifact-specific repositories.

Software Engineering Experience and Knowledge

The terms knowledge, information, and experience are defined in multiple, more or less formal, and often contradictory ways. Models that define these terms and the processes that transit from one to another differentiate between tacit and implicit knowledge (Nonaka & Takeuchi, 1995) or between data, information, knowledge, ability, capability, and competence (North, 2002). There are positions such as by Stenmark (Stenmark, 2001) that consider the usage of the term "knowledge," for information stored in a computer, inappropriate. In this model, tacit knowledge can, in fact, exist only in the heads of people, and explicit knowledge is actually information. However, the terminology used in the theory and practice of knowledge-based systems (KBS) and knowledge-discovery in databases (KDD) considers knowledge to be information stored together with its context, and we follow this convention throughout this paper.

Types of Knowledge

More specifically, we base our view of knowledge on the model of the architecture of human knowledge developed by Anderson. He classified knowledge not according to its content, but according to its state in the person's long-term memory. Two types of knowledge were defined (Anderson, 1993; Gagné, Briggs, & Wager, 1988):

- **Declarative knowledge** consists of 'knowing about'—e.g., facts, impressions, lists, objects and procedures, and 'knowing that' certain principles hold. Declarative knowledge is based on concepts that are connected by a set of relations forming a network that models the memory of a person. For instance, declarative knowledge items in the domain of software engineering might be: a definition of 'test case,' a listing of defect types, a detailed explanation of key testing principles.
- **Procedural knowledge** consists of 'knowing how' to do something, that is, skills to construct, connect and use declarative knowledge. It contains the discrete steps or actions to be taken, and the available alternatives to perform a given task. For instance, procedural knowledge items in the domain of software engineering might be: a method

for deriving test cases from requirements, a method for classifying defects choosing the right reading technique to perform an inspection.

Both declarative and procedural knowledge can be abstract or concrete. The knowledge can be connected to more or less concrete information that can be described technically, for example, by semantic networks. Nevertheless, knowledge about situations experienced or about evaluating facts or determining circumstances in given situations cannot be classified as declarative or procedural knowledge. Therefore, a third form of knowledge has extended the spectrum of knowledge in cognitive science (Enns, 1993) when Tennyson and Rasch (Tennyson & Rasch, 1988) defined contextual knowledge as another type of knowledge:

- **Contextual knowledge** consists of 'knowing when, where and why' to use or apply declarative or procedural knowledge. Contextual knowledge is created by reflecting on the usage of declarative and procedural knowledge in practice in different contexts. Contextual knowledge enables the individual to be aware of commonalities between situations, and of the appropriateness or applicability of principles or procedures in a new context.

In summary, three types of knowledge have been presented. Documented and stored experiences consist mainly of contextual knowledge. They originate, in most cases, from expert memories, and lack declarative basis background knowledge and detailed procedural knowledge, resulting in them being ineligible for learning purposes.

Furthermore, we identified the following *general artifacts* that are based on the six knowledge types from knowledge management (Mason, 2005):

- **Know-how:** Recorded information about how to do something. This can range from general *guidelines* about how to design a system to more specific and personal experiences about using a tool or software library.
- **Know-who:** Recorded information about a *person* (i.e., developer or maintainer). This can range from who designed a specific subsystem to who has knowledge about a customer.
- **Know-why:** Recorded *rationales* why something was done or decided. For example, why a design technique was applied or a defect had not been removed from the system.
- **Know-what:** Recorded information about the *status* of something (e.g., the current situation). This includes information about the current status of a project or system that was elicited from the project manager.
- **Know-where:** Recorded information about the *location* of something (esp. knowledge). This includes where an algorithm is implemented in the source code as well as information about where to find the project plan or a process description.
- **Know-when:** Recorded information about the *situation* (e.g., time or version) when something happens or a process is applied. This can range from when defects are introduced into the system to when the quality assurance process should be applied.
- **Know-if:** Recorded information about the *consequences* of an action. This can range from what happens if a defect is removed to the effects of applying a specific quality assurance technique.
- **Know-that:** Recorded information about the *facts* of something. This can range from information about the characteristics of a quality defect to a model of the development process.

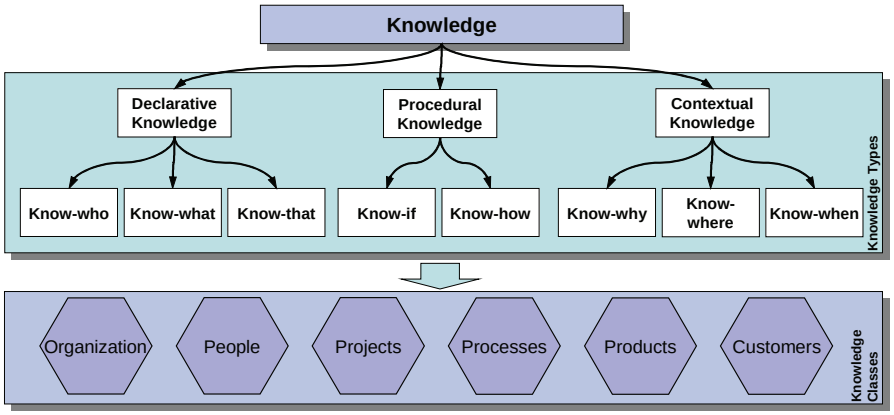
The interconnections between these knowledge types are depicted in Figure 2. The organizational elements at the bottom of the picture are classes of information that are typically existent in (software) organizations. In a KM system, knowledge, or more specifically the know-how, know-who, and so forth, about the organization, products, projects, people, processes, customers, as well as further knowledge (e.g., about a technology such as Java or EJB) is stored. We refer to this group as the OP4C model (Organization, People, Processes, Projects, Products, and Customers).

Software Engineering Artifacts

Software engineering is concerned with the planning, management, design, development, and implementation of large-scale, complex software systems. During these processes, several documents, architectures, ideas, and experiences are created or made. In software reuse, the umbrella term *artifact* is used to describe explicit knowledge objects that are considered for reuse, such as:

- **Methods and techniques** from phases such as project planning, requirements analysis, or programming that describe the know-how.
- **Patterns** about the analysis and structure of software systems (i.e., analysis and design patterns) or the management of processes (i.e., process patterns). In general, these represent knowledge that was created by the aggregation of single experiences, either by human expert judgement or by methodological approaches (Basili, Costa, Lindvall, Mendonca, Seaman, Tesoriero, et al., 2002).
- **Products** used in software projects such as project plans, product descriptions, demonstration installations, or presentation slides, as well as products produced and consumed in these projects, such as requirement documentations, architectures, software libraries, or source code.
- **Models** about the software quality (i.e., quality models) or the execution of processes (i.e., process or lifecycle models) that were used in projects and adapted to the organization and the individual needs of the project and its team.
- **Defects** found in a software system or in a project that reduce its functionality, maintainability, or other aspects like performance. Associated information of this artifact is, for example, their location,

Figure 2. Types and characteristics of knowledge



characteristics, and reactive or preventive measures. If a specific type of defect is recorded over and over again, it might be summarized in a so-called antipattern.

- **Descriptions** of the organization, people, projects, processes, products, and customers (O4PC) as well as technologies (e.g., EJB and JBoss), development rules, interface style guides, corporate quality assurance guidelines, or the corporate identity style.

Knowledge Management and Learning Management

Knowledge as the fourth factor of production is one of the most important assets for any kind of organization, and for all areas of science. Unfortunately, today few experts who have acquired valuable experience through their day-to-day work share this knowledge with other people in the organization. For example, experiences about how to solve complex problems in software development, such as installation or optimization issues, are typically not shared with colleagues.

Knowledge Management

Knowledge management is concerned with methods and technologies to enable an organization to record, inspect, adapt, and share knowledge cooperatively on a large scale. Recording, elicitation, and adaptation of knowledge are the most critical tasks in knowledge management with respect to further reuse of knowledge. The lower the quality of the knowledge is that is recorded (e.g., due to missing context information), the more complex it gets to understand and apply the knowledge in a new context. The participation of colleagues such as managers, employees, or external experts helps to record knowledge from multiple points of view.

In the domain of software engineering, software development can be considered as a human-based, knowledge-intensive activity. Together with sound methods, techniques, and tools, the success of a software project and the software quality itself strongly depends on the knowledge and experience brought to the project by its developers. This fact led to the development and use of experience-based information systems (EbIS) for capturing, storing, and transferring experience (Nick, 2005). Experience management (EM) can be seen as a subfield of KM that aims at supporting the management and transfer of relevant experiences (Bergmann, 2002; Tautz, 2001). The software system used for managing, storing, retrieving, and disseminating these experiences is called an experience-based information system (EbIS) (Jedlitschka & Nick, 2003), which is based on the experience factory concept. Another type of system that is not based on the EF concept is lessons learned systems (LLS) (Weber, Aha, & Becerra-Fernandez, 2001). Amongst the definitions for lessons learned, the most complete definition, is:

A lesson learned is knowledge or understanding gained by experience. The experience may be positive, as in a successful test or mission, or negative, as in a mishap or failure. Successes are also considered sources of lessons learned. A lesson must be significant in that it has a real or assumed impact on operations; valid in that it is factually and technically correct; and applicable in that it identifies a specific design, process, or decision that reduces or eliminates the potential for failures and mishaps, or reinforces a positive result. (Secchi, Ciaschi, & Spence, 1999)

Learning Management

KM and learning management (LM) both serve the same purpose: facilitating learning and competence development of individuals, in projects, and in organizations. However, they follow two different perspectives. KM is related to an organizational perspective, because it addresses the lack of sharing knowledge among members of the

organizations by encouraging individuals to make their knowledge explicit by creating knowledge elements, which can be stored in knowledge bases for later reuse or for participating in communities of practice. In contrast, e-learning emphasizes an individual perspective, as it focuses on the individual acquisition of new knowledge and the socio-technical means to support this internalization process. In the following two sections, expectations on learning content and related specifications and standards that address these expectations are presented.

Expectations on Today's Learning Content

Especially in industrial training settings, learning objectives mostly correspond to concrete, well-defined job-related skills, specific tasks to be done, or problems to be solved. Hence, the delivered learning material and learning approach must suit the current situation that the learner finds himself in. The situation changes over time while the learner is performing his work. Nevertheless, conventional learning systems leave no space for dynamic selection and sequencing of learning material. In addition, the expectations on e-learning content are high (cf. SCORM 2004 2nd Edition Overview page 1-22, http://www.adlnet.org/scorm/index.cfm):

- **Accessibility:** "the ability to locate and access instructional components from one remote location and deliver them to many other locations".
- **Adaptability:** "the ability to tailor instruction to individual and organizational needs".
- **Affordability:** "the ability to increase efficiency and productivity by reducing the time and costs involved in delivering instruction".
- **Durability:** "the ability to withstand technology evolution and changes without costly redesign, reconfiguration or recoding".
- **Interoperability:** "the ability to take instructional components developed in one location with one set of tools or platform and use them in another location with a different set of tools or platform".
- **Reusability:** "the flexibility to incorporate instructional components in multiple applications and contexts".

It is impossible to improve all aspects together. For example, in order to support high adaptability of learning content, more effort has to be spent on realizing adaptation mechanisms and preparing learning content so that it can be adapted to specific learning situations. This will decrease the reusability of the content and its interoperability with other learning management systems. Therefore, tradeoffs have to be made by focusing on the most important aspects for a given purpose. *Semantic-Based Reuse in SE: the RISE Approach* will elaborate on which aspects Riki will focus on. The requirement for content that fulfills these aims leads to the concept of cutting learning material into so-called learning objects with associated metadata. Since then, many standards and specifications have been developed to offer a strong fundament for learning content with the previously listed characteristics.

E-Learning Standards and Specifications

Numerous initiatives like AICC (the Aviation Industry CBT Committee), ADL (Advanced Distributed Learning), IEEE LTSC (the Learning Technology Standards Committee of the IEEE) and IMS Global Learning Consortium have made efforts to establish standards. For several years, a number of initiatives have agreed to cooperate in the field of standards. Among the many available standards and specifications, the ones most relevant for Riki are described in the following.

The LTSC has developed the Learning Object Metadata Standard (LOM). This standard will specify the syntax and semantics of learning

object metadata, defined as the attributes required to fully/adequately describe a learning object. Learning objects are defined here as "any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning." A huge amount of specifications is being developed by the IMS consortium. Several of these specifications have been incorporated and, in some cases, been adapted by ADL to define the SCORM reference model. SCORM describes that technical framework by providing a harmonized set of guidelines, specifications, and standards based on the work of several distinct e-learning specifications and standards bodies. These specifications have one aspect in common: by separating the content from the structure and layout, they enable the author to develop different variants of learning material very efficiently, while relying on the same set of learning objects. *SCORM sequencing and navigation* provides techniques to sequence learning objects by means of learning activity trees, and the IMS Learning Design specification allows expressing more sophisticated pedagogical concepts by means of a more extensive role concept. However, they prescribe structures for expressing more or less generic instructional designs, and do not provide possibilities for adapting instructional design during run-time. Nevertheless, despite their limitations, Riki will especially make use of specific concepts within SCORM (such as *sharable content object*, *organization*, *manifest*, *learning activity tree*, etc.) and parts of sequencing and navigation. Especially the contextualization of knowledge and experience (i.e., embedding the information in a context) plays a crucial role in Riki. Part of the context description could be done, for example, according to the IMS Learner Information Package specification in order to describe the content owner, or other content related roles. The Dublin Core Metadata Initiative (DCMI) has developed the *Dublin Core Metadata Element Set* outside of the e-learning domain. DC defines a very simple set of metadata attributes that can be used for describing general resources. Since the content of a Riki should be annotated as unintrusively as possible, DC will be a starting point for describing RISE content.

Wikis and Semantics in Software Engineering

The Semantic Web initiative is concerned with the enrichment of information on the Internet through the use of exchangeable and machine-readable metadata. To structure the knowledge in a "mini"-Internet, as represented by a KM system such as a Wiki, we also planned to enrich the knowledge encoded on the Wiki pages with additional metadata. In the remainder of this section, we describe Wikis in general, followed by two overviews of Wikis for SE organizations and Wikis with integrated metadata support, which form the basis of our Riki system (i.e., semantic Wikis).

Wikis facilitate communication through a basic set of features, which allows the project team to coordinate their work in a flexible way. From the authors' point of view, these basic features are: *one place publishing*, meaning that there is only one version of a document available that is regarded as the current version; *simple and safe collaboration*, which refers to versioning and locking mechanisms that most Wikis provide; *easy linking*, meaning that documents within a Wiki can be linked by their title using a simple markup; *description on demand*, which means that links can be defined to pages that have not been created yet, but might be filled with content in the future. Furthermore, the simple mechanism of URL allows easy reference and thus, traceability of Wiki content into other software documents like code. For a further discussion on Wiki features, refer to Cunningham (2005).

Besides those technical aspects, Wikis foster a mindset of a fit-for-use, evolutional approach to requirements documentation and management. The approach of Wikis—in particular, Wikipedia

as the most prominent representative (Wikipedia, 2006)—demands that an initially created document is adequate for its intended usage (fit-for-use). This initial version is then extended, based on the demand of the people using this document.

Software Engineering-Oriented Wikis

Wikis were initially used in a software engineering setting, namely the Portland Pattern Repository (Leuf & Cunningham, 2001). Furthermore, they are often used to support software development, in particular in the area of open source software. The Wikis of the Apache Foundation are a prominent example of this application scenario. Some examples of Wikis offer specific functionality for software engineering:

- **Trac** (Alrubaie, 2006) is a Wiki, written in python, that integrates an issue tracker, allowing it to relate Wiki pages to issues, and vice versa. Furthermore, the python code of a project can be integrated as read-only documents.
- **MASE** (Maurer, 2002) is an extension to the JSP Wiki that offers plugins for agile software development, in particular for iteration planning and integration of automated measurement results.
- **SnipSnap** (John, Jugel, & Schmidt, 2005) is implemented in Java and allows read-only integration of code documentation. Furthermore, it offers support for the integration of Wiki entries into the integrated development environment eclipse.
- **Subwiki** (SubWiki, 2005) is a Wiki implementation that uses the versioning system subversion as a data repository. Since subversion allows attaching metadata to files, the resulting Wiki is supposed to have the same features. However, this project has not released a stable version yet.
- **EclipseWiki** (EclipseWiki, 2005) is a plugin for the eclipse software development platform. It uses the workspace of a project as (local) data storage. By using a versioning system, the Wiki files can be shared and edited by a project team.
- **WikiDoc** (Oezbek, 2005) is a conceptual work that supports adding java code documentation via a Wiki interface. This allows nonprogrammers to participate in the creation of code documentation.

All those examples show that Wikis are increasingly being used as a platform for software development. However, "regular" Wikis (see (WikiEngines, 2005) for an overview of most of the Wikis currently available and WikiMatrix, (2006) for a configurable overview), as well as software engineering-oriented Wikis, build upon the fact that the relations between documents and further metadata are maintained by the users of those Wikis. This lack of explicit semantic information is addressed by an extension to the regular Wiki functionality that is developed in the RISE project. These so-called *semantic Wikis* are elaborated in the next section.

Semantic Wikis

In this section, we present further examples of semantic Wikis. Most of these examples are taken from the overviews presented in Dahl and Eisenbach (2005) and Semantic Wikis (2005). Those examples show that a) even "regular" Wikis offer some support for structuring their content and b) that semantic, RDF-based Wikis can be implemented. Most of those examples are general purpose Wikis that do not focus on software engineering in particular. A general overview of semantic Wikis can be found in Völkel, Schaffert, Kiesel, Oren, and Decker (2005):

- The most common way to categorize within Wikis is the usage of the backlink function of a Wiki (Aumüller, 2005). Basically, a page is created that represents a certain category.

Pages belonging to this category have a reference to this page. The backlink function lists all of these references. However, this approach has one major drawback: pages that are used to navigate to the category entry (and thus do not semantically belong to this category) are also included in the backlink list.

- Some Wikis offer additional support for structuring content. For example, *TikiWiki*, (TikiWiki, 2005) allows assigning pages to structures (table of contents) and categories (taxonomy style classification). XWiki (2005) offers forms (templates) that contain metadata that are instantiated in documents derived from this form.
- From the area of SE-specific Wikis, TRAC offers a labeling feature for pages (smart tags) that could be used for a facetted presentation of the pages annotated with those tags. SnipSnap allows determining the template of a document and offers RDF export.
- *Platypus* (Platypus, 2005) and *SHAWN* (Aumüller, 2005) allow adding RDF annotations for each page. Pages within Platypus represent a concept. While viewing a page, the RDF triples are displayed that have the current page as object (in particular, pages that reference this page) and as subject (in particular, metadata about a page). SHAWN also offers navigation support based on the ontology information added to a page. Wiki. Ont is still in a preliminary version.
- *Rhizome* (Rhizome, 2005; Souzis, 2001) and *RDF-Wiki* (Palmer, 2005) are Wikis that provide their content in RDF, thus allowing to reason about their context.

Only the Wikis mentioned in the last two bullets can be seen as "real" semantic Wikis, since they allow relating their content to an RDF-based ontology. However, all of them—at least in their current state—do not integrate their ontology into the Wiki, for example, they neither provide metadata templates to be filled in based on an ontology nor do they check whether metadata entered is consistent with an ontology. Therefore, when related to the Semantic Web layer cake (Berners-Lee, 2005), all of these semantic Wikis implement the RDF and RDFS layer. The vocabulary layer of these applications is not domain specific and thus, does not allow inferring about domain specific relations.

Ontology Development

The development of ontologies to classify and structure knowledge—often called knowledge engineering—is a rather mature field that has its origins in artificial intelligence. Several methodologies, approaches, and standards have been developed that focus on ontology development in general, or resulted in an ontology for software engineering knowledge. This work is implicitly included in the RODent method presented later, and developers of Riki-specific ontologies might find these works helpful as a source of inspiration for their own ontology development. Furthermore, by adhering to the following standards, the tools and framework based on (some of) these standards can be employed for reasoning-specific tasks in RISE:

- **Ontology development:** CommonKADS (Schreiber, Wielinga, Hoog, Akkermans, & Velde, 1994) is a methodology for developing knowledge-based systems and contains guidelines for developing ontologies.
- **Ontology description standards:** The language set of RDF (Manola & Miller, 2004), RDF-Schema (Brickley & Guha, 2004) and OWL (Smith, Welty, & McGuinness, 2004) defines language constructs that can be used to define ontology in a way suitable for machine reasoning.
- **General ontology standards:** Ontology standards make use of those ontology description standards. In RISE, we referred to

the Super Upper Merged Ontology (Legrand, Tyrväinen, & Saarikoski, 2003; SUMO Ontology), describing general relations (such as part-of) and the Dublin Core metadata to have a general set of document metadata (such as author, creation date) (2005).

- **Software-engineering-specific ontologies:** These ontologies define a general set of concepts and their relations in the field of software engineering. This allows the entries within Riki to be annotated with domain-specific meaning. Examples of such ontologies are the software engineering body of knowledge (Software Engineering Body of Knowledge, Iron Man Version, 2004) and the classification of the Association for Computing Machinery (2005).

This background section illuminated the background of our work and gave a short overview of the application field of software engineering and software reuse, as well as the Wiki technology that was used in the RISE project. These fields represent the context in which several problems were encountered that are described in the following section.

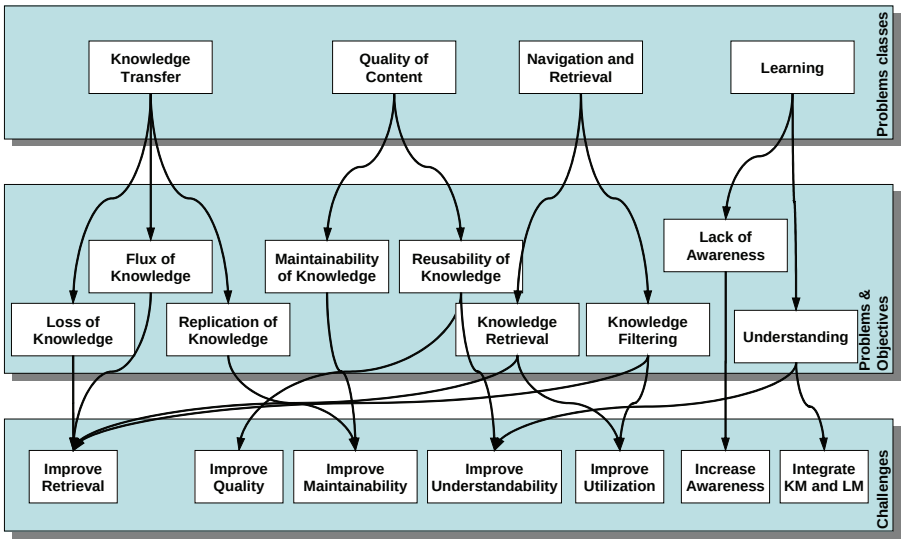
PROBLEMS AND CHALLENGES

After the description of the background of software engineering and the application of techniques from knowledge management and learning management, we look at the specific problems addressed within the RISE project. As depicted in Figure 3, we identified several problems of knowledge and learning management in software engineering.

Problems of Knowledge Transfer and Reuse in Software Engineering

As motivated in *Software Engineering and Reusable Knowledge*, the development of software is a highly knowledge-intensive activity that produces many reusable documents, ideas, and experiences. The productivity as well as the quality of the products developed heavily depends on

Figure 3. Problem overview



the efficient and effective storage and retrieval of knowledge in a socio-technical system. A knowledge management system should at least reduce or prevent the occurrence of the following basic problems in KM:

- **Loss of knowledge:** Key personnel with valuable or even critical knowledge about products, processes, or customers leave the organization and leave a knowledge gap. In a software organization, this might be a senior software developer who is the only person with critical information about the inner structure or function of a software system.
- **Replication of knowledge:** Products, intellectual property, or experiences are developed over and over again in slightly different versions. In a software organization, an algorithm might be developed twice because one project does not always know what other projects are doing. This might also apply to defects that are introduced to different parts of the software system and removed by different testers doing the same work (i.e., solving the same problem) twice.
- **Flux of knowledge:** The knowledge about new technologies, projects, processes, products, people, or standards is continuously subject to change. In order not to fall behind the innovations, competitors who are using these gaps have to be discovered and handled. For example, software organizations have to cope with many new open source projects, competitive systems, development standards, or frameworks that are developed or changed every year.

In the following sections, we have grouped specific problems into classes regarding quality aspects, the integration of KM and LM, the learning processes, as well as retrieval and reasoning about the content.

Problems Related to the Quality of Content

A knowledge base (KB) serves as an asynchronous communication medium or intermediary between people such as managers, developers, or even customers. The whole process is one of mediated communication (i.e., people are leaving messages to other people). Furthermore, there is the danger that relevant content from the outside is not considered, especially from outside the knowledge base (i.e., on the Intranet), from the Internet, and from the social system at large (i.e., the people). Until today, there are several problems related to the quality of the content in a knowledge base that still represent a barrier in knowledge management:

- **Problem of reusability of knowledge:** The quality of the reported and recorded knowledge and experience highly depends on the individual skills of the contributor, for example, the ability to structure the content, to formulate the artifacts with accuracy, and to describe it properly according to the needs of the target audience. Without assistance, this leads to quality discrepancies and gaps in the knowledge base. To be included in the knowledge base, the new content must meet minimal quality requirements, such as described in a correct, complete, consistent, concise, and nonambiguous way, including information on the context of the specific content.
- **Problem of maintainability of the knowledge base:** A regular evaluation of all content and removal of outdated entries is required. Storing several contradictory solutions for a sole problem, originating from different persons at different points in time, is a source of confusion and mistrust. The capacity to retrieve previous related experiences exists in most KB approaches, but it requires time and effort to review them. The risk of

applying outdated content is high if content does not have an attached expiry date, and on many occasions, these repositories become a sort of graveyards: some content is added, but nothing is ever thrown away. With the continually growing size of KBs, it is difficult to keep an overview in order to connect related packages and avoid inconsistencies.

While the use of Wiki systems, as applied in our project, facilitate and improve several of the previously mentioned problems due to the increased flexibility, agility, and simplicity, they also bring along their own new problems, such as the *motivation to contribute* or the *sustainability* of the Wiki system. Furthermore, the flexible and easy creation of links between Wiki pages (i.e., knowledge elements) leads to high cross-linking and hence, to the *deterioration and loss of structure*.

Problems Related to the Navigation and Retrieval

A long running KM system that is accepted by the users and continuously increases the amount of knowledge stored within it typically amasses a plethora of knowledge that is unstructured, not connected, or outdated. The retrieval of a specific knowledge component, either by navigation or search mechanisms, typically leads to an information flood. In order to enable and improve the elicitation, organization, retrieval, and usage of knowledge in software organizations, several problems have to be addressed:

- **Problem of knowledge retrieval:** Software development in general has to cope with the overwhelming amount of new processes, technologies, and tools that periodically appears. As a consequence, the fast retrieval of up-to-date information about new technologies, corporate software systems, or available experts becomes more and more important. Furthermore, there are many knowledge components that become outdated within a few years (e.g., information about Java frameworks).
- **Problem of knowledge filtering:** Searching for crucial information in a large knowledge base, especially if it is focused on a single topic such as software engineering, leads to a flood of results. The more similar the components are in a knowledge base, the more precise a search or navigational structure has to be in order to find the relevant information.

Problems Related to Learning

KM systems focus mainly on organizational learning, that is, where learning leads to collecting knowledge for the organization in order to be used by its employees or for the modifications of the software organization's processes, internal standards, objectives, or strategies. However, Rus and Lindvall stated that individual learning is considered to be a fundamental part of KM because employees must internalize (learn) shared knowledge before they can use it to perform specific tasks (2002). KM systems make the assumption that the problem of continuous competence development can be partially solved by using intelligent retrieval mechanisms and benefitting from innovative presentations of retrieval results. As a result, knowledge-based systems (KBSs) focus mainly on knowledge acquisition, storage, and retrieval, and less on the learning processes themselves, the integration with the work process, and the personal learning needs of the software developers. More specific problems related to experiential learning (i.e., learning from experience) are listed below:

- **Problem of understanding and applying of documented experience:** Experience is often documented by domain experts. A

problem that occurs when expert knowledge is used for teaching novices is that there is a quantitative difference between expert and novice knowledge bases, and also a qualitative difference, for example, the way in which knowledge is organized (Ericsson, Krampe, & Tesch-Romer, 1993). Novices lack SE background knowledge and are not able to connect the experience to their knowledge base. Hence, they often misinterpret other people's documented experience. The organization of knowledge at the experienced provider's level and at the consumer's level makes the transfer of knowledge between different levels of expertise extremely difficult. Expert knowledge is somehow "routine." This makes it challenging for experts to document experiences appropriately and to make them reusable for others. A more detailed summary of problems related to learning from documented experience can be found in Ras and Weibelzahl (2004).

- **Problem of overview and lack of awareness:** Systemic thinking is the conceptual cornerstone of the "Fifth Discipline" (Senge, 1990). It addresses the problem that we tend to focus on the parts rather than seeing the whole, and fail to see organizations as a dynamic process. Unfortunately, learning by documented experience is limited to a very focused view of SE, and does not relate the experience to the whole SE context and related domains. Another problem is that systems do not point explicitly to other available SE methods, techniques, and tools that could be useful for the work at hand, which leads to a lack of awareness.

Objectives, Challenges and Research Issues

Most of the previously mentioned problems were tackled by several researchers in projects around the world, but a solution to them is still far in the future. We tried to improve the situation by using the open Wiki technology and integrated aspects of LM and KM to support reuse in software engineering. This system, which we call reuse-oriented Wiki (Riki), has to support the comprehensive reuse of the artifacts as described in Background, and has to help the users to collect, record, retrieve, reuse, and learn from them. Furthermore, the administrative staff needs to be supported in the packaging, formalization, aggregation, and generalization of artifacts in this knowledge base.

The overall goal of the RISE project was to integrate lightweight experience management with agile software development. RISE pursued the following specific objectives:

- **Improvement of the retrieval of knowledge and orientation** in a body of knowledge to optimize the amount of knowledge and accelerate the time to access relevant knowledge.
- **Improvement of the quality of transferred knowledge** by assisting software engineers in creating optimized artifacts (i.e., with optimized content and structure) based on didactical principles and by delivering didactically augmented experiences. These artifacts should easily be adopted and internalized by users of different expertise levels to support them in their daily work and performance.
- **Improvement of utilization and usability of the KM systems** to allow the user the goal-oriented search for suitable solutions to his problem in minimal time, and to support him in the adaptation of the solution to his specific problems.
- **Integration of knowledge management and experience management** with e-learning and work itself in order to improve the reuse of documented experience and to bring learning to the work place.

- **Improving the understandability** of documented expert experience by explicitly offering additional SE learning components, and by explicitly stimulating learning activities through didactical principles.
- **Increasing the awareness of available but unknown SE topics** in order to increase the software quality based on the application of new software development approaches that have not been applied before.
- **Improve the maintenance of knowledge in KM systems**, and especially Wikis, in order to optimize the amount of potentially relevant knowledge.

SEMANTIC-BASED REUSE IN SE: THE RISE APPROACH

As we have argued, neither the technology nor the methodology currently available is sufficient to challenge the problems listed in *Problems and Challenges*. After introducing the basic concepts of Riki, we describe the methodology and technology that was developed and used in the RISE project.

Basic Concepts in RISE

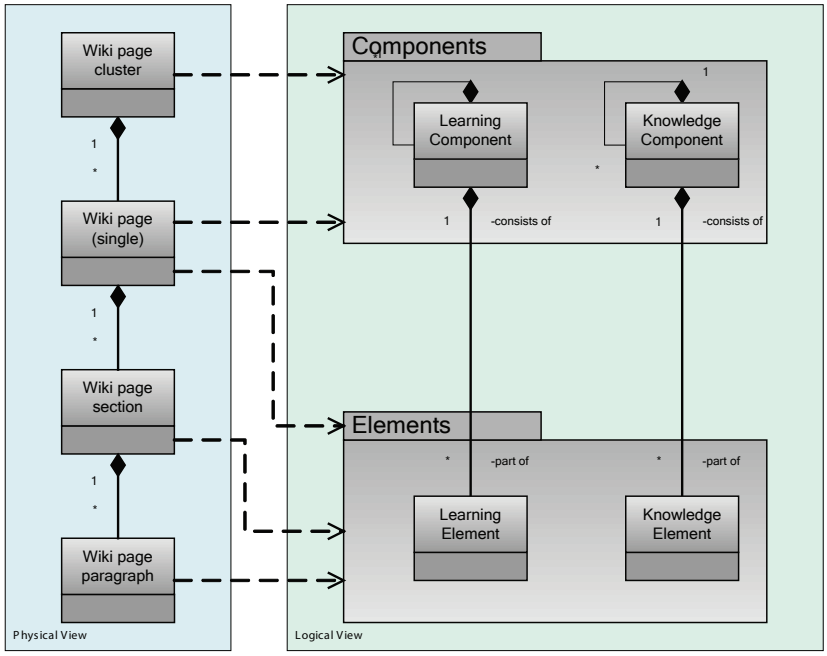
In general, we use the model defined by Tautz: "*knowledge* is the range of learned information or understanding of a human or intelligent information system, experience is considered to be knowledge or practical wisdom gained through human senses, from directly observing, encountering, or undergoing things during the participation in events or in a particular activity" (Tautz, 2001).

Representation of Knowledge in Wiki Systems

After describing Wiki systems and the types of knowledge that might be stored in those systems (see *Wikis and Semantics in Software Engineering*), we shed some light on the storage of knowledge artifacts in a Wiki system. We use the following concepts:

- **Knowledge elements:** are the most basic containers for knowledge and cannot be further divided without destroying the ability to understand them using other knowledge elements.
- **Knowledge components:** are complete and self-sufficient (i.e., independent of other knowledge elements) descriptions of knowledge (e.g., a SE artifact). A knowledge component consists of one or more knowledge elements.
- **Learning elements:** are the most basic learning resources. They are the electronic representation of media, such as images, text, sound, or any other piece of data that could serve as a learning resource when aggregated with other learning elements to form a learning component (Note: Learning elements can be compared with assets of the SCORM content aggregation model).
- **Learning components:** are units of instruction that contain at least one learning element. A learning component represents the lowest granularity of a learning resource that can be reused by the system for learning purposes. The difference between a learning component and a learning element is that a learning component is related to a learning activity and a learning objective. In addition, it can be referenced as a learning resource by the system (e.g., by using hyperlinks). Another difference is that a learning component could possess contractually specified interfaces and explicit context dependencies when these are used within a so-called learning space (Note: Learning components are similar to sharable content objects of the SCORM content aggregation model).

Figure 4. Knowledge in Wiki systems



- **Learning space:** consists of a hyperspace that contains at least one or more learning components that are presented, for example, by linked Wiki pages. A learning space follows a specific global learning goal, and is created based on context information of the current situation (e.g., learner needs, working tasks the learner is currently performing, or attributes of software artifacts). The goal of a learning space is to provide a learning environment for self-directed situated learning (see *Learning Space Analysis* for more details).

As depicted in Figure 4, knowledge elements are the typical content of a Wiki page, while a short knowledge component might equally fit. Furthermore, a knowledge element might have to be split into multiple Wiki pages if the content is too large or structured, as a training course.

Classes of Knowledge in a Riki System

The entrance of a Riki system consists of the six general classes of knowledge: projects, products, processes, people, customers, and (further) knowledge. Around these classes, several knowledge leaves are developed, based on the knowledge types from *Software Engineering Experience and Knowledge* (i.e., know-how, know-where, etc.). Therefore, a specific project page (e.g., about the RISE project) includes information about the requirements and designs of the changes planned for the products involved in the RISE project, but the product page (e.g., about the Riki) will include all requirements relevant for this specific product. However, a single knowledge leaf is not always purely classifiable as a single knowledge type (e.g., know-how), but might include other types of knowledge (e.g., source code developed in a project might include know-where, (i.e., location) as well as know-when, (i.e., version) knowledge.

Figure 5. Knowledge component classes in an SE organization based upon O4PKC

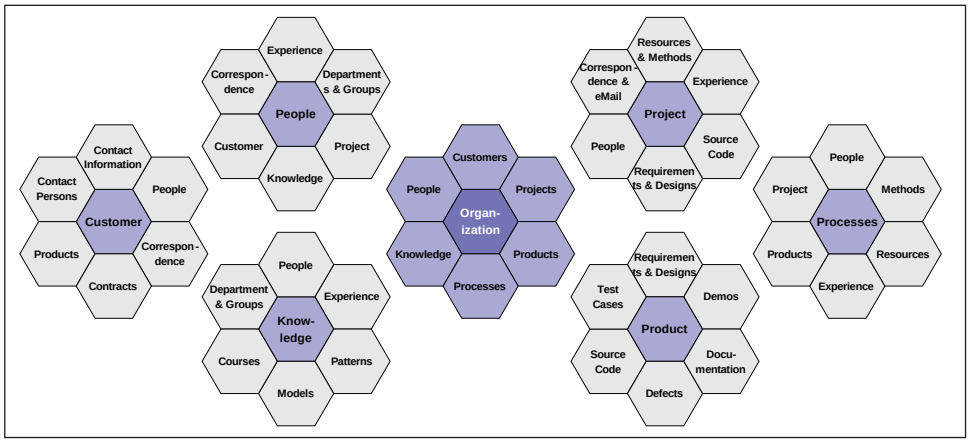
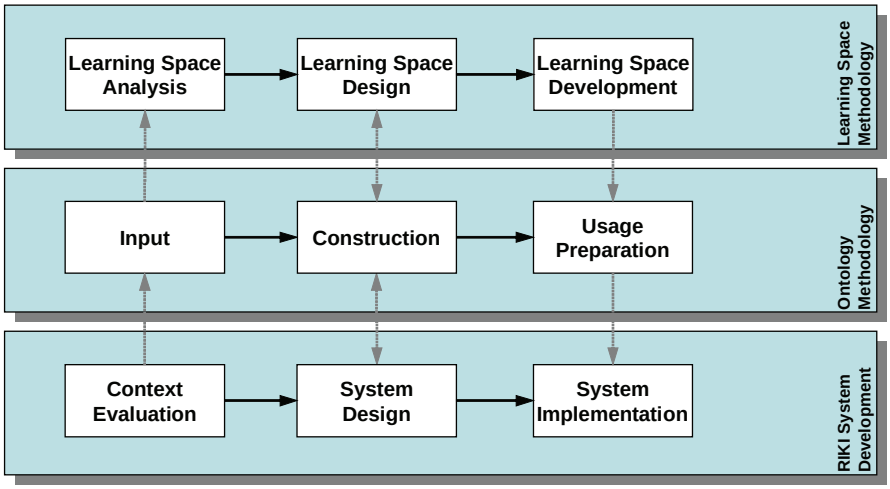


Figure 6. The RIME components (ontology, learning space, and Riki system development)



As depicted in Figure 5, the artifacts from *Types of Knowledge* might be grouped around several core artifacts, such as projects or products, and represent connection points usable for further inference. For example, in the configuration shown, one could easily infer employees who have worked in a project that was initiated by a specific customer and therefore might have valuable knowledge for further acquisition activities.

The more often these or similar artifacts are generated, the more probable it is that they might be reused in a new project.

RIME: The RISE Methodology for Knowledge Transfer and Reuse

In this section, we describe three parts of RIME, including the design and development methodol-

ogy for Riki systems, ontologies in a Riki, and learning spaces in a Riki. As depicted in Figure 6, all three are based on a detailed context evaluation in software organizations whose results were used for deriving the requirements for a Riki in the context to be addressed.

The RISE methodology represents a concept for the structuring, implementation, and operation of the Riki system.

Riki System Development

To develop a KM system such as Riki, we first analyzed and evaluated the context in two SMEs, which both develop search technologies systems. From the information we gathered in these context evaluations, we designed and developed the Riki system, considering the people, processes, and available technologies. Finally we developed, tailored, and implemented these systems in the organizations.

Context Evaluation: Analyzing the Existing Socio-Technical Infrastructure

The goal of our context evaluation step was to elicit the existing socio-technical infrastructure of the companies. Therefore, we developed a catalog of questions to elicit information about concrete topics that we wanted to know about (e.g., what technical systems are in use). In order to answer the following questions, and therefore elicit knowledge about the context the RISE system will be embedded in, we conducted an interview using a refined questionnaire based on the know questions from *Software Engineering Expereince and Knowledge*. The basic topics of the questions were:

- Where and what are technical systems with potentially valuable knowledge?
- Who are the experts for specific topics, such as products or technologies?
- How is knowledge transferred from tools to persons? What technical systems for knowledge transfer are available?
- How is knowledge transferred from person to person (interpersonal)? What social knowledge transfer techniques are used?
- How is knowledge collected from persons to tools?
- How is knowledge collected or transformed from tools to tools?
- Who needs and who produces new knowledge?
- What processes and tools are used to develop the software? Where can knowledge be collected and injected?
- When is knowledge collected, transferred, reused, needed, or produced?
- Why is knowledge currently being collected, transferred, or reused?
- What knowledge is (seemed to be) valuable for collection, transfer, or reuse? Which artifacts (as in *Types of Knowledge*) are used and should be reused?

Beside the use of questionnaires to query the employees of the companies, and elicit knowledge about the context a social-technical system has to be embedded in, we also conducted open group discussions and analyzed the existing artifacts and storage system ourselves. Overall, we used the following techniques to gather the knowledge:

- Goal-oriented, questionnaire-based *interviews* were used to put the previously listed questions to three to ten persons in two to four sessions. The collected answers were summarized and validated by the participants via e-mail.
- *Group discussions* were done at every company to collect any additional information, opinions, ideas, and so forth, that were not covered by the interviews. The discussion was started with a specific topic (e.g., why

had the old Wiki not worked for you as a KM system?) ,and every person could state what they expected from an improved knowledge management infrastructure.
- *Artifact analyses* were conducted to identify knowledge sources, the type of knowledge within, as well as how people structure their documents and knowledge in existing storage systems (e.g., the hierarchy of directories in personal file systems or in pre-existing Wiki systems).

Using the information extracted with these techniques, we derived and created tailored versions for each company of:

- **Ontology** with metadata for all pages in the KM system (i.e., the Riki) and relations between them. This metadata is used for inference, search and presentation purposes. The ontology is based upon information from the existing artifacts of the company, the knowledge lifecycle, and standards of knowledge description such as LOM.
- **Templates** for every artifact type with a special focus on requirement documents. These help to record new knowledge of that type and are used in inference, for example, to find similar documents of the same type. The templates represent the inner structure of artifacts and esp. Knowledge Elements on a Wiki page. Information to define the templates came from the identified knowledge artifacts, the knowledge lifecycle (i.e., how would a user read the document in a specific activity of the process?), and industrial best practices such as the Volere requirement template (Robertson & Robertson, 2005a) or the ReadySET requirements-engineering template.
- **Navigational structures** within the KM system (Riki) were created to support the goal-oriented and artifact-centric access to the knowledge. The structures are based upon information from the existing structures used by the employees of the company and the processes, products, and groups of the company.
- **Riki software system** with plugins to support the users with additional information from inference via the currently viewed knowledge as well as connectors to the technical infrastructure. The system design is based upon information from available software systems in the organization, established knowledge transfer techniques, and usability aspects.

System Design: Tailoring a Riki for a Learning Software Organization

After we elicited the information about the context our socio-technical system should be embedded in, we started the design of the system. Beside the information about the context, we needed further information about the technical features and the extensibility of the available systems. The following information was found to be useful for the design of the system:

- **Survey** of systems (e.g., open source), extensible without too much work, and flexible enough to realize the planned systems.
- **Selection** of an appropriate system that fulfills the optimal set of requirements by the specific customer based on the use cases and scenarios as defined by the customer's business or development processes.
- **Design of plugins and portlets** that give specific views on related articles, similar pages (of the same document type) as well as the plugin interface. In general, every know question from *Software Engineering Experience and Knowledge* represents one plugin that shows information related to the currently viewed knowledge element,

for example, the persons who have further knowledge (i.e., know-who) or methods for post-processing (know-how).

- **Design of templates and interface masks** for every specific knowledge type [e.g., templates and masks for use cases (requirements)].

Furthermore, based on the features already existent in Wikis and extensions planned for the Riki, as well as on the features of the search and inference technology, we shaped the ontology and vice versa.

System Implementation: Introducing a Riki into a Software Organization

Based upon the information from the design, the system and required plugins are built using the chosen Wiki systems. During operation, the experiences acquired from the daily work (e.g., projects) are seized and didactically augmented to support users in current or future projects.

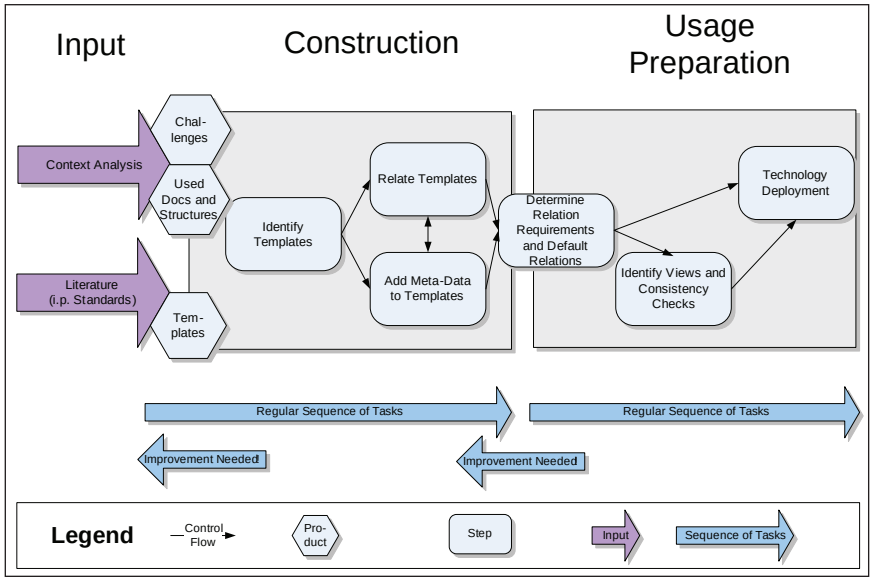
The operation and maintenance of the Riki system, its underlying ontology, and the knowledge stored within it is currently being done by the software organizations themselves.

RODent: Riki Ontology Development

This section explains the RODent method for the design and development of Riki ontologies. In parallel with systems building, we need to develop an ontology to be used by the Riki. In general, an ontology is defined as "an explicit and formal specification of a conceptualization" (Gruber, 1995).

In RISE, we instantiate this definition: An ontology is a set of templates, their metadata,and the relations among those templates. The role of the resulting ontology is to structure and relate the information captured inside the Riki. Therefore, the ontology can be seen as a link between the implementation of a Riki and the content of a Riki. The guiding idea is *Wikitology*, that is, that the Wiki is the physical representation of

Figure 7. Overview of RODent



the ontology with pages as concepts and links as relations (Decker, Ras, Rech, Klein, Reuschling, Höcht, et al., 2005; Decker, Rech, Althoff, Klotz, Leopold, & Voss, 2005).

Besides the ontology itself, the result of this method is an initial set of document templates and content to seed a Riki. In the following sections, an overview of RODent is presented, using requirements engineering as an example application.

Outline of RODent

As depicted in Figure 7, RODent is divided into two groups of subtasks: *ontology construction*, where the concepts and their relations are identified, and *usage preparation*, where the usage of the ontology is defined. A more detailed description of the following tasks is presented in the subsequent sections:

- **Input** to RODent comes from two sources: The first source is the result of the context analysis, with the prioritized challenges of the organization and the currently used structures and documents defining the "asIs" status of the organization. The second source is an analysis of domain-specific literature relevant for the domain where the ontology should be built (in the example, from the area of requirements engineering). This literature will provide an initial set of templates.
- **Construction** as the first group of tasks uses this input as follows: First, the actual templates used in Riki are identified. Second, those templates are related with each other and "equipped" with metadata.
- **Usage Preparation** as the second group is about setting up a Riki to use the ontology defined in the subsequent phase. The task "Determine Relation Requirements and Default Relation" is a link between those phases since it refines ontology relations, but also covers aspects of ontology usage in the Riki. "Identify views and consistency checks" takes care of using the ontology for navigational and editing support as well as to define inconsistencies within the knowledge captured in Riki. In "Technology Deployment," it is defined where the ontology, the views, and consistency checks are put in the Riki.

The application of RODent—provided that no increments or iterations occur—is depicted from left to right in Figure 7.

The challenges, as well as the feedback when developing and using the systems, guide the execution of the method:

- In the first execution of the method, one has to concentrate on solving the most relevant challenges most of the people using the Riki are interested in. In particular, one should try to identify multipliers, that is, people in the organization who have a special stake in the Riki usage and support their daily work. For example, if inconsistent and outdated documentation is a challenge, then the resulting ontology should support the documentation author.
- During development and usage of the systems, increments and iterations are likely to occur. Since those terms are sometimes used with interchangeable meaning, we define their meaning within RODent and their impact on ontology development: Increments are additions to the current ontology without the need to change further parts. Iterations also include a change within the existing ontology.

Within the RISE project, we encountered several increments (e.g., adding an architectural description) and some iterations (e.g., rearranging use cases), but no severe change of the ontology occurred. This experience supports the demand-oriented approach that is the basis of RODent.

However, this stability of the resulting ontology might be a result of the comprehensive context analysis and literature survey.

The following sections describe the task of RODent in detail and provide an example of the usage of this method for requirements engineering.

Identify Templates

The main objective of this task is to identify the templates capturing knowledge inside a Riki, and to create prototypes of the templates needed. As outlined in the method's overview, this task has the following two inputs:

- The result of the context analysis, with its overview of the currently used documents and the challenges of the organization.
- The literature survey identifying standards and other useful information to be used in template development.

During identification and development of those templates, we found the following guidelines to be helpful:

- **Split and link self-contained documents:** Resources found in the literature (e,g., standards) are normally part of a self-contained document written in a linear way. Since Riki is a hypertext-system, one needs split and link coherent information chunks found in those documents. Potential candidates are the different sections within these documents. For example, in the Volere Requirement Specification Template (Robertson & Robertson, 2005b), we identified the different user descriptions based on this approach.
- **Refactor documents covering multiple aspects:** The resulting templates should cover one aspect of information. Given a readable font, this guideline boils down to the following operational rules: (1) When a template is completed, it should not be shorter than one and not longer than five screens. (2) Each (main) section of the template should fit on one screen. Adherence to this guideline is the basis for the reuse of knowledge, since it allows referring to specific information.

When all templates concerning the current challenges are addressed in the first execution of RODent, further templates demanding too much additional effort should be left out. To find out about those templates, one should ask the user whether the information is relevant and whether the template would be completed.

As a preparation to the following steps, "Relate Templates" and "Add Metadata to Templates," the templates should be arranged into main categories according to their purpose. In RISE, we found the following categories helpful:

- **Context templates** like project and homepage of employees (people) based on O4PKC.
- **Navigation templates** like overviews about certain types of documents for specific reader groups.
- **Core templates** that define the structure of SE artifacts (cf. 0) directly related to software engineering activities (in the example, requirements engineering documents).

Finally, in order to test the templates, one should try to map some of the available information in the organization to the developed templates. This test might indicate hints that some of the information required by the templates is not available within the organization.

Add Metadata to Templates

After the templates and their contents are identified, they are annotated with metadata to support filtering and finding of relevant documents. To test

which set of metadata is suitable, one should test it with a template. To identify relevant metadata, the following sources should be considered:

- **General standards:** Domain-independent, standardized metadata-sets like Dublin Core and Learning Object Standards (e.g., LOM) provide general-purpose metadata. This allows other applications that rely on these standards to reuse the content of a Riki.
- **Template-specific metadata:** Some templates derived from existing standards and documents might already contain a set of specific metadata to classify their content. Furthermore, similar templates that are on different sections of the outline of the document are candidates for identifing metadata. One example is the description of stakeholders and users in the Volere Requirement Template (Robertson & Robertson, 2005b), which has a similar structure.
- **Method-specific metadata:** Some metadata might not be directly related to a template, but to the underlying method. Using this metadata allows providing method-specific support in the form of know-how documents. An example from RISE is the TORE method (Paech & Kohler, 2003), which classifies requirements engineering documents (e.g., as description of the current state "As Is" and the system to be developed "to Be").

For adding metadata in general, remember the following guidelines: *Automatically derivable metadata* (like the author's name) can be added without further consideration. *Manually entered metadata* should only be considered if it is of direct benefit to the person who will enter it. In our experience, it is no sufficient benefit that the author is able to find this document based on the metadata entered: This search will be performed in later stages of the project. Therefore, improving the searchability of the document does not provide any direct benefit. An example of metadata with direct use is the use case classification, which can be used to generate overviews. Whether this direct use is actually taking place should be checked in the task "Identify view and consistency checks."

Relate Templates

The tasks before focused on identifying and creating single documents that have a high internal coherence. In this task, these templates are linked in order to define their semantical relations, which are later used by a Riki to identify relevant documents.

In the work, several general types of relations are identified that form the foundation for defining the relations within RISE:

- **Inheritance/Subclass:** This relation denotes that one subconcept (template) inherits the properties (in RODent, the metadata) of a super-concept (a generalized template).
- **Instantiation:** This relation denotes that an object (in Riki: a document) belongs to a certain class (in Riki: a template) and thus, inherits the properties of this class.
- **Part-of/composition:** This relation denotes that an object derived from a concept is supposed to be part of another object.
- **Temporal:** This relation denotes that there is a relation within time between one concept to another concept. For example, it is a temporal relation that one object is created before another one.

Within RODent, those general relations are refined further according to their interrelations between templates and their documents. The actual use of these relations for reasoning is described in the task "identify views and relations":

- **Hierarchy** among documents of the same templates (part-of type): Instances of templates might have a hierarchical relation. For

example, one use case might have several subuse cases. It might be necessary to distinguish between different types of hierarchies for documents derived from one template: However, in RISE, we found one hierarchy to be sufficient.

- **Used in** (part-of type): This relation describes a "strong" relation between documents of different templates: If the content of a document is needed to understand another document. For example, the description of an Actor is part of the description of a use case. Without knowing the description of the Actor, the use case is not sufficiently understandable.
- **Refined in** (temporal): Refined in defines a temporal relation between documents. For example, a user story (a prose text of a certain requirement) is refined in a use case (a more structured representation).
- **Of Interest** (part-of type): This relation is used for weak relations between documents of different templates, that is, where information relevant to one document can be found in another one.
- **Context definition** (inheritance type, relation between context documents and other documents): This relation is used when additional context of a document is defined in another document. For example, the project context (like size, programming language) of a use case is defined in the project homepage. Simplified, when such a relation is established, the referencing document "inherits" the metadata of the document (in this case, the metadata about the project).
- **IsA** (inheritance type, between templates and documents): The IsA relation is reserved for denoting the relation between a template and an instance of this template.

Determine Relation Requirements and Default Relations

In the previous task, (relevant) potential relations between documents have been identified. In this task, those relations are annotated to define requirements concerning the relation itself. These relation requirements are used in the subsequent task to derive views and consistency checks. In addition to these additional relation requirements, a default relation between documents derived from a certain template is defined.

The additional requirements define the nature of a relation between templates and thus, the nature of the link between documents derived from those templates that have this relation. In other words, they define the nature of a link between the source document (i.e., where the link is defined) and the target document (where the links points to). In RISE, the following additional requirements concerning relations were identified:

- "*Required*," "*recommended*," "*optional*," and "*not allowed*" denotes the degree to which a link is supposed to be established between documents.
- "*Unique*" and "*multiple*" denotes the cardinality of the relation, that is, the source document can have only one relation to a target document, or it may have multiple relations.

Based on this overview of relations between these templates, the default relation —that is, the one used most often—is determined. For example, the default relation between use cases and actors is Used In. As long as no other relation is stated, a Riki assumes that the link between those documents is the default relation. Therefore, a Riki must state less explicit information about

Table 1. Ontology elements

Ontology element / structure	Views	Consistency Check
Metadata	Present metadata annotations	n/a
Inheritance	n/a	Is there a cyclic relation among subclasses?
Instantiation (IsA)	Show all documents belonging to a template	Does a template have documents? Does each document belong to a template?
Part-of	Show documents that are the target of a part-of relation	Are there cyclic relations among documents derived from templates that have a part-of relation (i.e., is the "higher" document part-of of a "lower" document")?
Temporal relationship	Show timeline of creation of documents	Does the document derived from the source template have an earlier editing date than the target document?
Quality of relation (i.e., required and not allowed)	Show documents derived from templates that are the target of the required relation.	Are all required or not allowed relations satisfied?
Cardinality of relations	n/a	Are cardinality constraints violated?

the type of the link. In addition, a user not aware of the additional Riki features can use a Riki like any other Wiki system.

Identify Views and Consistency Checks

In the previous steps, the templates, their metadata, and relations were defined. In the task "Identify views and consistency checks," the actual use of these results is determined. In particular, the definition of views and consistency checks is based on the general types of relations and the additional requirements presented in the previous two tasks. For metadata, relation type and additional potential requirement views and consistency checks are presented in Table 1.

Technology Deployment

The task "technology deployment" covers how the templates, metadata, and their relations are implemented within Riki. The task itself is subdivided into two sequentially performed subtasks: 1) Deployment of templates, metadata, and ontology upon distinct Wikis (if any) and 2) the deployment of ontology concepts on native Wiki syntax, that is, how parts of the metadata and relations are expressed by using naming conventions.

The *deployment on different Wikis* is trivial if only one Wiki is used. However, based on the experience from RISE, it is sensible to use, at least in larger organizations, one Wiki for each group (in particular, projects) and one main Wiki for the whole organization. This separation between different Wikis has several advantages independent of whether or not Riki is used: First, it reduces the complexity of access right administration by dividing it into several sub-Wikis. Second, the possibility of name collision is lowered. An example of such a naming collision would be if two projects using the same Wiki have a meeting on the same day. If both want to use "meeting<date>" as the name for the document containing the meeting minutes, a name collision occurs. Furthermore,

if there is no syntactical collision (i.e., the same name), a semantic collision might happen: Members of one project might link to the entry of the other project because they cannot differentiate between the two documents (e.g., "meeting<date>" and "<date>meeting"). Second, it is more likely that the group responsible for a Wiki accepts it as "their" Wiki. The separation into several Wikis also has an advantage for Riki: Since each group owns "their" Wiki, contextual information about this group can be derived. An example is project Wikis, where context information about the project could be derived from the metadata defined on the project homepage.

The *deployment on native Wiki syntax* is partially based on the deployment on several Wikis. By using the interwiki feature, the context of a document can be derived even if referred from a different Wiki. Another example for deployment of parts of the ontology into native Wiki concepts is the instantiation (IsA) relation. Within Riki, we had the following naming convention to denote this relation without the need to explicitly create a link: The instances of a template have the name of the template as prefix. For instance, documents of the type "Actor" are named "Actor_<NameOfActor>."

When an ontology developed using RODent is finished, it should contain the relevant information to execute the software engineering activities it is intended for. How it is actually used and gradually enriched with experience and further information on how to perform software engineering tasks is covered by the learning spaces described in the following section. These are an unintrusive approach for providing the procedural knowledge needed to perform a certain task.

Riki Learning Space Development

In many publications about learning objects, terminological issues are discussed, because there is a lack of solid theoretical foundation in the field of e-learning (Self, 1992), especially concerning learning objects. It seems that the main reason why learning objects have been invented is reuse and the desire for more flexible, adaptive learning systems. Current development efforts with learning objects are mainly concerned with metadata and content packaging aspects. Current object metadata says little about how to combine learning objects with others, and this will limit the success of the numerous repositories of learning objects that are being developed. Nevertheless, a model such as the SCORM *Sequencing and Navigation* is a first step towards a more systematic way for defining the sequencing of learning content and navigating through it.

Sometimes, learning objects are put into relation to object-oriented programming objects. The term "object" is somehow misleading. Some authors toss around theoretical connections to object-oriented theory that stem from computer science. One reason why many of us attempt to connect learning objects to code objects is that there is a grammatical affinity between "object" as used in "learning object" and object-oriented programming theory. It is not wrong to refer to concepts from object-oriented theory in order to increase our understanding of learning objects and our belief in successful reuse. Friesen states that there is not only a conceptual confusion in the literature between software objects and learning objects, it also seems that object-orientated programming objects and learning objects do not fit together at all (Friesen, 2001).

Therefore, we will use the term *learning component* instead of learning object. They should be considered as components instead of objects because their characteristics are similar to software components. A software component is a "unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties" (Szyperski & Pfister, 1997). Component-based software development is based on the principles of "separation of concerns" in order to

Figure 8. Development Dimension of Learning Space Engineering

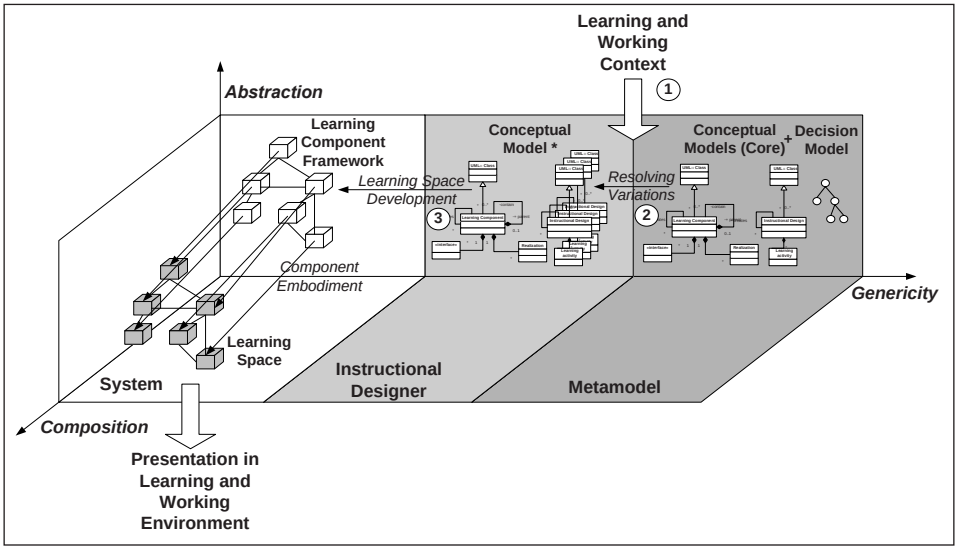
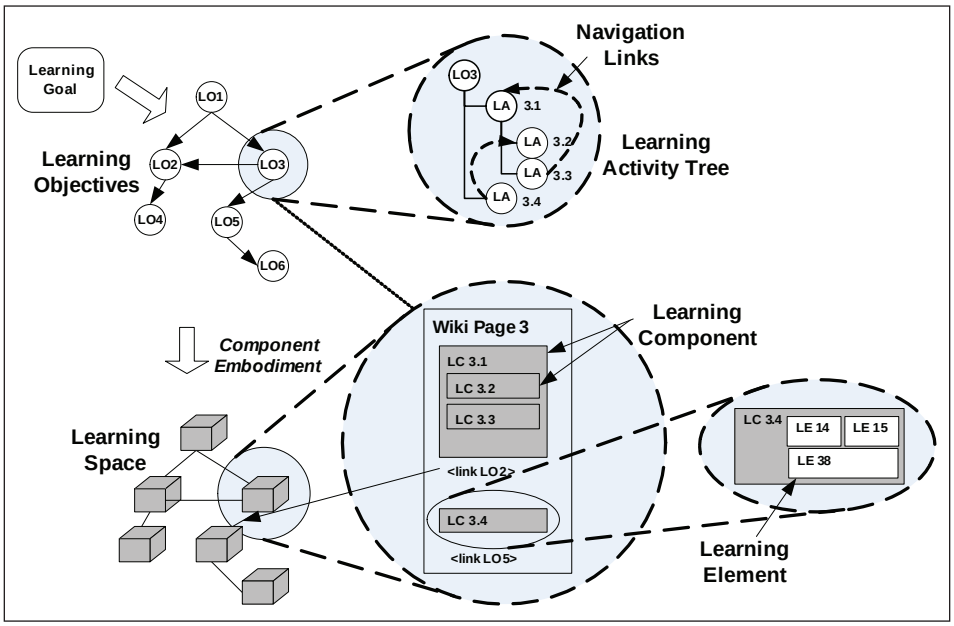


Figure 9. Details of an embodied learning space



decrease the complexity of system design and its latter embodiment with components. Therefore, a set of formal description models for components, their interfaces, and context dependencies are defined in a metamodel in order to develop components from an abstract, generic, and coarse-grained view of a conceptual component model into concrete, specific, and fine-grained learning material. This material is composed of learning components that can be integrated into a learning space (see Figure 9).

In order to develop learning spaces (see *Representation of Knowledge in Wiki Systems* for the definition of concepts used in these sections), the following three phases can be identified (see numbering in Figure 8):

- During the *learning space analysis* phase, the results of the context evaluation are analyzed. The main goals are to find out what types of content are available and useful for developing appropriate learning spaces, how learning needs can be identified based on analyzing the produced software artifacts, and in which way learning spaces can be connected to the working environment.
- During the *learning space design* phase, the analysis results are used to describe the learning space from a conceptual point of view. This means that appropriate learning methods, learning goals, and a content classification have to be chosen. Based on this information, the metamodels of the learning components and the instructional design are adapted to the current learning context.
- During the *learning space development* phase, the physical learning space is created to be explored by the learner by transforming, in a first step, the metamodels into a framework with "empty" placeholders for learning components and learning elements (see *Representation of Knowledge in Wiki Systems* for the definition of these concepts). The automatic embodiment replaces the placeholders within the framework with concrete learning components and elements, that is, Wiki pages with appropriate links and navigation structures are created during run-time.

The subsequent sections explain the three phases, which have to be performed for setting up a learning space, in more detail. Each phase is described with a few main questions that cover the goal of that phase and end with a concrete example.

Learning Space Analysis

This phase can be motivated around the following four questions:

- What is the most relevant domain for improving task performance and competence development, and which target group do we address?
- What type of knowledge that is suitable for learning is available or created by the software engineers during work?
- Where are the triggers for learning space creation and how can a learning space be related to the working context?
- Which software artifacts could be used to identify (e.g., automatically) competence and skill gaps to be solved by a learning space/learning arrangement?

In general, learning spaces can have two different purposes. First, a learning space can improve short-term task performance, that is, by providing solutions in order to solve problems more efficiently, or by offering different methods or tools that enhance a specific well-known task. The domain under consideration is very narrow. Second, long-term competence development involves an analysis of a much broader domain software engineers are working in. Here, we would like to find our competence gaps that cover perhaps more than one software development phase, such as general software engineering principles or completely new technologies or development approaches.

Usually, a software project covers the whole development cycle from requirements elicitation to programming, testing, and delivery of the software to the customer. The aim of the Riki methodology is not only to address the whole development

Table 2. Overview of educational goals and associated learning element types

Knowledge Type	**Knowledge Dimension**	The Cognitive Process Dimension (Learning Objectives)						Learning Element Types
		1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create	•Definition •Description •Example •Counterexp. •Analogy •History •Overview •Summary •Scenario •Procedure •Explanation •Theorem •Rule •Law •Principle •Model •Practice item •Checklist •Strategy •Reference •Learning Space Map • …
Know-What Know-who	A. Factual Knowledge	LA 1						
Know-That	B. Conceptual Knowledge		LA2	LA4				
Know-how	C. Procedural Knowledge		LA3	Overall Learning Goal	LA5			
Know-if **Know-when** **Know-where** **Know-why**	D. Meta-Cognitive Knowledge			LA6				

process, but to focus on very specific tasks first and to extend the scope of the Riki later. This has the advantage that early successes on important and difficult tasks, in terms of better task performance and competence development, can be used to extend the system's scope. This step is called *domain scoping*, and will also influence the development of the domain ontology. We found out that, especially during the requirements and programming phase, suitable tasks can be found to get the system started. The information that is necessary to choose such tasks was gathered through personal interviews, by analyzing content available in knowledge repositories, and by looking at artifacts produced, such as code or software documentation. In addition, a rough process model and a role model are derived in order to relate the learning space to the well-known processes and roles. It is essential to ask software engineers what kind of problems bother them the most, and which tools, methods, and techniques they apply to create software artifacts.

During the context evaluation, available content was already analyzed and software engineering challenges were identified. In this phase, we check especially if this knowledge could be used for learning components and learning elements. The content is classified according to the knowledge dimensions and types (see Table 2 for details). Furthermore, metadata and granularity (i.e., size) are investigated in order to make first decisions about metadata extensions, respectively adaptations, to the metamodel of learning content. The idea is not to change existing policies and rules for content authoring. Learning spaces, and hence the learning components, should be adapted to the existing situation and content within the organization.

Another important issue is to find out which situations trigger a learning need. The generation of a leaning space is demand-driven, that is, learning spaces are created based on a trigger. We distinguish between two types of triggers. The human-based trigger is created by the software engineer himself, for example, by searching for specific information, documented experience, or solutions. These search results could be embedded in a learning space. The system-based trigger is

related to an internal system event, for example, the technical software development environment analyzes a code artifact and finds a defect, which leads to a learning space where the engineer could learn how to remove this defect. Human triggers can be found by interviewing the engineers. System triggers have to be defined by Riki experts. Those triggers strongly depend on the capability of analyzing produced software artifacts such as code or other formally described documents. Based on those findings, appropriate learning spaces could be created in order to improve the quality of those artifacts or their development process.

The last aspect in this phase is to relate learning spaces directly to the working context. Since the learning space should be delivered as close to the work place as possible, connection points have to be found. In the domain where Riki can be implemented, those points are usually located within the technical development environment (e.g., Eclipse IDE) or within the knowledge management environment.

Learning Space Design

After finding out what domains are the most promising for learning (i.e., both long-term as well as short-term learning), and what kind of artifacts could be used to identify skill and competence gaps, this phase defines learning goals and appropriate methods to be applied for the learning space development in the last phase. This phase can be motivated around the following four questions:

- What kind of learning goals could be defined and how are they related to the knowledge dimensions/types and learning content?
- What are appropriate learning methods for the identified learning needs, triggers, and competence levels?
- How is the domain ontology used or adapted to meet the domain context and requirements for creating learning spaces?
- How is the instructional designer supported in adapting the core metamodels to the requirements for creating learning spaces?

Our learning spaces follow the constructivist learning theory and are related to the current working situation as closely as possible. Before we provide answers to these questions, we would like to say some words about learning theories and learning approaches that are suitable for the software engineering context.

First, following the constructivist learning theory, learning can be seen as a self-directed process where the focus lies on opening up information and on constructing individual knowledge, for example, (Bruner, 1973). Constructive learning theories criticize former learning methods for ignoring almost completely the transfer of lessons learned to new situations in practice, that is, knowledge remained in a latent condition and was not applicable.

Second, situated learning approaches developed mainly at the end of the 1980s emphasize that a human's tasks always depend on the situation they are performed in, that is, they are influenced by the characteristics and relationships of the context (Brown, Collins, & Duguid, 1989). Because of the relation between cognition and context, knowledge and the cognitive activities meant to create, adapt, and restructure the knowledge cannot be seen as isolated psychological products: they all depend on the situation in which they take place. Learning involves interpreting individual situations in the world based on one's own subjective experience structures. Learners have an active role and derive most of the knowledge from real situations by themselves, and this knowledge is afterwards integrated into their own knowledge structures. Learning and applying the lessons taught should happen in a situated context, that is, during the development of software artifacts.

Third, research in cognitive psychology has shown that students learn better when in-

volved in solving problems. Collins' Cognitive Apprenticeship (Collins, Brown, & Newman, 1989), Schank's Goal Based Scenarios (Schank, Bermann, & Macperson, 1999), and the 4 Component Instructional Design (4C/ID) Model of Merriënboer (Merriënboer, 1997) are just three of the instructional models that address problem-based learning. Merrill proposed the *First Principles of Instruction*: Learning is facilitated when previous experience is *activated*, when the lessons learned are *demonstrated* to the learners instead of just being presented to them, when the learners are required to *apply* their knowledge or skill to solve a problem, and when the learners are motivated to *integrate* the new knowledge or skill into their daily work (Merrill, 2000).

In fact, instructional design could be handled in two obvious places: embedded within a learning component or as a separate object (e.g., by using specifications such as SCORM Sequencing and Navigation or IMS Learning Design). Riki handles the pedagogical rules of instructional design *outside* of the learning components in a metamodel for instructional design. By applying those didactical rules to this metadata, adequate learning spaces can be created that fit the current situation of the user.

Based on the identified triggers, appropriate learning methods are chosen. In the domain of software engineering, problem-based learning methods, as listed previously, and experiential learning (i.e., compared to experience-based learning, experiential learning integrates elements of reflection, support, and transfer) methods are a good starting set for creating learning spaces. The utilization of a knowledge management system is usually problem-driven, that is, a problem arising during the completion of a software engineering task motivates the software developer to search for suitable information or even complete solutions in the repository. When reusing an experience, a developer is usually engaged in active problem solving while reading, understanding, abstracting, or instantiating the experience, and trying to apply the gathered knowledge to the real problem situation. Ideally, software engineers could learn effectively from experiences when all four phases of Kolb's Experiential Learning Circle (Kolb, 1984) are passed: making a concrete experience, observing and reflecting about the occurrence, forming abstract concepts, and testing these concepts in new situations. When a software engineer documents an experience for later reuse (i.e., this is usually done by creating abstractions), he or she profits from being involved in the situation that leads to the experience, and his or her own observation and reflection about the happening. When a software engineer other than the experience's provider wants to reuse this documented experience, he or she will lack specific knowledge about the event that led to the experience, and the knowledge that results from observation and reflection. Hence, the learning space should focus on the delivery of appropriate content, in addition to the experience, in order to support knowledge construction as described in Kolb's learning cycle.

In addition to the triggers, the role and the competence level of the software developer plays a crucial role in how learning methods are implemented in a learning space. We follow a self-directed learning approach with a specific amount of guidance based on the capabilities of the learners, that is, learners proceed through the learning space at their own pace, and decide by themselves which learning components they want to access in which order. Nevertheless, when learning spaces are created, a certain amount of guidance and suitable content is provided to learners, depending on their competence level. Riki distinguishes between three competence levels: novice (knowledge dimension: declarative knowledge), practitioner (declarative and procedural knowledge), and expert (all kinds of knowledge).

Content for learning has been identified in the first phase, and domain-related basic software-engineering content that is not available has to

be added to the repository. This content is now classified according to learning element types. A basic set, as listed in Table 2, is used to categorize the learning content. The set is a mixture of different instructional designs as well as software engineering specific types. This set could be extended if necessary. Learning elements could not be related directly to the cognitive processes categories, respectively learning objectives, because many learning elements could be used in different cognitive processes such as "Example."

We refer to Bloom's taxonomy of educational goals (Bloom, Engelhart, Furst, Hill, & Krathwohl, 1956), which is widely accepted and applied in various topic areas including software engineering (Dupuis, Bourque, & Abran, 2003). In addition, we refer especially to the revision of the original taxonomy by Anderson and Krathwohl (2001), which is briefly explained next. The new taxonomy can be explained by two dimensions: the knowledge and the cognitive processes dimension. Our work is based on a similar knowledge dimension as the one in this taxonomy. Regarding the cognitive process dimension, Anderson and Krathwohl distinguish between six different categories:

- **Remembering** is to promote the retention of the presented material, that is, the learner is able to retrieve relevant knowledge from long-term memory. The associated cognitive processes are *recognizing* and *recalling.*
- **Understanding** is the first level to promote transfer, that is, the learner is able to construct meaning from instructional messages. He/she builds a connection between the "new" knowledge to be gained and prior knowledge. Conceptual knowledge provides the basis for understanding. The associated cognitive processes are *interpreting, exemplifying, classifying, summarizing, inferring, comparing,* and *explaining.*
- **Applying** also promotes transfer and means carrying out or using a procedure in a given situation to perform exercises or solve problems. An exercise can be done by using a well-known procedure that the learner has developed a fairly routinized approach to. A problem is a task for which the learner must locate a procedure to solve the problem. Applying is closely related to procedural knowledge. The associated cognitive processes are *executing* and *implementing.*
- **Analyzing** also promotes transfer, and means breaking material into its constituent parts and determining how the parts are related to one another, as well as to an overall structure or purpose. Analyzing could be considered as an extension of Understanding and a prelude to Evaluating and Creating. The associated cognitive processes are *differentiating, organizing,* and *attributing.*
- **Evaluating** also promotes transfer and means making judgments based on criteria and/or standards. The criteria used are mostly quality, effectiveness, efficiency, and consistency. The associated cognitive processes are *checking* and *critiquing.*
- **Creating** also promotes transfer and is putting elements together to form a coherent whole or to make a product. Learners are involved in making a new product by mentally reorganizing some elements or parts into a pattern or structure not clearly presented before. The associated cognitive processes are *generating, planning,* and *producing.*

Riki addresses all the categories of these dimensions, with the focus being on the first three categories, because these are important for reaching the upper levels and can be taught directly, while the fourth to sixth levels require a longer term and deeper understanding of a subject matter.

Common instructional design theories often speak of the following elements in the design of instruction: generalities, examples, explanations,

practice items, test items, overviews, advance organizers, and analogies, among others (Yacci, 1999). Table 2 shows the educational goals and the related learning component types that were selected as a first set for a Riki in the domain of software engineering.

Although a lot of effort has been put into the definition of standards, and although the LOM standard seems to be widely accepted by now, "key issues related to global content classification such as a common schema, ontology, granularity, taxonomy and semantics of learning objects which are critical to the design and implementation of learning objects remain unsolved" (Mohan & Daniel, 2004). Hence, one of the key issues of this phase is to adapt the domain ontology in order to describe the semantics of learning components/elements and their relations, and to find a common vocabulary for describing learning components/elements.

The resulting ontology created by RODent covers the domain that was identified in the analysis phase. Learning spaces tie into this ontology. After defining the types of learning components, each learning component/element has to be related to the concepts of this ontology. This means that the ontology concepts are used for describing the learning components/elements with metadata. Riki will use LOM as metadata description. The suggested LOM vocabularies are adapted to the types of Table 2 (right column) in order to specify the value range of the LOM attributes.

The most complex task in this phase is the adaptation of both metamodels to the working and learning context where the Riki should be implemented. There exists one learning component core metamodel for defining the learning component and its composition of learning elements on a conceptual level, and one core metamodel for instructional design applied to this domain. The adapted metamodels are used to produce a learning space framework (i.e., a concrete implementation framework to create learning spaces). The types and relations between learning components (i.e., according to the relations defined by RODent) are explicitly modeled in the metamodels. The conceptual model covers aspects, such as the specification of metadata (i.e., by using LOM), containment rules that specify the parent-child relationships between learning components/elements within a containment tree (e.g., for modeling the location of learning components), specialization rules that define the types of learning components and their specialization (e.g., definitions-, examples-, table of content-, overview-, and summary-components). Furthermore, this model could contain elements that specify the kind of interaction between the system and the user, and adaptation rules for adapting the learning components to learner types or context aspects. These rules are very similar to the contracts used as specifications for interfaces. Beside the conceptual core model for instructional design that is based on learning objectives with related learning activities (see Table 2 right section), a decision model exists that enables the instructional designer to adapt the model. The decision model contains so-called variation points, their resolution space, and their effects on the conceptual model. The variation points mark variable parts of the model that are resolved by questions. The instructional designer uses these questions in order to change the conceptual model in a systematic way. The questions refer to the categorization of learning components (i.e., the instructional designer adapts the categories or the specialization structure), instructional design strategies (i.e., the instructional designer adapts the containment rules, for example, to an *experiential learning* strategy), or the questions consider adaptation aspects (i.e., the instructional designer changes the variable parts of the logical learning component such as their composition of learning elements, see Table 2, middle section). The answers to the questions include solution packages, so that the instructional designer gets support on how to adapt the model. One possibility for defining such solution packages is to use design patterns. They are very useful

for describing instructional design strategies in a comprehensive manner. A design pattern can be understood as a transformation process from one conceptual model state to a new state. Each transformation step relates to specific parts of the model and tells the instructional designer how to change those parts.

Based on these models, frameworks can be derived that can be used for creating learning spaces. The next section describes how a pedagogical agent creates learning spaces.

Learning Space Development

After the selecting of learning goals and appropriate methods as well as the generation of the metamodels, this section explains how a learning space could be created based on a framework from the metamodels.

This last phase can be motivated around the following four questions:

- How can a framework be built from the adapted metamodels in order to develop learning spaces?
- How can this framework be embodied with learning components and elements?
- How can the learning space be presented by means of Wiki pages?
- How can sequencing of learning activities and navigation through a learning space be realized?

Before we answer those questions, a small excursus about pedagogical agents is given. A learning space is created automatically by a so-called pedagogical information agent. Information agents are a special kind of intelligent software agents (Wooldridge & Jennings, 1995). Software agent technology itself originates from distributed artificial intelligence. Software agents have access to multiple, heterogeneous, and geographically distributed information sources. Klusch provides an overview of information agents and describes their main tasks as performing proactive searches, as well as maintaining and communicating relevant information on behalf of their users or other agents. This includes skills such as retrieving, analyzing, manipulating, and fusing heterogeneous information, as well as visualizing and guiding the user through the available individual space (Klusch, 2001).

The pedagogical agent is a special type of information agent: it puts its emphasis especially on the mediation of information by taking into account learner profiles' learning preferences, such as preferred learning styles, presentation modes, and so forth, and creates their learning space based on the metamodel. The difference, as compared to instructional tutoring systems (ITS), lies in the fact that agents react proactively and take into account the current environment the learner is working in, instead of simply analyzing the current status of the learner's knowledge as ITS do. In our approach, a task agent observes specific software engineering tasks that are suitable for monitoring. For example, the activity of programming in an integrated development environment, such as the *Eclipse IDE,* can be monitored. Once a trigger condition, as specified in the design phase, has been observed, the task agent sends a notification message to the pedagogical information agent that has registered interest in the occurrence of particular triggers during the monitored task.

For each instructional design metamodel, a framework is derived that is consistent with the metamodel for learning components, which was developed in the previous phase. An instructional design refines a learning goal into several objectives (see values in the cells of Table 2). Each learning objective refers to a cognitive process (e.g., remember the task of refactoring). As illustrated in Figure 9, arrows between the learning objectives show how the learning objectives should be sequenced in a learning space. The difference between a learning goal and a learning objective is that usually, learning goals are broad, often im-

precise statements of what learners will be able to do when they have completed the learning space. Learning objectives are more specific, have a finer granularity, and are measurable by performing assessments (i.e., through tests, questionnaires). The learning objective network is transformed into the learning component framework. The next step is the embodiment of this framework by learning components and elements.

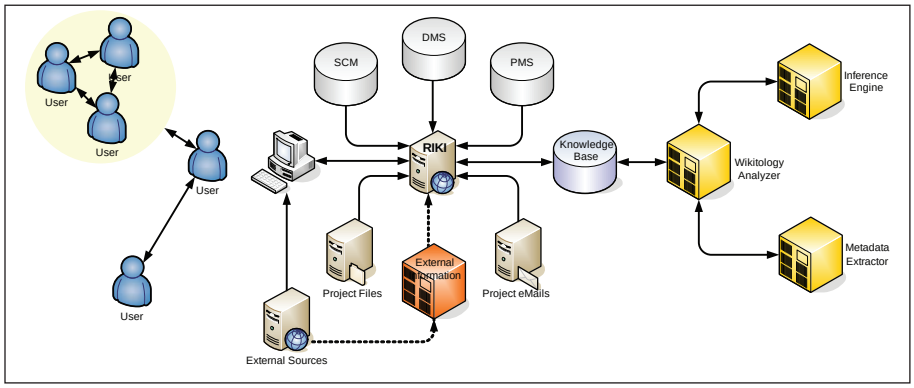
Before the embodiment can take place, each learning object is refined in a learning activity tree (i.e., similar to the SCORM activity tree that could be derived from the SCORM content packages). The tree serves as a help structure for navigation. Each activity tree consists of learning activities that enable the learner to reach the related learning objective (e.g., reading, thinking about a question posed, removing a real code defect). In contrast to the learning objective network, this structure can only be created during run-time, that is, after a notification message has been sent to the pedagogical agent. This structure depends on the following factors: the learning objective, the addressed domain and topic, the available content, and the context. A default structure for each learning objective/topic category pair is defined in the instructional design metamodel. The two latter factors influence the adaptation of these structures during run-time, for example, learning activities have to be abandoned if suitable content is not available, or if the context allows transferring learning activities directly into the working environment instead of keeping it within the learning space environment. The embodiment creates learning components from learning elements by using the domain ontology. Afterwards, each learning activity of the activity tree is extended with a reference to learning components created.

The learning components are presented to the learner by means of Wiki pages. Wiki pages are used within Riki because of their advantages of relating pages; other technologies could also be used instead of Wiki pages. Each learning objective represents one Wiki page, respectively one learning activity tree. They are shown by sections and paragraphs using the Wiki syntax. The difference to a standard Wiki page is that only parts of the Wiki page can be changed by the learner. These parts usually consist of specific knowledge, such as project, customer, people, or product knowledge (see Figure 2 for the knowledge types), or assessment parts (e.g., answering questions etc.). General learning content cannot be changed (e.g., the definition of "Software Quality" or explanation of "Polymorphism"). Only one Wiki page is generated by the agent at a time. The activity tree sometimes offers alternative navigation options to proceed to the next learning objective, respectively to another Wiki page (see link of *LO3* Figure 9). When such a link is chosen, the agent creates the corresponding Wiki page by first resolving the references to learning components and second, by generating navigation options by means of links within the Wiki page or external navigation portlets on the Web site.

By default, sequencing of learning activities on a Wiki page is done hierarchically (see top of Figure 9). Beside the navigation options between learning objectives, additional navigation options could also be offered within the same Wiki page, that is, between learning components. Within the SCORM Sequencing and Navigation model, navigation requests are processed based on a kind of learner model, that is, data about, for example, answered multiple-choice questions that are stored. This data influences which navigation options are available, or whether they are not. In the Riki context, assessments play a minor role. Only data from the software development task and changing learner preferences will be used to adapt the navigation options.

In summary, a learning space consists of several Wiki pages with links forming a hypermedia network. Agent technology allows us to adapt the learning space dynamically during run-time, for example, while the user is browsing through the space and working on his task. Important triggers

Figure 10. Infrastructure around a Riki system



are forwarded by the task agent to the pedagogical information agent, who adapts the content selecting, sequencing, and navigation of the learning space. Observing certain tasks performed by a software engineer and the demand-driven creation of learning spaces ensures close integration of the learning process and the working task, and enables the provision of a situated learning environment for the user where he/she can construct his/her individual knowledge.

Riki: The Reuse-Oriented Wiki

In RISE, we developed a platform and a methodology for the management of experiences in SE organizations, which is integrated smoothly into the infrastructure. In this section, we will elaborate the architecture and technology of a plugin-based, reuse-oriented Wiki (Riki). This technical system sets the stage for the knowledge-based development of software systems strengthened via the support of social relationships, as well as competence development, and knowledge sharing in and between projects. As depicted in Figure 10, three domains or spaces are connected to the core system.

The social-technical system of RISE into which the Riki is embedded consists of the *social space* on the left side, the *technical space* symbolized in the middle, and the *logical space* on the right. The social space encompasses all users of the Riki from the target group(s). They are all accessible via the personal pages and references such as (co-) author or inspector (i.e., know-who entries). In the technical space, all hard- and software systems are collected. The Riki is statically or dynamically (i.e., online) integrated into the existing infrastructure and previously collected information. The integration of information includes internal sources such as software configuration management (SCM) systems, defect management systems (DMS), project management systems (PMS), as well as project files and e-mail messages. For example, SCM systems can be used to extract information about persons who have changed the source code over the different configurations. If a newbie (at least for this part of the system) needs more information about this subsystem, the newbie can directly access these persons and knowledge about it. Furthermore, information from external sources, such as technology specific mailing groups or search engines such as Google, are dynamically integrated.

The Riki Architecture

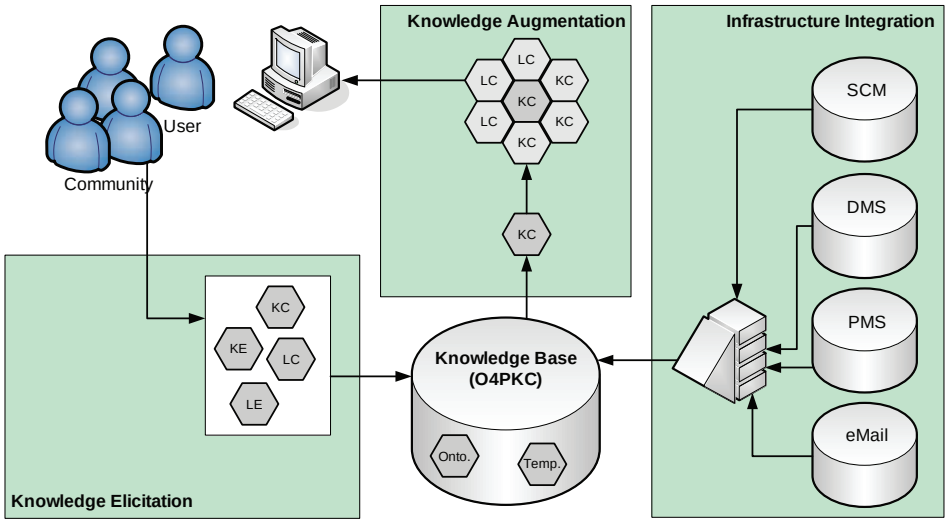
The general plugin concept of the RISE system is a service that is invoked via the Web service interface. The Web service approach was chosen to provide the added functionality independent of the actual implementation. The result of the Web service request is delivered text or simple html. The result is then integrated into the GUI via a content plugin, the templating mechanism of the Wiki, or as a pop-up window.

The RISE system has three different types of plugins that are differentiated based on their integration: Plugins that change the workflow of the underlying Wiki, plugins whose results are integrated into a single point on the GUI, and plugins whose results are integrated into the content.

This integration is supported by plugin interfaces that are already provided by some Wikis: First, Wikis might offer a plugin interface to add dynamic content to pages. The RISE system uses this interface by implementing a general plugin to request Web services and to present the result. Second, Wikis might provide an XML-RPC interface to read and write content of a page. The RISE system uses this interface to standardize the access to Wiki content independent of the actual Wiki used.

The basic knowledge flow of the Riki is shown in Figure 11. It depicts the *elicitation subsystem*, where information from the community of users (or experts) is injected into the system as a first pool of knowledge in the knowledge base. This knowledge as well as the created ontology and approach for creating learning space are used in the *augmentation subsystem* to construct the learning spaces and augmented knowledge (e.g., a knowledge component about a project with context-specific information such as similar projects) that are presented to the user. The *integration subsystem* provides additional knowledge and information from the existing infrastructure.

Figure 11. Basic architecture of the Riki system (KC: Knowledge Component, LC: Learning Component, KE: Knowledge Element, LE: Learning Element, Onto: Ontology, Temp: Template)



RISE IN RETROSPECTIVE

In this section, we will raise requirements and lessons learned from the development and application of a Riki in the context of two SMEs that one should keep in mind when building such a system.

Lessons Learned and Implications for Developing a Riki

During the RISE project, while building two KM systems for SMEs in the software industry, we developed the Riki technology and RIME methodology. From the development and evaluation, we extracted several lessons learned that are stated in the following:

- The KM system should be *tailored* to the characteristics and needs of the organization, its projects, and the target group(s). The process where knowledge is generated, how it is recorded, and who reuses it should be considered. In a medium to large organization (i.e., more than 100 persons), the knowledge should be far more general and self-sufficient than in a small organization (e.g., with five persons). The larger the organization, the more probable it is that the knowledge is used by persons with a totally different context, from another culture, or without the opportunity to ask the author.
- The KM system should be *integrated* into the existing tool landscape and socio-technical infrastructure of the organization. It should represent either integrated relevant information from other systems as a single point of consistent knowledge (SPOCK) or offer links to these other systems. Yet, another system that is used to store information will only be used at the start and then vanish over time as nobody cares to store information in multiple systems.
- The knowledge within the knowledge base should be *goal-oriented*. The author should have a clear organizational, project-specific, or personal goal in mind and identify the problem, potential solutions, and side effects. A simple observational experience that does not inform the reader about a specific problem or solution is of low interest and use in a KM system.
- The knowledge within the knowledge base should allow *individual structuring* by the users. Everybody structures his knowledge individually based on his personal worldview (in German, *weltbild*), which needs to be mapped to the general ontology as implemented in the KM system. This can be realized either by giving all the rights to insert personal links or by mapping between personal, project-specific, departmental, organizational, or community-specific ontologies.
- The knowledge in a knowledge base should be versioned and managed using *authorized groups*. For example, knowledge about marketing processes should only be modifiable by marketing personnel, while knowledge about the interface of two departments (e.g., travel application) should be modifiable by all those involved.
- The plugin-based design and development of a KM system allows defining development tasks for separate subgroups of the project. This enables the independent implementation of functionality and improves the further extension of the system. In contrast to a monolithic system architecture, this furthermore allows the independent testing of plugins, thereby reducing development risks.
- By abstracting from the actual implementation and by implementing plugins independently from other plugins, this architecture allows using different configurations of

plugins for tailored KM systems. This is particularly important when functionality is provided by different partners that should work independently even after the project.

Patterns and Anti-Patterns in Knowledge Management

In the 1990s, a new concept was transferred from architecture to computer science, which helped to represent typical and reoccurring patterns of good and bad software architectures. These design patterns (Gamma, Richard, Johnson, & Vlissides, 1994) and anti-patterns (Brown, Malveau, McCormick, & Mowbray, 1998) were the beginning of the description of many patterns in diverse software phases and products. Today, we have thousands of patterns (Rising, 2000) for topics such as agile software projects (Andrea, Meszaros, & Smith, 2002) or pedagogies (http://www.pedagogicalpatterns.org/) (Abreu, 1997; Fincher & Utting, 2002). Many other patterns are stored in pattern repositories such as the Portland pattern repository (PPR, 2005) or the Hillside pattern library (HPL, 2005), and are being continuously expanded via conferences such as PLOP (Pattern Languages of Programming; see http://hillside.net/conferences/).

While there are similar concepts such as barriers (Riege, 2005; Sun & Scott, 2005) and incentives (Ravindran & Sarkar, 2000) in KM and software reuse (Judicibus, 1996), the identification of patterns seems to be underdeveloped and informal.

We will use the concept of patterns and antipatterns to describe the experience and knowledge we acquired during RISE and several other projects, such as indiGo (Rech, Decker, althoff, Voss, Klotz, & Leopold, 2005), V(I)SEK (Feldmann & Pizka, 2003), or ESERNET (Jedlitschka & Ciolkowski, 2004). In software engineering, design patterns are defined as follows:

- **Design pattern:** A design pattern is a general, proven, and beneficial solution to a common, reoccurring problem in software design. Built upon similar experiences, they represent "best-practices" about how to structure or build a software architecture. An example is the façade pattern that recommends encapsulating a complex subsystem and only allows the connection via a single interface (or "façade") class. This enables the easy exchange and modification of the subsystem.

By transferring the concept of patterns to knowledge management, we therefore define knowledge and knowledge management patterns as follows:

- **Knowledge pattern:** A knowledge pattern is a general, proven, and beneficial solution to a common, reoccurring problem in knowledge design, that is, the structuring and composition of the knowledge (e.g., on or via Wiki pages) or the ontology defining metadata and potential relationships between knowledge components.
- **Knowledge management pattern:** A knowledge management pattern is a general, proven, and beneficial solution to a common, reoccurring problem in knowledge management, that is, the implementation, interconnection, or interface of technical knowledge management systems (e.g., a Wiki system), as well as social methods or systems to foster knowledge elicitation, exchange, or comprehension.

Furthermore, we cluster the patterns into six groups ranging from KM system (Wiki) patterns via content patterns to KM maintenance patterns. We describe several patterns and anti-patterns from three of these groups. In the following, the

Table 3. Knowledge Content Patterns and Anti-Patterns

Name	***Knowledge Blob Anti-Pattern***
Problem	The description of an experience or knowledge component get's larger and larger over time and subsumes more and more information. The search for an arbitrary knowledge component will often include the knowledge blob. The knowledge blob can be used for different problems, has multiple solutions or contact data.
Solution/Countermeasures	• **Compact knowledge:** Summarize and rewrite the knowledge in a shorter form on one page. • **Extract elements:** Apply divide & conquer to create several mutually exclusive pages with parts of the original page. • **Extract commonalities:** Find elements on other pages with overlapping knowledge and extract this overlapping element from both (or all) pages to a new page.
Causes/Consequences	The KM system makes it easy to find and change (extend) a knowledge component; the users are not sensitized to create individual experiences; or there is no maintenance of the knowledge in the KM system.

Name	***Redundant Information Anti-Pattern***
Problem	The description of an experience or knowledge component is too long and has information that is either not relevant to the topic, already stored elsewhere, or outdated. The reader has to read more to get little relevant information, which might lead to an abandoned system. Furthermore, the description is longer than one page in the KM system and requires that the user scrolls (and has to interrupt his learning mode).
Solution/Countermeasures	• **Compact knowledge:** Summarize and rewrite the knowledge in a shorter form on one page • **Offer templates:** Find all knowledge components of a specific type and offer a distinct template for every type.
Causes/Consequences	The writer does not really know what to describe in order to produce a simple, short and comprehensive knowledge component.

Name	***Unnecessary Breakdown Anti-Pattern***
Problem	Multiple pages are used to describe one topic that is not reusable for other knowledge descriptions, and all have to be read to understand the knowledge. The reader has to read several pages in order to understand the knowledge; he/she interrupts his/her learning mode and might interrupt the learning activity altogether. Furthermore, a search on the knowledge base might return only a page within this knowledge chain.
Solution/Countermeasures	• **Compact knowledge:** Summarize and rewrite the knowledge in a shorter form on one page. • See Explicit Start Pattern
Causes/Consequences	The writer does not really know what to describe in order to produce a simple, short and comprehensive knowledge component.

Table 4. Knowledge Maintenance Patterns and Anti-Patterns

Name	***Duplicated Knowledge Anti-Pattern***
Problem	Multiple versions of the same information reside in different locations in the knowledge base. The change of one piece of information causes changes to be made on several pages of different knowledge components. If not all replications are changed as well, multiple, slightly different versions might exist in the knowledge base.
Solution/Countermeasures	• **Compact knowledge:** Summarize and rewrite the knowledge in a shorter form on one page • **Extract commonalities:** Find elements on other pages with overlapping knowledge and extract this overlapping element from both (or all) pages to a new page.
Causes/Consequences	Writers are not aware of or do not care about similar knowledge. Furthermore, either the knowledge base is not cleaned up from time to time, or similar knowledge components are not aggregated.

Name	***Dead Knowledge Antipattern***
Problem	Knowledge is considered useless, is not reused anymore by the users, and wastes space in the knowledge base or computational power (e.g., in search algorithms).
Solution/Countermeasures	• **Fuse knowledge:** Find a similar and "nondead" knowledge component and integrate the remaining useful information (i.e., combine, compact, or rewrite their descriptions). • **Forget knowledge:** Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties).
Causes/Consequences	The knowledge is outdated, too specific, or too general.

Name	***Undead Knowledge Antipattern***
Problem	Knowledge is not used anymore by the system and undiscoverable by the users. While it might be useful to the users it, cannot be reused anymore and wastes space or computational power (e.g., in search algorithms).
Solution/Countermeasures	• **Reintegrate knowledge:** Reintegrate the component in the search index or an applicable navigational structure. • **Fuse knowledge:** Find a similar and "non-dead" knowledge component and fuse them together (i.e., combine, compact, or rewrite their descriptions). • **Forget knowledge:** Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties).
Causes/Consequences	The knowledge is not linked anymore and does not show up in any navigational structures or search results.

format to describe these patterns consists of the pattern name, the description of the problem, the solutions or countermeasures, and causes or consequences. While patterns typically state and emphasize a single solution to multiple problems, antipatterns typically state and emphasize a single problem to multiple solutions.

Knowledge Content Patterns and Anti-Patterns

These patterns and anti-patterns apply to the content of knowledge components or elements and are typically used from the viewpoint of the reader or writer.

See Table 3.

Knowledge Maintenance Patterns and Anti-Patterns

These patterns and anti-patterns apply to the maintenance of knowledge components or elements and are typically used by the knowledge maintainer or gardener.

See Table 4.

CONCLUSION

We have shown that reuse in software engineering needs support in order to work in agile software organizations. Poor documentation and management of knowledge, experiences, decisions, or architectural information accompanies the development of software with agile methods in distributed software organizations. The Wiki technology promises a lightweight solution to capture, organize, and distribute knowledge that emerges and is needed fast in agile software organizations.

The RISE framework sketches our approach for agile reuse and tackles several problems in traditional KM and agile software organizations. Semi-automatic indexing of pages improves the retrieval and enables the semi-automatic creation and evolution of ontologies from Wikis (i.e., Wikitologies). The cooperative adaptation of knowledge to community needs, and the didactic augmentation of the content and interface are targeted to improve the usability of lightweight KM applications in agile environments.

As a basis, we are using Wikis as repositories with fast and liberal access to deposit, mature, and reuse experiences made in agile projects. Our next step is the design and implementation of additional functionality to Wikis with a first version targeted for 2006. In the context of our project, we pursue the following research questions:

- **Are free structures of knowledge and hierarchies more accepted by the users than fixed structures?** A long-term goal would be the development of dynamic or individual structures based on personal arrangement of documents.
- **Does the extraction of information from existing sources (e.g., versioning, defect tracking, etc.) improve the integration and interrelation of knowledge?** This will improve the access to experts and knowledge carriers and will facilitate the build-up of goal-oriented face-to-face communication.
- **Does the didactical augmentation of knowledge improve the understandability and applicability of the knowledge, compared to conventional, non-enriched knowledge descriptions?** A long-term goal is to improve the mechanism for creating learning spaces by considering different instructional designs tailored to software engineers.
- **Is Wiki-based management of knowledge better accepted by the users than classical knowledge management applications in agile processes?** Classical knowledge management applications need a well-defined process to be integrated. Wikis—in particular, if enhanced by ontologies—might provide a solution for agile and hence, less-structured processes.

By using a plugin-based architecture, the Riki system represents a flexible and expandable infrastructure to support reuse in software organizations of different shapes and sizes. The main variability mechanism in this infrastructure is realized by using the plugins as independent services. Currently, this includes only functional aspects of the developed system, but we plan to adapt this idea to knowledge inside the Riki system to improve the reusability of knowledge across software development organizations. By bundling content and functionality, additional complexity is introduced concerning the variability mecha-

nisms needed to establish this product line. This interrelation between content and functionality is subject of the research area of knowledge product lines. The RISE project will provide a first step into this research area.

FUTURE TRENDS

This section discusses future and emerging trends and provides insights about the future of knowledge transfer and reuse in software engineering. A system for knowledge reuse and transfer, such as the Riki, might not only be used in software engineering but also in other domains.

- **OSS reuse—from code to content:** Currently, most of the reuse within OSS is focused on software code. However, with Wikipedia and WikiCommons, there is a growing amount of content (such as music, videos, or research results) available under an open-source-style license. This content will help to overcome the initial seeding problem observed in current reuse systems.
- **Bi-directional openness:** Through open standards and APIs (Web 2.0, Semantic Web), future reuse systems can rely on content and functionality already available to the public. For example, code search engines like Koders.com or Krugle.com can be integrated in reuse systems to provide reusable (code) artifacts from outside the organization.
- **Proactive suggesting instead of searching:** Future reuse systems will make even more use of the context of a user than depicted in Riki. Based on the metadata and their underlying ontologies, inference is done to support the user in generating more high-quality knowledge. In particular, showing similar and relevant content reduces redundancy because people are aware of available content and do not create the content from scratch. Hence, the content is subject to continuous evolution.
- **Amount of guidance during learning:** Riki provides a first significant step towards the integration of e-learning and knowledge management. Nevertheless, several problems remain to be solved and addressed. Riki intends for users to learn at their own pace, and decide which content is suitable. Self-directed learning requires that the system provides a certain amount of guidance and support during learning. For example, experts need a different kind and amount of guidance than novice people.
- **Support for situated learning:** These approaches, developed mainly at the end of the 1980s, emphasize that a human's tasks always depend on the situation they are performed in, that is, they are influenced by the characteristics and relationships of the context (Brown et al., 1989). Because of the relation between cognition and context, knowledge and the cognitive activities meant to create, adapt, and restructure the knowledge cannot be seen as isolated psychological products; they all depend on the situation in which they take place. This means that the Riki has to gather more context information in order to tailor the learning space for situated learning.

Further barriers to integrating learning management and knowledge management were identified during the LOKMOL2005 workshop (Ras, Memmel, & Weibelzahl, 2005). Examples are the lack of interactivity, lack of dynamic adaptation of content, or adequate presentation of content.

To cope with these problems, we see the need for further research and development in the following directions:

- **Ontology usage and development:** The content of reuse systems needs to be indexed according to currently available ontologies

such as SWEBOK, Dublin Core, and FOAF, in order to make it available to inference support. This indexing should be done automatically wherever possible. Based on the experience gained during this indexing, the need for further software ontologies can be derived (e.g., an ontology of software engineering artifacts).

- **Integrated context models:** Besides indexing content according to ontologies, models for describing content information are another issue to be addressed further in the future. Based on context information, knowledge, and learning space can be tailored to the current situation. Different approaches in software engineering exist for describing domains and context. However, they mostly focus on one context dimension (e.g., organizational context, group context, activity context, project context, product context, individual or process context). Context models have to be developed that integrate the different dimensions in order to tailor the content delivery (e.g., by learning space) to the current situation and needs of the software engineer. Standards such as AttentionXML will play a bigger role in context description in the future (see http://developers.technorati.com/wiki/attentionxml).
- **Integrated user models:** In order to provide user tailored content, we need to know about the users' activities, their competence profiles, their learning and working preferences, their roles, and their relationships to other people and teams. The first challenge is to integrate this information in a standardized user model, and the second one is to investigate how this information can be gathered automatically during daily work.

ACKNOWLEDGMENT

Our work is part of the project RISE (Reuse in Software Engineering), funded by the German Ministry of Education and Science (BMBF) grant number 01ISC13D. We thank our colleagues Bertin Klein, Christian Höcht, Lars Kilian, Volker Haas, and Ralph Trapphöner as well as Prof. Klaus-Dieter Althoff, Dr. Markus Nick, Ludger van Elst, Heiko Maus, and Dr. Ingeborg Schüssler for their ideas during the first phases of the project.

REFERENCES

Abreu, F. B. E. (1997). Pedagogical patterns: Picking up the design patterns approach. *Object Expert, 2*(3), 37, 41.

Alrubaie, M. (2006). *A tagging system for Trac*. Retrieved 24/11/2006 from http://trac-hacks.org/wiki/TagsPlugin

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: L. Erlbaum Associates.

Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives* (Complete ed.). New York: Longman.

Andrea, J., Meszaros, G., & Smith, S. (2002). *Catalog of XP project 'smells'*. Paper presented at the 3rd International Conference on XP and Agile Processes in Software Engineering (XP 2002), Alghero, Sardinia, Italy.

Aumüller, D. (2005). *SHAWN: Structure helps a Wiki navigate*. Retrieved 29.9.05, 2005, from http://the.navigable.info/2005/aumueller05shawn.pdf

Basili, V. R., Caldiera, G., & Cantone, G. (1992). A reference architecture for the component factory. *ACM Transactions on Software Engineering and Methodology, 1*(1), 53-80.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Experience factory. In J. J. Marciniak (Ed.), *Encyclopedia of software engineering* (vol. 1, pp. 469-476). New York: John Wiley & Sons.

Basili, V. R., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., et al. (2002). *An experience management system for a software engineering research organization*. Paper presented at the Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop, 2001.

Basili, V. R., & Rombach, H. D. (1991). Support for comprehensive reuse. *Software Engineering Journal, 6*(5), 303-316.

Bergmann, R. (2002). *Experience management: Foundations, development methodology, and Internet-based applications.* Spring New York ISBN 3540441913

Berners-Lee, T. (2000)*Semantic Web layer cake.* Retrieved 29.09.05, 2005, from http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

Bloom, B. S. e., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives; The classification of educational goals* (1st ed.). New York: Longmans Green.

Brickley, D., & Guha, R. V. (2004). *RDF vocabulary description language 1.0: RDF Schema.* Retrieved 24/11/2006 from http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

Brown, J. S., Collins, A., & Duguid, P. (1989). *Situated cognition and the culture of learning* (No. 481). Champaign, IL: University of Illinois at Urbana-Champaign.

Brown, W. J., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis.* New York: John Wiley & Sons Inc.

Bruner, J. S. (1973). *Beyond the information given: Studies in the psychology of knowing* (1st ed.). New York: Norton.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing and mathematics. In L. B. Resnick (Ed.), *Knowing, learning and instruction: Essays in honor of Robert Glaser* (pp. 453-494): Hillsdale, NJ: Lawrence Erlbaum Associates.

Consortium, R. P. (2005). *RISE homepage*. Retrieved from http://www.rise-it.info

Cunningham, W. (2005). *Wiki design principles.* Retrieved from http://c2.com/cgi/wiki?WikiDesignPrinciples

Dahl, I., & Eisenbach, M. (2005). *Anwendung: Semantic Wikis.* Unpublished Seminal Thesis, Karlsruhe.

Decker, B., Ras, E., Rech, J., Klein, B., Reuschling, C., Höcht, C., et al. (2005). *A framework for agile reuse in software engineering using Wiki Technology.* Paper presented at the KMDAP Workshop 2005: Knowledge Management for Distributed Agile Processes, Kaiserslautern, Germany.

Decker, B., Rech, J., Althoff, K.-D., Klotz, A., Leopold, E., & Voss, A. (2005). eParticipative process learning - Process-oriented experience management and conflict solving. *Data & Knowledge Engineering, 52*(1), 5-31.

Dupuis, R., Bourque, P., & Abran, A. (2003). Swebok guide: An overview of trial usages in the field of education. In *Proceedings—Frontiers in Education Conference* (vol. 3).

EclipseWiki. (2005). *EclipseWiki Web site*. Retrieved 6 Oct., 2005, from http://eclipsewiki.sourceforge.net/

Enns, C. Z. (1993). Integrating separate and connected knowing: The experiential learning model. *Teaching of Psychology, 20*(1), 7-13.

Ericsson, K. A., Krampe, R. T., & Tesch-Romer, C. (1993). The role of deliberate practice in the acquisition of expert performance. *Psychological Review, 100*(3), 363-406.

Feldmann, R. L., & Pizka, M. (2003, 6 Aug. 2002). *An on-line software engineering repository for Germany's SME—An experience report.* Paper presented at the 4th International Workshop, Advances in Learning Software Organizations (LSO 2002), Chicago.

Fincher, S., & Utting, I. (2002). Pedagogical patterns: Their place in the genre. *SIGCSE Bulletin, 34* (3), 199-202.

Friesen, N. (2001). What are learning objects? *Interactive Learning Environments, 9*(3), 323-230.

Gagné, R. M., Briggs, L. J., & Wager, W. W. (1988). *Principles of instructional design* (3rd ed.). Fort Worth: Holt Rinehart and Winston.

Gamma, E., Richard, H., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software* (3rd ed. vol. 5): Addison-Wesley.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human Computer Studies, 43*(5-6), 907-928.

HPL. (2005). *Hillside Pattern Library.* Retrieved 10, Oct., 2005, from http://hillside.net/patterns/

Jedlitschka, A., Althoff, K.-D., Decker, B., Hartkopf, S., Nick, M., & Rech, J. (2002). The Fraunhofer IESE experience management system. *KI, 16*(1), 70-73.

Jedlitschka, A., & Ciolkowski, M. (2004, 19-20 Aug. 2004). *Towards evidence in software engineering.* Paper presented at the International Symposium on Empirical Software Engineering, Redondo Beach, CA.

Jedlitschka, A., & Nick, M. (2003). Software engineering knowledge repositories. *Lecture Notes in Computer Science, 2765.*

John, M., Jugel, M., & Schmidt, S. (2005). *Software development documentation —A solution for an unsolved problem?* Paper presented at the International Conference on Agility, Otaniemi, Finland.

Judicibus, D. D. (1996, 8-9 Jan. 1996). *Reuse: A cultural change.* Paper presented at the Proceedings of the International Workshop on Systematic Reuse: Issues in Initiating and Improving a Reuse Program, Liverpool, UK.

Klusch, M. (2001). Information agent technology for the Internet: A survey. *Data & Knowledge Engineering Archive, 36*(3), 337-372.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development.* Englewood Cliffs, NJ: Prentice-Hall.

Legrand, S., Tyrväinen, P., & Saarikoski, H. (2003). *Bridging the word disambiguation gap with the help of OWL and Semantic Web ontologies.* Paper presented at the EROLAN 2003, the Semantic Web and Language Technology, Budapest.

Leuf, B., & Cunningham, W. (2001). *The Wiki way. Quick collaboration on the Web.* Boston: Addison-Wesley.

Manola, F., & Miller, E. (2004). *RDF primer.* Retrieved 24/11/2006 from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

Mason, J. (2005). *From e-learning to e-knowledge. In M. Rao (Ed.), Knowledge management tools and techniques* (paperback ed., pp. 320-328). London: Elsevier.

Maurer, F. (2002). *Supporting distributed extreme programming.* Paper presented at the XP Agile Universe.

McIllroy, M. D. (1968, 7th to 11th October 1968). *Mass-produced software components*. Paper presented at the NATO Conference on Software Engineering, Garmisch, Germany.

Merriënboer, J. J. G. v. (1997). *Training complex cognitive skills: A four-component instructional design model for technical training.* Englewood Cliffs, NJ: Educational Technology Publications.

Merrill, M. D. (2000). *First principles of instruction*. Paper presented at the International conference of the Association for Educational Communications and Technology (AECT), Denver.

Mohan, P., & Daniel, B. (2004). *The learning objects' approach: Challenges and opportunities*. Paper presented at the E-Learn 2004, World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education, Washington DC.

Naur, P., & Randell, B. (1968). *Software engineering: Report of a conference*. Garmisch, Germany: sponsored by the NATO Science Committee.

Nick, M. (2005). *Experience maintenance through closed-loop feedback*. Unpublished PhD thesis, Technical University of Kaiserslautern, Kaiserslautern.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company*. New York: Oxford University Press.

North, K. (2002). *Wissensorientierte Unternehmensführung: Wertschöpfung Durch Wissen* (3., Aktualisierte Und Erw. Aufl. ed.). Wiesbaden: Gabler.

Oezbek, C. (2005). *WikiDoc homepage*. Retrieved September 28, 2005, from http://www.inf.fu-berlin.de/~oezbek/

Paech, B., & Kohler, K. (2003). Task-driven requirements in object-oriented development. In J. H. J. C. D. Sampaio do Prado Leite (Ed.), *Perspectives on Software Requirements* (pp. 45-67)

Palmer, S. B. (2005). *RDFWiki homepage*. Retrieved September 29, 2005, from http://infomesh.net/2001/05/sw/#rdfwik

Platypus. (2005). *Platypus Wiki Web site*. Retrieved October 6, 2005, from http://platypuswiki.sourceforge.net/

PPR. (2005). *Portland pattern repository*. Retrieved October 10, 2005, from http://c2.com/ppr/, http://en.wikipedia.org/wiki/Portland_Pattern_Repository

Ras, E., Memmel, M., & Weibelzahl, S. (2005). *Integration of e-learning and knowledge management - Barriers, solutions and future issues.* Paper presented at the Professional Knowledge Management (WM2005).

Ras, E., & Weibelzahl, S. (2004). *Embedding experiences in micro-didactical arrangements.* Paper presented at the 6th International Workshop on Advances in Learning Software Organisations, Banff, Canada.

Ravindran, S., & Sarkar, S. (2000, May 21-24). Incentives and mechanisms for intra-organizational knowledge sharing. In *Proceedings of 2000 Information Resources Management Association International Conference*, Anchorage, AK (p. 858). Hershey, PA: Idea Group Publishing.

Rech, J., Decker, B., Althoff, K.-D., Voss, A., Klotz, A., & Leopold, E. (2005). Distributed participative knowledge management: The indiGo system. In R. A. Ajami & M. M. Bear (Eds.), *Global entrepreneurship and knowledge management: Local innovations and value creation.* Binghamton, NY: Haworth Press.

Rhizome. (2005). *Rhizome Web site*. Retrieved October 6, 2005, from http://rx4rdf.liminalzone.org/

Riege, A. (2005). Three-dozen knowledge-sharing barriers managers must consider. *Journal of Knowledge Management, 9*(3), p 18-35efs.

Rising, L. (2000). *The pattern almanac 2000.* Boston: Addison-Wesley.

Robbins, J. (2005). *Readyset requirements specification template.* Retrieved 24/11/2006 from http://readyset.tigris.org/

Robertson, J., & Robertson, S. (2005a). *Volere requirements specification template,* Version 10.1. Retrieved October 10, 2005, from http://www.volere.co.uk/template.htm

Robertson, S., & Robertson, J. (2005b). *Volere requirements specification template*. Retrieved November 24, 2006 from http://www.volere.co.uk/

Ruhe, G., & Bomarius, F. (1999, June 16-19, 1999). *Proceedings of learning software organizations (LSO): Methodology and applications.* Paper presented at the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslautern, Germany.

Rus, I., & Lindvall, M. (2002). Knowledge management—Knowledge management in software engineering—Guest Editors' Introduction. *IEEE Software, 19*(3), 26-38.

Schank, R. C., Bermann, T. R., & Macperson, K. A. (1999). Learning by doing. In R. R. C. (Ed.), *Instructional design theories and models: A new paradigm of instructional theory* (vol. II, pp. 161-181). Mahwah: NJ: Lawrence Erlbaum Associates.

Schreiber, G., Wielinga, B., Hoog, R. d., Akkermans, H., & Velde, W. V. d. (1994). *CommonKADS: A comprehensive methodology for KBS development IEEE Expert: Intelligent Systems and Their Applications, 9*(6), 28-37

Secchi, P., Ciaschi, R., & Spence, D. (1999). *A concept for an ESA lessons learned system* (No. Tech. Rep. WPP-167). Noordwijk: The Netherlands: ESTEC.

Self, J. (1992). Computational mathematics: The missing link in intelligent tutoring systems research? *Directions in Intelligent Tutoring Systems,* (91), 36-56.

Semantic Wikis. (2005). *Semantic Wiki overview.* Retrieved October 6, 2005, from http://c2.com/cgi/wiki?SemanticWikiWikiWeb

Senge, P. M. (1990). *The fifth discipline: The art and practice of the learning organization* (1st ed.). New York: Doubleday/Currency.

Simons, C. L., Parmee, I. C., & Coward, P. D. (2003). 35 years on: To what extent has software engineering design achieved its goals? in *IEEE Proceedings Software, 150*(6), 337-350.

Smith, M. K., Welty, C., & McGuinness, D. L. (2004). *OWL Web ontology language guide.* Retrieved November 24, 2006, from http://www.w3.org/TR/2004/ REC-owl-guide-20040210/

Software Engineering Body of Knowledge, Iron Man Version. (2004). Retrieved November 24, 2006, from http://www.swebok.org/ironman/pdf/Swebok_ Ironman_June_23_%202004.pdf

Souzis, A. (2001). *Rhizome position paper.* Retrieved September 29, 2005, from Adam Souzis.

Stenmark, D. (2001). *The relationship between information and knowledge.* Paper presented at the IRIS-24, Ulvik, Norway.

SubWiki. (2005). *SubWiki Web site.* Retrieved October 6, 2005, from http://subwiki.tigris.org/

SUMO Ontology. from http://ontology.teknowledge.com/

Sun, P. Y. T., & Scott, J. L. (2005). An investigation of barriers to knowledge transfer. *Journal of Knowledge Management, 9*(2), 75-90.

Szyperski, C., & Pfister, C. (1997). *Workshop on component-oriented programming, summary.* Paper presented at the ECOOP96.

Tautz, C. (2001). *Customizing software engineering experience management systems to organizational needs.* Unpublished PhD thesis, University of Kaiserslautern, Kaiserslautern.

Tennyson, R. D., & Rasch, M. (1988). Linking cognitive learning theory to instructional prescriptions. *Instructional Science, 17,* 369-385.

TikiWiki. (2005). *TikiWiki Web site*. Retrieved October 6, 2005, from http://www.tikiwiki.org

Volere. (2005). *Requirements tools.* Retrieved from http://www.volere.co.uk/tools.htm

Völkel, M., Schaffert, S., Kiesel, M., Oren, E., & Decker, B. (2005). *Semantic Wiki state of the art paper.* Retrieved from November 24, 2006, from http://wiki.ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art_Paper

Weber, R., Aha, D. W., & Becerra-Fernandez, I. (2001). *Intelligent lessons learned systems. Expert Systems with Applications, 20*(1) 94-100.

WikiEngines. (2005). *WikiEngines Compilation.* Retrieved October 6, 2005, from http://www.c2.com/cgi/wiki?WikiEngines

WikiMatrix. (2006). *WikiMatrix—Overview of Wikis.* November 24, 2006, from http://wikimatrix.org

Wikipedia. (2006). *Your first article*. Retrieved from http://en.wikipedia.org/w/index.php?title=Wikipedia:Your_first_article&oldid=41631731

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review, 10*(2), 115-152.

XWiki. (2005). *XWiki Web site.* Retrieved October 6, 2005, from http://www.xwiki.org/xwiki/bin/view/Main/WebHome

Yacci, M. (1999). The knowledge warehouse: Reusing knowledge components. *Performance Improvement Quarterly, 12*(3), 132-140.

APPENDIX I: INTERNET SESSION: KNOWLEDGE AND SOFTWARE REUSE

http://www.sei.cmu.edu/productlines/index.html
Software product lines are a way to describe commonalities and variabilities in a software system that are used in different contexts by different groups of people such as embedded software systems on mobile phones (each with different hardware characteristics and software features).

Interaction

Survey the information presented at the websites on product-line software engineering (PLSE) and software reuse methods and theory. Prepare a brief presentation on the core concepts and history of software reuse with a focus on PLSE. Alternatively, transfer the ideas behind PLSE to knowledge or learning management and assume that "software = knowledge" or "software = course". How would a "knowledge product line" or "course product line" look like? Are there commonalities or variabilites in KM/LM systems or the knowledge/courses itself?

APPENDIX II: USEFUL URLS

RISE: Web site of the RISE project:
http://www.rise-it.info

SWEBOK: The software engineering body of knowledge, with more information on SE and software reuse (see page 4-4):
http://www.swebok.org/

ICSR-09: The Ninth Biannual International Conference on Software Reuse on June 12-15, 2006 in Torino, Italy:
http://softeng.polito.it/ICSR9/

Lombard-Hill's bibliography: The largest bibliography on literature about software reuse:
http://www.lombardhill.com/biblio1.html

The TOA bibliography: A similar large bibliography:
http://www.toa.com/pub/reusebib.htm

Sverker Janson: Internet survey on Software Agents and Agent-based Systems:
http://www.sics.se/isl/abc/survey.html

A collection of arguments why Wikis work:
http://c2.com/cgi/wiki?WhyWikiWorks and **http://en.wikipedia.org/wiki/Wikipedia:Our_Replies_to_Our_Critics**

An overview of Wikis with a comparison feature:
http://www.wikimatrix.org

Different types of knowledge that might be taken into consideration when building a KM system (e.g., for knowledge flow descriptions or templates in a KM system):
http://www.knowledge-sharing.com/TypesOfKnowledge.htm

Pattern in general: Descriptions of patterns with links to patterns in architecture:
http://en.wikipedia.org/wiki/Patterns

Software patterns: Starting page with information about patterns in software engineering:
http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29

Pedagogical patterns for seminars and teaching:
http://www.pedagogicalpatterns.org/

The Hillside Pattern Library:
http://hillside.net/

The Portland pattern repository:
http://c2.com/ppr/ and, http://en.wikipedia.org/wiki/Portland_Pattern_Repository

Classification and references for patterns based on the book "The pattern Almanac:"
http://www.smallmemory.com/almanac/

Hillside Pattern bibliography:
http://hillside.net/patterns/papersbibliographys.htm

Quality overview: The paper "Construction of a systemic quality model for evaluating a software product" gives a nice overview about several software quality models:
http://www.lisi.usb.ve/publicaciones/SQJ%2011%203%202003%20Ortega%20Perez%20and%20Rojas.pdf

ISO 9126: Part 1 of the Software Quality standard with a focus on Quality models:
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22749

Dromey's quality model in the paper "A Model for Software Product Quality:"
http://www.sqi.gu.edu.au/docs/sqi/technical/Model_For_S_W_Prod_Qual.pdf

APPENDIX III: FURTHER READINGS

Biggerstaff, T. J., & Perlis, A. J. (1989). *Software reusability: Volume I, Concepts and models.* ACM Press.

Biggerstaff, T. J., & Perlis, A. J. (1989). *Software reusability: Volume II, Applications and experience.* ACM Press.

Cunningham, W. & Leuf, B. (2001). *The Wiki way. Collaboration and sharing on the Internet.* Addison-Wesley.

Fensel, D. (2000). *Ontologies: Silver bullet for knowledge management and electronic commerce.* Berlin: Springer-Verlag.

Fowler, M. (1999). *Refactoring: Improving the design of existing code* (1st ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1997). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software reuse: Architecture, process and organization for business success.* ACM Press.

Karlsson, E. (1995). *Software reuse: A holistic approach.* John Wiley & Sons Ltd.

Mili, H., Mili, A., Yacoub, S., & Addy, E. (2002). *Reuse-based software engineering.* John Wiley & Sons Inc.

Kerievsky, J. (2005). *Refactoring to patterns.* Boston: Addison-Wesley.

Rising, L. (2000). *The pattern almanac.* Boston.

Roock, S., & Lippert, M. (2005). *Refactoring in large software projects.* John Wiley & Sons.

Sametinger, J. (1997). *Software engineering with reusable components.* Springer-Verlag.

Schaefer, W., Prieto-Diaz, R., & Matsumoto, M. (1994). *Software reusability.* Ellis Horwood.

Staab, S. & Studer, R. (Eds.). (2004). Handbook on ontologies. In *International Handbooks on Information Systems.* Springer.

Possible Titles for Papers/Essays

- Commonalities and Variabilities of Software Reuse and Knowledge Management
- Software Patterns for Knowledge and Knowledge Management
- Knowledge Management in Learning Software Organizations: Methods and Tools
- An Effective Knowledge Management System
- Knowledge Management in Software Engineering: Problems and Research Directions

This work was previously published in Open Source for Knowledge and Learning Management: Strategies Beyond Tools, edited by M. Lytras and A. Naeve, pp. 52-121, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.21
Constructivist Learning During Software Development

Václav Rajlich
Wayne State University, USA

Shaochun Xu
Laurentian University, Canada

ABSTRACT

This article explores the non-monotonic nature of the programmer learning that takes place during incremental program development. It uses a constructivist learning model that consists of four fundamental cognitive activities: absorption that adds new facts to the knowledge, denial that rejects facts that do not fit in, reorganization that reorganizes the knowledge, and expulsion that rejects obsolete knowledge. A case study of an incremental program development illustrates the application of the model and demonstrates that it can explain the learning process with episodes of both increase and decrease in the knowledge. Implications for the documentation systems are discussed in the conclusions.

INTRODUCTION

One of the puzzling issues of software engineering is the nature of the knowledge that is needed in order to develop and evolve a program. The program itself is a repository of knowledge about the program domain and may contain knowledge that is not available elsewhere, as documented by Kozaczynski and Wilde (1992). It also contains knowledge of all design decisions that were made during the program development and consequent program evolution (Rugaber, Ornburn, & LeBlanc, 1990). When evolving or maintaining the program, it is necessary to recover this knowledge; otherwise, maintenance or evolution will be impossible. It is also necessary to communicate this knowledge to all new programmers who are joining an existing software project. The loss of the programming knowledge can be a serious

problem and was identified as a leading cause of the code decay (Rajlich & Bennett, 2000).

Although the knowledge is embedded in the program, it cannot be easily recovered since it is encoded in programming structures and delocalized into different components of the program. Moreover, the consequences of the decisions, rather than the decisions themselves, appear in the code. In many ways, the recovery of knowledge from the code is similar to solving a puzzle and is laborious and error prone.

One of the most basic questions that concerns the nature of the programmer knowledge is the issue of its monotonicity. According to a naïve view, the knowledge steadily increases, as the new facts emerge and are absorbed by the programming team; many current documentation systems are geared towards that (Ye, 2006). However, in this article we show that there are also episodes of the knowledge retraction, and the documentation systems should provide an adequate support for that also.

Our approach in the article is based on cognitive informatics (CI). CI is a multidisciplinary study of cognition and information sciences, which investigates human information processing mechanisms and processes and their applications in computing (Wang & Kinsner, 2006); studying the knowledge and cognitive process involved in software development is one of the goals of cognitive informatics.

In order to understand the nature of programming knowledge and its acquisition, we adopted and further developed a constructivist model of programmer learning that is based on four basic cognitive activities: absorption, denial, reorganization, and expulsion of the knowledge. We validated this model in a case study of the pair programming that is a part of eXtreme Programming (Martin, 2002). In pair programming, two programmers work side-by-side at one machine as they collaborate in program design, implementation, and testing. The programming pair has to communicate and share the knowledge, and this gives an opportunity to analyze unobtrusively their dialog for the indications of the programmer knowledge and learning.

The first section of this article describes our theory of constructivist learning. The second section describes the case study. The third section contains the discussion of the results of the case study and the fourth section has an overview of the related literature. The fifth section contains general conclusions and future work.

THEORY OF CONSTRUCTIVIST LEARNING

The constructivist learning model is based on the work of Piaget (Piaget, 1954). The original aim of Piaget was to explain learning in children, but the constructivist theory extends to adult learning and to epistemology (von Glasersfeld, 1995). The theory assumes that the learners actively and incrementally construct their knowledge. They start from some preliminary knowledge, and they extend it by adding new facts to it; they may go through stages in which they may accept ideas that they will later discard as wrong. The two main activities are assimilation and accommodation, where assimilation describes how learners deal with new knowledge, and accommodation describes how learners reorganize their existing knowledge.

We modified this theory by dividing assimilation into two separate activities. Absorption means that the learners add new facts to their knowledge. However, if the new facts do not fit in, the learners may reject them; we call this second activity a denial. We also divided accommodation into two separate activities. Reorganization means that the learners reorganize their knowledge to aid future absorption of new facts. Expulsion is the process where part of the knowledge becomes obsolete or provably incorrect and the learners reject it. Of course, there are also mixed activities: learners may absorb a modified fact, rather than make an

Table 1. Learning activities

Activity	Symbol	Characterization of the activity
Absorption	a	The learners add new facts to their knowledge.
Denial	d	The learners reject the new facts that do not fit in.
Reorganization	r	The learners reorganize their knowledge to aid future absorption.
Expulsion	e	A part of the knowledge becomes obsolete and the learners reject it.

outright denial; learners may reorganize their knowledge while absorbing new facts, and so forth. Table 1 lists the four basic learning activities.

In order for learning to occur, the learners must possess *preliminary knowledge*. Preliminary knowledge makes learning possible; the more the learners know, the more they can learn. Sometimes this preliminary knowledge turned out to be inaccurate or even completely wrong, and the learners employed the four cognitive activities to build more accurate knowledge. The theory is particularly suitable to the situations where learners must discover the facts of the knowledge on their own, without a teacher, which is a common situation in software engineering.

The assertion of this theory is the following: When the process of learning is recorded and divided into episodes, every episode can be classified in terms of the categories of the constructivist learning model. The model would be falsified (Popper, 2003) if there were episodes that laid outside of our classification scheme, or if independent observers frequently arrived at different conclusions about the same observed episode, indicating that the classifications are arbitrary.

Each episode deals with specific concepts that are part of the knowledge. In the context of program development, the concepts can be classified as either domain concepts or programming concepts. Domain concepts belong to a specific domain that the program addresses (Biggerstaff, Mitbander, & Webster, 1994), while programming concepts belong to the knowledge of programming, such as the programming language, program development process, design decisions, and so forth. Design decisions involve program architecture, selection of the program classes, methods, or attributes, and so forth (Ran & Kuusela, 1996).

The leaning process can be explained in terms of analogy with incremental program development. The knowledge that is constructed by the programmers is analogous to the program they

Table 2. Analogy between programming activities and cognitive activities

Programming Activities	Cognitive Activities
Incremental change	Absorption
Rejection of change request	Denial
Refactoring	Reorganization
Retraction	Expulsion

incrementally develop, and the activities of their learning are analogous to the activities of incremental development. The analogy is summarized in Table 2 where analogous terms are in the same row. The case study of the next section examines an application of this model.

THE CASE STUDY

For our case study, we utilized the eXtreme Programming process (Beck, 2000), where pair programming is one of the recommended practices. Pair programming offers a unique opportunity to study the parallel construction of both the program and the programmer knowledge. In pair programming, the programmers communicate with each other about the evolving program. By recording and analyzing their dialog, we study how both the knowledge and program grow.

The program developed during this case study records bowling scores (Martin, 2002). The programming was done by two experts who have been working in the industry for more than 25 years (Martin, 2002).

The programming pair started the process with the preliminary knowledge of the domain (bowling rules), represented by the concept map

Figure 1. Domain concepts

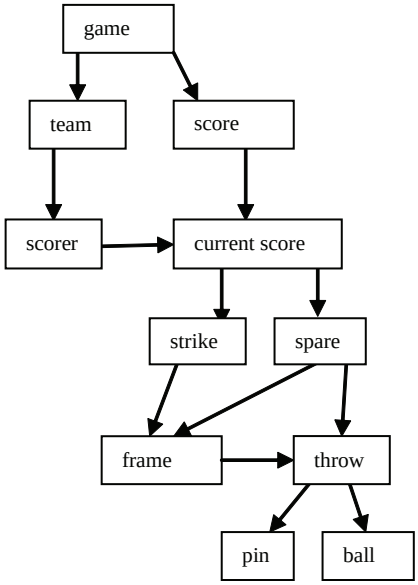


Figure 2. UML diagram of the first version

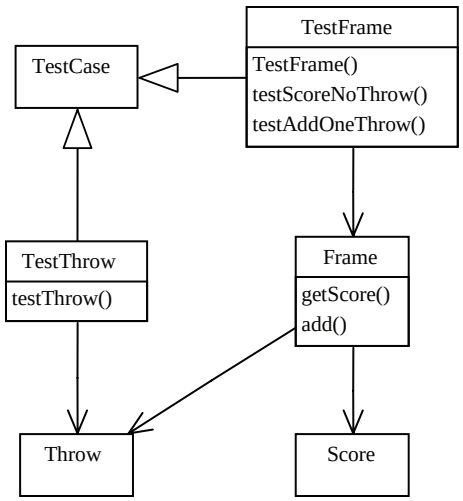
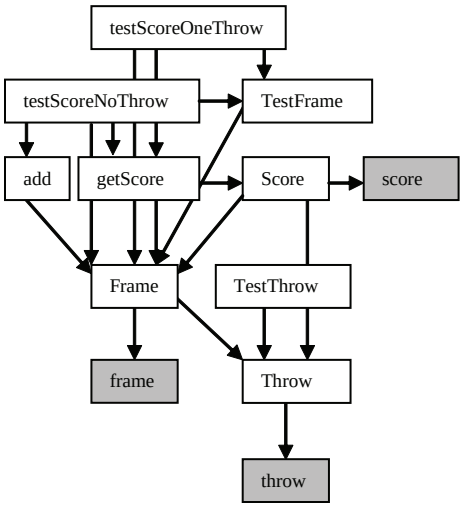


Figure 3. Design decisions for the first version



in Figure 1 (Novak, 1998), where concepts are represented by rectangles and the arrows represent the dependencies. The dependencies stand for the order in which the concepts have to be explained to somebody who is not familiar with the domain; a concept can be explained only if all previous concepts on which it is dependent have been explained and understood. Preliminary programming knowledge includes knowledge of the programming language, algorithms, eXtreme Programming practices, and so forth.

Equipped with this knowledge, the programmers implemented a sequence of the program versions and recorded their dialog. The UML class diagram (Fowler, 1999) of the first version is in Figure 2. The design decisions that led to this diagram were extracted from the recorded dialog and appear in Figure 3. The rectangles represent design decisions, while arrows represent the order in which the design decisions were made. Dark rectangles represent domain concepts that serve as the basis of some of the design decisions. Please

Figure 4. Design decisions after episode 87

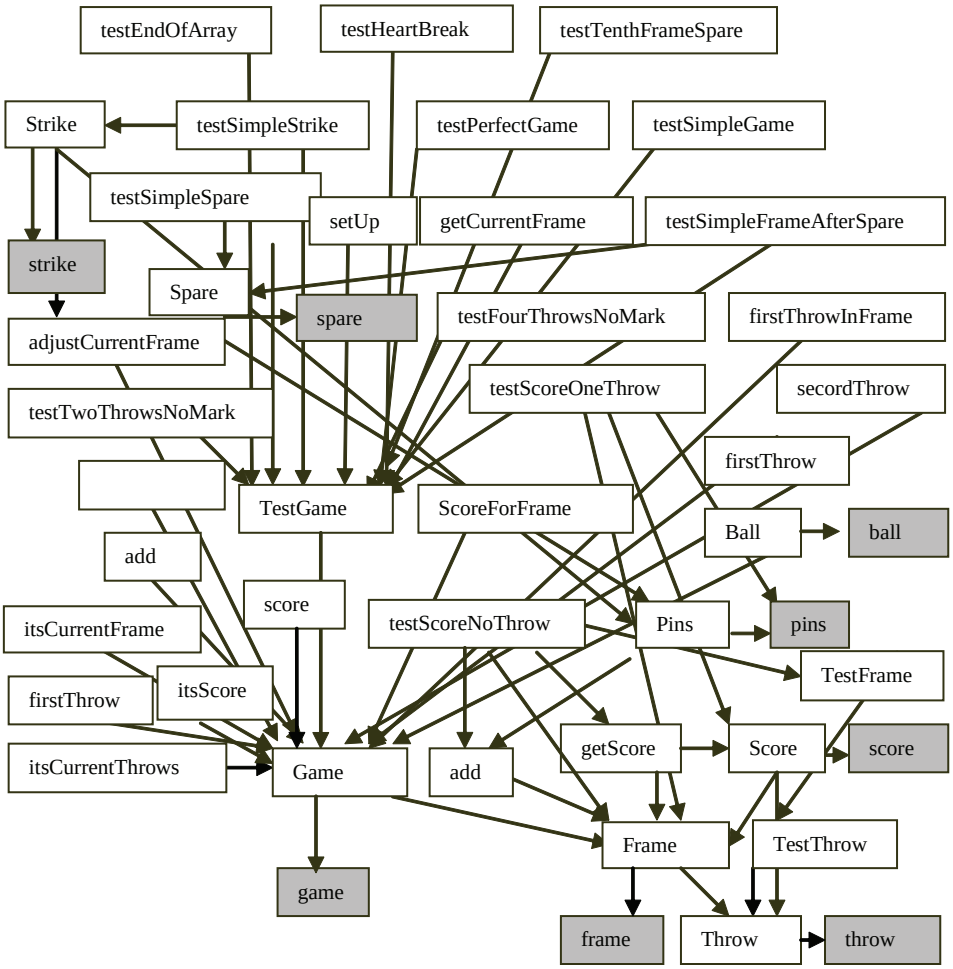
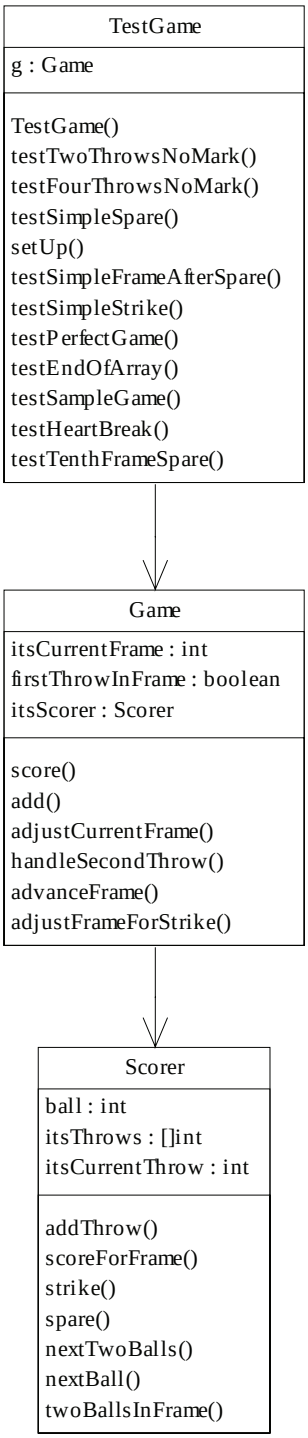


Figure 5. Final program



note that the order of the design decisions does not correspond to the order of the dependencies in the UML diagram. If the two orders were identical, that would mean that all design decisions were done in bottom-up order.

The development continued through several steps; see the progress of the learning in Figure 4 that shows the situation toward the end of the dialog. Programmers realized that the class "Frame" and the related classes do not contribute toward the functionality of the program and deleted them. The resulting UML class diagram of the program is in Figure 5.

We divided this dialog into 94 small episodes, and then three independent observers classified each episode as one of the four cognitive activities, using the characterization of the activities in Table 1. Out of 94 episodes, only 5 episodes were classified differently by each of the observers; in 30 episodes there was a partial disagreement where one observer differed from the other two; in these cases we discussed the cases and chose the majority opinion as the final classification. While acknowledging these differences, we found the classification in general was acceptable and reliable. The sample dialog and its resulting classification are shown in Table 3.

THE RESULT OF THE CASE STUDY

In the case study, we observed that the knowledge required by even a small program is quite extensive, as demonstrated in Figure 4. It should be remembered that these figures are only the "tip of the iceberg," as there is a large preliminary knowledge of the domain and programming that these figures do not capture. Yet all that knowledge is necessary to evolve the program.

During the incremental program development, the most common cognitive activity was absorption (about 71.3%) as shown in Table 4,

Table 3. Fragment of the dialog and the classification

No	Programmers' actions	Activity	Concepts updated
35	**K**: Why the magic number 21? **M**: That's the maximum possible number of throws in a game. **K**: **scoreForFrame** needs to be refactored to be more communicative.	r	
36	**K**: But before we consider refactoring, let me ask another question: Is **Game** the best place for this method? In my mind, **Game** is violating Bertrand Meyer's SRP (Single Responsibility Principle). It is accepting throws *and* it knows how to score for each frame. What would you think about a **Scorer** object?	a	Game, Scorer
37	**M**: But there are side-effects in the **score+=** expression. They don't matter here because it doesn't matter which order the two **addend** expressions are evaluated in. **K**: I suppose we could do an experiment to verify that there aren't any side-effects, but that function isn't going to work with spares and strikes. Should we keep trying to make it more readable or should we push further on its functionality?	d	
38	**M**: The experiment would only have meaning on certain compilers. Other compilers might use different evaluation orders. Let's get rid of the order dependency and then push on with more test cases.	r	
39	**M**: Next test case. Let's try a spare. **K**: Let's refactor the test and put the creation of the game in a **setUp** function.	r	
40	**M**: That's better now. Let's write the spare test case. I think the increment of ball in the **frameScore==10** case shouldn't be there. Here's a test case that proves my point.	a	Spare
41	**M**: See, that fails. Now if we just take out that pesky extra increment. It still fails.... Could it be that the **score** method is wrong?	d	Score
42	**M**: I'll test that by changing the test case to use **scoreForFrame (2).**	a	
43	**M**: That passes. The **score** method must be messed up.	a	
44	**M**: That's wrong. The **score** method is just returning the sum of the pins, not the proper score. What we need **score** to do is call **scoreForFrame** with the current frame.	a	Score
45	**K**: We don't know what the current frame is. Let's add that message to each of our current tests, one at a time.	a	
46	**M**: OK, that works. But it's stupid. Let's do the next test case.	a	
47	**M**: Let's try the next. This one fails. Now let's make it pass.	a	
48	**K**: I think the algorithm is trivial. Just divide the number of throws by two, since there are two throws per frame. Unless we have a strike ... but we don't have strikes yet, so let's ignore them here too. What if we don't calculate it each time? What if we adjust a **currentFrame** member variable after each throw?	a	Frame

and it dominated the programming process from the beginning to the end. This coincides with the intuition that the knowledge increases during the program implementation.

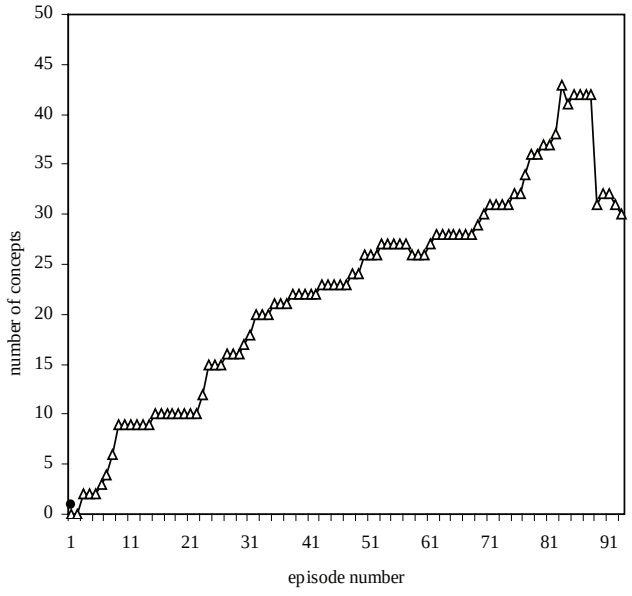
While absorption was the driving force of the learning, there was one large episode of knowledge expulsion in the last part of the dialog. Class "Frame" of Figure 4 and six related design decisions out of the total of 39 were retracted as a part of code "cleaning," resulting in a substantial knowledge change. This episode illustrates the non-monotonic nature of constructivist learning. The changing number of concepts are presented in Figure 6.

There were also episodes of reorganization, which amounts to 17% of all episodes. The rejection of concepts "Score Card" and "Team" in the first part of the dialog are examples of denial.

Table 4. The distribution of the cognitive activities in the case study

Activities	**beginning**	**middle**	**end**	**total**	**percentage**
Absorption	27	18	22	67	71.3%
Denial	1	4	0	5	5.3%
Reorganization	0	7	9	16	17%
Expulsion	2	1	3	6	6.4%
Total	30	30	34	94	100%

Figure 6. Changing numbers of the concepts and their dependencies



We observed that the absorption occurred mostly at the beginning and middle of the dialog, while the expulsion happened most often in the last part. Reorganization appeared mostly in the middle and last parts, and denial appeared only in the first and middle parts of the dialog.

Programmers discussed nine domain concepts in three episodes before the coding started, but during the rest of the development almost all the domain concepts were revisited, and their comprehension was updated. The domain concept "Frame" was updated as many as nine times, with each update resulting in more accurate knowledge. Many design decisions were also updated, but less frequently than domain concepts. As a result of the case study, we conclude that the constructivist learning model explains the learning that takes place in incremental software development and that the learning is significantly non-monotonic.

RELATED WORK

CI is a multidisciplinary research area that is at the intersection of cognitive science, neural psychology, philosophy, software engineering, and other disciplines (Wang, 2002). Wang (2004) previously investigated the relationship between software engineering and cognitive informatics.

There have been numerous publications on constructivist learning, including the classical work of Piaget (Piaget, 1954). von Glasersfeld (1995) described constructivism as a theory of knowledge and defined radical constructivism as a theory of learning. Novak (1998) addressed theories of learning, knowledge, and instruction and used concept maps as tools to describe the knowledge. The knowledge graphs of this article follow Novak's approach.

Other notable learning theories include Vygotsky (1978), who proposed the social cognition learning that emphasizes social interaction in the development of cognition. Although social interaction is an important element during software engineering processes, we believe that our emphasis must be on the individual programmers and their autonomous construction of their knowledge, and hence we found this theory less applicable.

Observational learning theory states that learning occurs through the simple processes of observing someone else's activity (Biederman, Stepaniuk, Davey, Raven, & Ahn, 1999). However, software engineering processes involve more complicated processes than simple observation; therefore, observational learning is not suitable here. Behaviorism focuses on the objectively observable behaviors at the expense of mental activities; therefore, we do not use it for explanation of software engineering activities.

Brain-based learning is based on the structure and function of the brain and emphasizes the fact that the brain can perform several activities at once, like tasting and smelling. Learning involves both focused attention and peripheral perception and both conscious and unconscious processes (Jensen, 2000). Because brain-based learning deals with the functions of the brain rather than the cognitive activity of learners, we did not use it.

Control theory claims that behavior is not caused by a response to an outside stimulus, but inspired by what a person wants most, such as survival, love, and freedom (Glasser, 1998). Again we did not find that view useful for the study of software engineering learning.

We concluded that although these theories contain valuable insights, they are not directly applicable to our purposes. The most promising theories we found include the constructivist learning, which we adopted as part of our model of constructivist learning.

Incremental software development has been described in numerous publications, for example Beck (2000), Martin (2002), and Williams, Kessler, Cuningham, and Jeffries (2000). Beck (2000) introduced a new approach to software

development, called eXtreme Programming that is based on practices of incremental development, continuous testing, pair programming, and several other practices.

Several researchers have studied knowledge contained in a program. According to Brooks (1983), a programmer understands a program through construction of a mental model that consists of successive knowledge domains. Fischer, McCall, Ostwald, Reeves, and Shipman (1994) proposed a support for the incremental development based on a specific knowledge model, where seeding, evolution, and reseeding are the three stages of knowledge capture and transformation. Henninger (1997) recognized that software development is a process involving various knowledge resources, which keep changing during the process. Robillard (1999) identified two types of knowledge: topical and episodic. Topical knowledge refers to the meaning of words, and episodic knowledge consists of people's experience with knowledge.

In our previous work, Rajlich (2002) presented the program comprehension as a learning process. In a case study of an incremental software development, Rajlich and Xu (2003) described an analogy between incremental software development and constructivist learning. A study of program debugging was based on Bloom's taxonomy (Bloom, 1956; Xu & Rajlich, 2004). Xu and Rajlich (2005) developed a dialog-based protocol, a novel empirical research method that is based on the analysis of the dialogs in a programming pair, and give an insight into the programmer cognitive activities. This approach may reduce Hawthorne and placebo effects that are present in other empirical techniques.

Many existing software documentation tools do not pay sufficient attention to the changes in knowledge and essentially assume that the knowledge is unchanged (Forward & Lethbridge, 2002). Among them, JavaDoc extracts documentation statically from Java source files and produces formatted HTML output (Gosling, Joy, & Guy, 1996), Doc++ (Wunderling & Zцckler, n.d.) is useful for creating hierarchical documentations of class libraries; Doxygen (2004) is used for documentation of Java, C, and C++ programs. Donald Knuth's Literate Programming (Knuth, 1984) places both source code and documentation into the same file. None of these systems directly supports non-monotonicity in the construction of the knowledge. PAS (Rostkowycz, Rajlich, & Marcus,2004) is a hypertext-based documentation system suitable for incremental redocumentation

CONCLUSION AND FUTURE WORK

In this article, we introduced a model of constructivist learning and used it to explain programmer learning during a software engineering process. The model is based on the four cognitive activities: absorption that adds new facts to the knowledge, denial that rejects facts that do not fit in, reorganization that reorganizes the knowledge, and expulsion that rejects obsolete knowledge.

We validated the model in a case study of pair programming during incremental program development. The data of this article are based on the analysis of the dialog in a programming pair. From the data we concluded that the classification of programmers' actions according to the constructivist model of learning reflects well the process of learning.

We noted that the knowledge required for incremental software development is large and changes rapidly. We plan to investigate nature of this knowledge in more detail and see whether there are any particular substructures of this knowledge that require greater programmer attention.

We observed episodes of both knowledge increase and decrease and hence the growth of the knowledge in non-monotone. This fact is important for the program documentation systems.

Future work includes additional studies of pairs of programmers during the development of larger programs, in order to further assess the insights provided by the constructivist learning, and to analyze the differences between novices and experts. We also plan to develop specialized documentation tools that will support the non-monotonic nature of the knowledge growth during the program development.

ACKNOWLEDGMENT

The authors would like to acknowledge Jay Rajnovich for his help with writing this article and Claudia Iacob for being one of the observers who classified the episodes. We also thank the anonymous reviewers for their comments that significantly improved this article. This research was supported in part by grants from the National Science Foundation (CCF-0438970), the National Institute for Health (NHGRI 1R01HG003491), and by 2005 and 2006 IBM Faculty Awards. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NIH, or IBM.

REFERENCES

Beck, K. (2000). *Extreme programming explained*. MA: Addison-Wesley.

Biederman, G. B., Stepaniuk, S., Davey, V. A., Raven, K., & Ahn, D. (1999). Observational learning in children with Down syndrome and developmental delays: The effect of presentation speed in videotaped modeling. *Down Syndrome Research and Practice, 6*(1), 12-18.

Biggerstaff, T. J., Mitbander, B. G., & Webster, D. E. (1994). Program understanding and the concept assignment problem. *Communication of the ACM, 37*(5), 72-82.

Bloom, B. S. (Ed.). (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook, I, Cognitive domain*. New York, Toronto: Longmans, Green.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies, 18*(6), 543-554.

Doxygen. (2004). *Doxygen Web site*. Retrieved January 7, 2005, from http://www.stack.nl/~dimitri/doxygen/

Fischer, G., Mccall, R., Ostwald, J., Reeves, B., & Shipman, F. (1994). *Seeding, evolutionary growth and reseeding: Supporting the incremental development of design environments*. Paper presented at the Conference on Computer-Human Interaction (Chi'94), Boston, MA.

Forward, A., & Lethbridge, T. (2002). *The relevance of software documentation, tools and techniqies: A survey*. Paper presented at the ACM Symposium on Document Engineering, Mclean, VA.

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. MA: Addison-Wesley.

Glasser, W. (1998). *The quality school*. Perennial.

Gosling, J., Joy, B., & Guy, S. (1996). *Java language specification*. MA: Addison-Wesley.

Henninger, S. (1997). *Tools supporting the creation and evolution of software development knowledge*. Paper presented at the International Conference on Automated Software Engineering (ASE'97), Incline Village, NV.

Jensen, E. (2000). *Brain-based learning: The new science of teaching and training* (revision ed.). Brain Store Inc.

Knuth, D. (1984). Literate programming. *The Computer Journal, 27*(2), 97-111.

Kozaczynski, W., & Wilde, N. (1992). On the re-engineering of transaction systems. *Journal of software maintenance, 4*, 143-162.

Martin, R. C. (2002). *Agile software development, principles, patterns, and practices.* MA: Addison Wesley.

Novak, J. D. (1998). *Learning, creating, and using knowledge*. Mahwah, NJ: Lawrence Erlbaum Associates.

Piaget, J. (1954). *The construction of reality in the child*. New York: Basic Books.

Popper, K. (2003). *The logic of scientific discovery.* Taylor & Francis Books Ltd.

Rajlich, V. (2002). *Program comprehension as a learning process.* Paper presented at the First IEEE International Conference on Cognitive Informatics, Calgary, Alberta.

Rajlich, V., & Bennett, K. H. (2000). A staged model for the software lifecycle. *Computer, 33*(7), 66-71.

Rajlich, V., & Xu, S. (2003). *Analogy of incremental program development and constructivist learning.* Paper presented at the Second IEEE International Conference on Cognitive Informatics, London, UK.

Ran, A., & Kuusela, J. (1996). *Design decision trees.* Paper presented at the Eighth International Workshop on Software Specification and Design, Paderborn, Germany.

Robillard, P. N. (1999). The role of knowledge in software development. *Communications of the ACM, 42*(1), 87-92.

Rostkowycz, A. J., Rajlich, V., & Marcus, A. (2004). *Case study on the long-term effects of software redocumentation.* Paper presented at the 20th IEEE International Conference on Software Maintenance, Chicago, IL.

Rugaber, S., Ornburn, S. B., & LeBlanc, R. J. (1990). Recognizing design decisions in programs. *IEEE Software, 7*(1), 46-54.

von Glasersfeld, E. (1995). *Radical constructivism*. London: The Falmer Press.

Vygotsky, L. S. (1978). *Mind in society.* Cambridge, MA: Harvard University Press.

Wang, Y. (2002, August), On Cognitive Informatics, Keynote Speech. In *Proceedings of First IEEE International Conference on Cognitive Informatics (ICCI'02)*, (pp.34-42). Calgary, AB., Canada. IEEE CS Press.

Wang, Y. (2004, August). On cognitive informatics foundations of software engineering. *In Proceedings 3rd IEEE International Conference on Cognitive Informatics (ICCI'04)* (pp. 22-31). Canada: IEEE CS Press.

Wang, Y., & Kinsner, W. (2006). Recent advances in cognitive informatics. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 121-123.

Williams, L., Kessler, R., Cuningham, W., & Jeffries, R. (2000). Strengthening the case for pair-programming. *IEEE Software, 17*(4), 19-25.

Wunderling, R., & Zцckler, M. (n.d.). *Docpp.* Retrieved from http://www.zib.de/visual/software/doc++/

Xu, S., & Rajlich, V. (2004). *Cognitive process during program debugging.* Paper presented at the Third IEEE International Conference on Cognitive Informatics, Victoria, BC.

Xu, S., & Rajlich, V. (2005). *Dialog-based protocol: An empirical research method for cognitive activity in software engineering.* Paper presented at the Fourth ACN/IEEE International Symposium on Empirical Software Engineering, Noosa Heads, Queensland.

Ye, Y. (2006). *Supporting software development as knowledge-intensive and collaborative activity.* Paper presented at the 2006 International Workshop on Interdisciplinary Software Engineering Research, Shanghai, China.

This work was previously published in the International Journal of Cognitive Informatics and Natural Intelligence, edited by Y. Wang, Volume 1, Issue 3, pp. 78-101, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.22
Designing for Service-Oriented Computing

Bill Vassiliadis
Hellenic Open University, Greece

EXECUTIVE SUMMARY

This case describes major management and technology issues which arise when designing advanced service-oriented architectures in distributed networked environments used for e-learning. The organization at hand uses a mixed funding model and is preparing for a large expansion of its services and capacity. This case takes place after the initiation of a project and during the design phase where significant decisions have to be taken about what is feasible, what are the risks and how they can be dealt with and finally, what is to be developed and how. The need to address diverse goals set by business, technology and education right from the start of the project requires new methodologies for documenting development plans, feasibility studies, risk, and human resource management policies. The project manager needs to go beyond traditional project management methods in order to cope with the needs of this use case and most importantly, to manage the risk that arises from many directions.

ORGANISATIONAL BACKGROUND

This case is set in the higher education industry involving an Open University which utilizes a mixed business model: its operation is funded by both public and private funds. This institution is supporting a diverse population of students which undertake undergraduate or postgraduate studies. Moreover, it provides postgraduate curricula to graduates who wish to extend or upgrade their studies to subjects related to their profession. The University's curricula correspond to various certificates, Bachelor or Master's degrees. A Bachelor degree is comprised of several research directions. Courses, organized in modules, are designed according to the distance learning methodology: students study using text books, participate in 5 tutorials for each module taking place in 8 towns, communicate with the corresponding tutor by telephone, fax, e-mail and letters, prepare 4 – 6 assignments for each module and finally, take a final examination 10 months later. A student belongs to one student group, called class. A class

is based on a major city in which class sessions take place. A tutor is allocated for each class of a maximum capacity of 32 students who inhabit in a specific geographical region.

The organization's educational services are targeted to a very specific audience:

- students: students are the main clients of the organization. They pay fees for attending courses.
- other Academic Institutions collaborating with the University: academic institutions are possible collaborators in the development and provision of MSc Curricula.

The academic personnel of the institution involve a small number of about 30 permanent personnel (Professors, Associated Professors and Assistant Professors) as well as a large number of tutors (about 1000). Each one of the permanent personnel undertakes, besides tutoring, the coordination of all classes and the overall academic responsibility for a specific course. Tutors cooperate with the University on an annual basis.

This organization is supported by a mixed funded scheme: it allows admission of students without an entry examination but although it is a state University, students pay fees for the cost of their studies. Fees cover the cost of the instructive material and all the expenses related to the studies. The number of annual registrations in each bachelor degree programme is about 500. To date, over 1800 students are registered for various stages of each course. Future plans of the administration foresee a significant increase in the number of new registrations will reach up to 1500 per year. The total number of students attending at all the educational programmes totals to 16000. Table 1 summarizes these figures.

The institution is relative new, with only five years of existence. It monopolizes the open and distance education of the country since it is the first and only higher education institute of its kind with bachelor and MSc. degrees recognized by the state. This fact has given the administration of the University the possibility to expand both its range of services and its capacity without serious competition in the domestic market. The following graph depicts in approximation, the annual increase of the organization's budget for the last 5 years.

It is worth noting that the unique services offered has created a large and ever-increasing client base. As the University is annually increasing its capacity, more and more applications are made for filling new student positions. Figure 2 presents this annual increase in capacity and the corresponding applications for registration for the last five years.

Since the organization is an Open University, new technologies are used for delivering education: electronic material, video lectures, electronic forums and e-mail. Advanced services include high-end communication and collaboration tools for online delivery of content and lectures in real time. It is reported that these tools have several drawbacks and they will be probably replaced in the near future. Asynchronous services are

Table 1. Organization's staff and clients

Administrative Staff	Permanent Academic Staff	Non-permanent Academic Staff	Students (clients)
100	30	1000	16000+

Figure 1. Annual budget increase

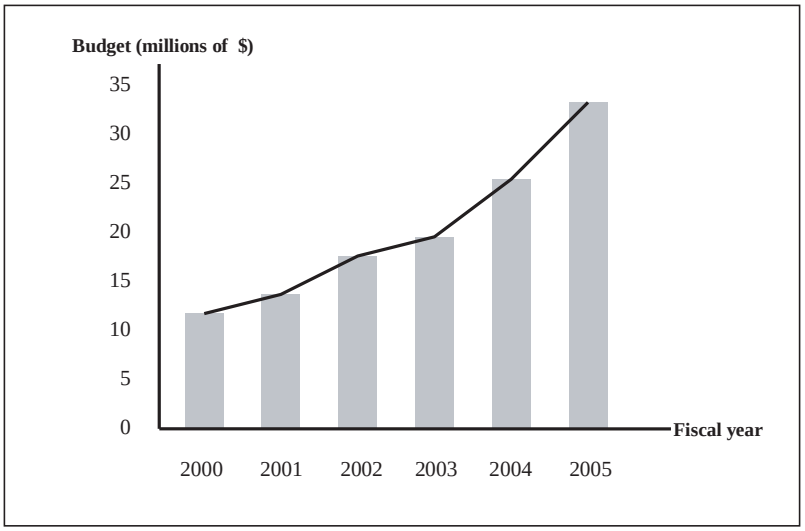
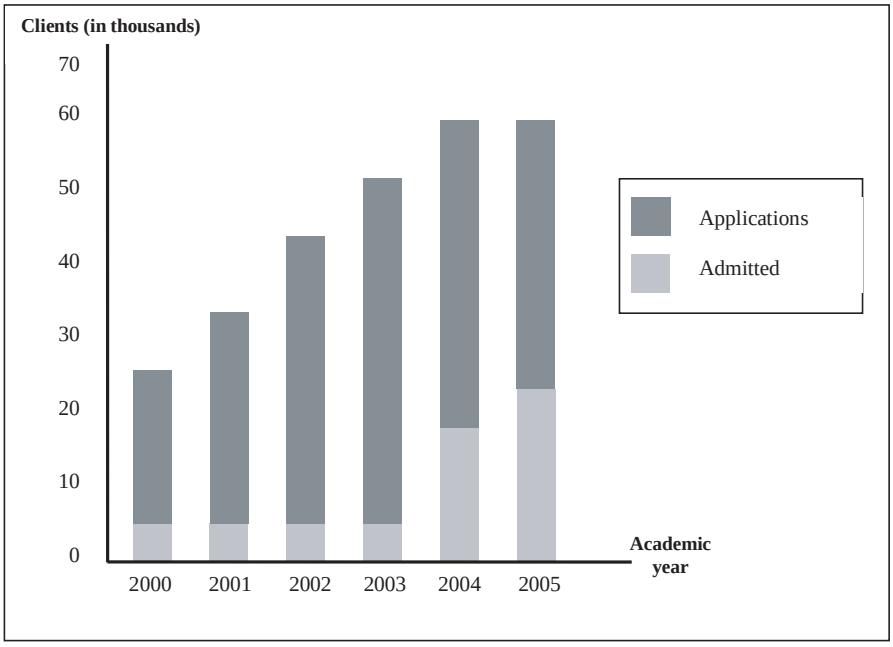


Figure 2. Applications vs. admission of students (clients) for the last five years



provided through Web sites, forums and e-mail. Frequent meetings between tutors and students take place for presentations, discussing problems or for providing additional information. Since this University covers the whole country, a small number of centres are used in major cities providing support such as lecture rooms and in the near future, video-conferencing facilities. It must be noted that a firm policy of the organization is to keep the tutor/student ratio steadily at 1:30, regardless of the increase in user capacity. This allows the continuing monitoring of student's performance and the establishment of tight relations between the learners and the academic staff.

The organization's culture is characterized by a spirit of innovation both technologically and administratively. Although some initial organizational problems prohibited rapid expansion in the first 4 years of its operation, rapid growth has taken place in the beginning of the fifth year while ambitious plans have been set for the next 5 years: a 50% expansion with regard to current numbers.

The population of students, the actual users of the organization's services, is extremely diverse. A typical student is rather of a mature age, part time student. Many students are already professionals and have different cultural backgrounds and career goals. They are highly geographically dispersed all around the country.

SETTING THE STAGE

In order to support the rapid expansion within the next five years, the organization has decided to use new technologies as a spearhead for increasing quality of service and reducing organizational complexity. The main effort was focused on the main product: e-learning service provision. The effort of upgrading services was of a high risk because the client population was very sensitive to changes in the way learning was delivered. On the other hand, repeated evaluations of current procedures had shown several deficiencies in service delivery and most of all inability to support the ever-increasing client population. In short, services were suffering from the following drawbacks:

- low participation. The participation of students to e-learning sessions and the use of collaboration tools were extremely low. The main reason was that users were not motivated enough. Although basic functionality such as video-conferencing and collaborative support were provided, procedures and tools seemed to lack the interactivity and the efficiency needed for broad acceptance.
- interactivity: The lack of interactivity in current services was largely reducing the interest of users.
- efficiency. Since students had significant time constraints (most of them were employed), services should have been tailored as mush as possible. This means that an upgraded version of the system should provide the means to cut down the costs of learning through adaptation, along with effective task assignment, execution duration control and monitoring.
- cost reduction. The inclusion of new tools or services to the existing system was deemed to be a too costly process. Costs increase when management of the integration of internal and external resources is needed: aspects such as security, heterogeneity and copyright were not dealt with by current approaches.
- ever-increasing number of clients. The number of clients (actually students) that need to be supported by distance learning applications was quite large. Their number is estimated to increase by 50% within the next five years. Normally, this would not be a problem if static Web applications were used for delivering content. A simple Web server would be adequate. Nevertheless, in

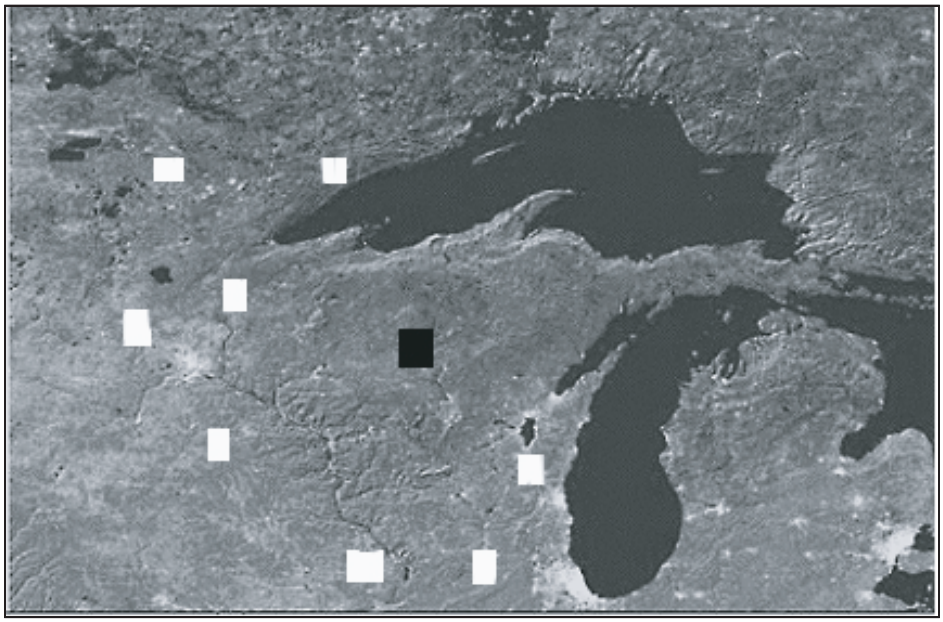
the case of more complex services, there was a problem on how to cope with tens or even hundreds of processes running at the same time by remote users. Furthermore, according to the organisation's rules, there could be no strict rules on when students may access a service or not. Access to services is quite random in terms of number of users, time of access and duration of use. The University's policy to improve the quality and capacity of the e-learning services (number of user supported), required a solution that could handle possibly thousands of users at the same time.

Existing tools used were quite cost-effective to support, relying mainly on Web-based technology. They included:

- asynchronous collaboration tool, which was being used for simple collaboration/ sharing of knowledge and resources and virtual class management.
- Web sites and forums, used for asynchronous delivery of educational material and for off-line collaboration.
- tools and Web browsers were used to access and manipulate information objects:
  - Internet Explorer/Mozilla for Web browsing,
  - Word for creating documents,
  - PowerPoint for creating, organizing and illustrating presentations,
  - WordPad for plain text editing.

Besides the obvious drawbacks of using such simple technology for supporting an extremely

Figure 3. Example of a distributed business model: headquarters (black square) and satellite centers (white squares) of the organization are geographically dispersed



demanding and diverse client population, another disadvantage relating to the philosophy of the organisation was soon discovered: its business model was distributed and current technological solutions were based on a client-server model. The University has its headquarters in a major city of the country while clients are supported by 8 satellite centers based on 8 other cities (Figure 3). This network was already supported by a high speed network backbone guarantying fast delivery of heavy loads. Although electronic services were geographically independent, other services were provided on-site. The latter included mainly facilities for regular face to face sessions, a significant aspect of the University's educational policy. Thus, several of the provided services were geographically-sensitive, but the supporting technology was not.

Such a business model was recognized as an advantage but only in the case were distributed computing would be used as its enabler. Administration as well as user services should be supported by a networked model mapped to the nodes of the existing, underlying business model. Classic practices used so far were minimizing ROI (Return on Investment) and kept quality and user satisfaction at low levels.

The above-mentioned situations required a mixed economic/technological/pedagogical solution that would not address all problems at once, but rather deal with the most important ones. This decision was made after considering several similar cases of organisations that tried to introduce state of the art technologies, although not all of them were based on a high risk/high gain model. Most such efforts involved small scale, single institute adoption of state of the art tools from which the organization's experts have drawn some useful conclusions (Haywood, Anderson, Coyle, Day, Haywood & Macleod, 2000; Jefferies, Thornton, Alltree & Jones, 2004; Saunders & Klemming, 2003). Cross-institution (Van Weert & Pilot, 2003) or nation-wide (Demb, Erickson & Hawkins-Wilding, 2004) efforts were also considered since one of the University's future and most bold goals was to become a nationwide service provider of e-learning services to the commercial sector also.

Before any decision was taken about how to proceed, feelings about changes were mixed. On one hand, most thought that using advanced IT for facilitating better service provision was a must but the economy and benefits of such a solution were under question. The benefits of introducing advanced e-learning services, even in an organization that already uses IT as a backbone, still remains in theory. Services that are considered the most promising are also the most costly: personalization, real-time communication and other advanced functionalities lead to significant costs. So the business problem faced in this situation was focused on how to improve e-learning services by keeping costs to a minimum and maximize ROI in such a diverse and difficult sector as the educational one (Bender, 2003).

CASE DESCRIPTION: GOALS

Before undertaking any large effort to upgrade the current system, it was decided that it would be wiser to test some high risk/high gain technological/educational solutions to a specific curriculum. Three main directions where considered as most important for project execution: economy, pedagogy and technology. Funding was provided by the participation of the University to a large international project with very ambitious goals: to design and develop new educational services using distributed technologies. Although this entailed a high risk, possible benefits were also high. Furthermore, the fact that funding was coming from external sources, did reduced somewhat the economic risk of the University itself.

The pedagogical goal of the project was to improve learning through real time cooperation/

working among individuals, sharing and reuse of information in highly distributed and dynamic environments. The pedagogical goals outlined had highly demanding technical requirements, many of which were also the concerns of distributed systems research (Duffy & Jonassen, 1992). Group working implies shared interactive resources, necessitating both concurrency control and awareness of others activities. Active learning requires interactive resources, many of which will only be engaging if they are suitably responsive – a quality of service (QoS) issue that depends on many components of a distributed system – the low-level infrastructure (hardware, OS, network), the middleware and the interface software. Concurrency control and interactive responsiveness can make conflicting demands on a system. Real world input, such as live student interaction makes a network connection mandatory, and this again raises QoS issues such as fault detection, masking and tolerance for the learning environment. Accessibility, as in anytime/anywhere, requires availability, which may be supported through replication of resources, but this creates further tensions with responsiveness and concurrency control due to the need to maintain state across replicas. Accessibility also means adapting to available capabilities. For example: can the same learning environment be delivered through low-bandwidth mobile devices and high-bandwidth multimedia workstations? Accessibility also means supporting special needs of the individual, such as disabilities. More generally, the individual user should be recognized and catered for, and this personalization requires semantic tagging and profiling that can be difficult to formulate, both conceptually and in terms of machine representation. Standards efforts have been particularly slow in addressing this problem. Finally, contextualization required a move from the traditional view of an online learning environment as a stable long-lived entity (e.g., during the lifetime of a teaching module) – to another where the environment may evolve and change much more frequently, perhaps even several times a year – a dynamicity that is alien to current e-learning products.

The overall goal of the project was set to be the improvement of the efficiency of current e-learning practices by promoting a collaborative/social learning model based on services rather on tools. Furthermore, to use the new services as reinforcements to the foundations laid by the linear structured text books and the online and off-line lectures. As mentioned in a previous section, the goal was not to solve all identified problems, but rather to integrate a new learning methodology that would work as a supplement to existing practices. The aim was to increase:

- student motivation to participate in online sessions,
- student confidence and satisfaction,
- efficiency of the learning process.

In order to reach the main goal, the following subgoals had to be accomplished:

Sub-Goal 1

- provide added value services in the form of highly interactive multi-cooperation sessions and advanced virtual communities support.
- provide advanced, media rich services in the form of multi-step, cooperative working sessions for the a specific course.
- facilitate social learning and collaboration through knowledge sharing and reuse between groups and individuals.

Sub-Goal 2

- support thousands of users while preserving adequate Quality of Service.

- provide the services to a diverse and very large student population.
- provide a single access point with a homogeneous interface.
- preserve quality of service (fault tolerance, response time) at any time.

Sub-Goal 3

- evaluate the didactical approach and the technology infrastructure by contacting a real, large scale experiment in the selected course.
- provide/configure metrics for measuring performance in terms of scalability, quality of service and security.
- test the infrastructure in a real situation.
- assess the results and provide feedback.

Supporting computationally expensive services for thousands of users with minimum costs and maximized ROI was the biggest challenge. A novel, and thus high risk technology was selected for delivering the services: grid computing.

GRID COMPUTING: STATE OF THE ART

Grid computing is a new paradigm that enables the virtualization, management and provision of heterogeneous and geographically dispersed computing resources (Berman, Fox & Hey, 2003). A resource may be CPU power, storage, information/knowledge or even services. The vision of the grid is to provide computing resources in the form of commodities just like as the electric grid provides electricity to consumers. Providing computing as a commodity means that the end-user just plugs into a provision and gets all the resources needed under payment, transparently and with adequate quality of service. Transparency is essential because users should not be worried about how to find and manage computing resources, they just consume them. Quality of service involves not only high availability but increased security as well.

In order for this vision to become a reality, the distributed computing research community defined a set of functions that a grid should perform: resource scheduling, data virtualization, provisioning, and resource management (Li & Baker, 2005). The enabling software of the grid, actually in the form of middleware, should be able to operate on a variety of operating systems and network configurations.

In the past ten years, the grid has matured from a purely scientific instrument that processed vast amount of data to a collaboration enabler (De Roure, Jennings & Shadbol, 2005). Besides standard distributed computing technologies (virtualization, grouping of resources), the

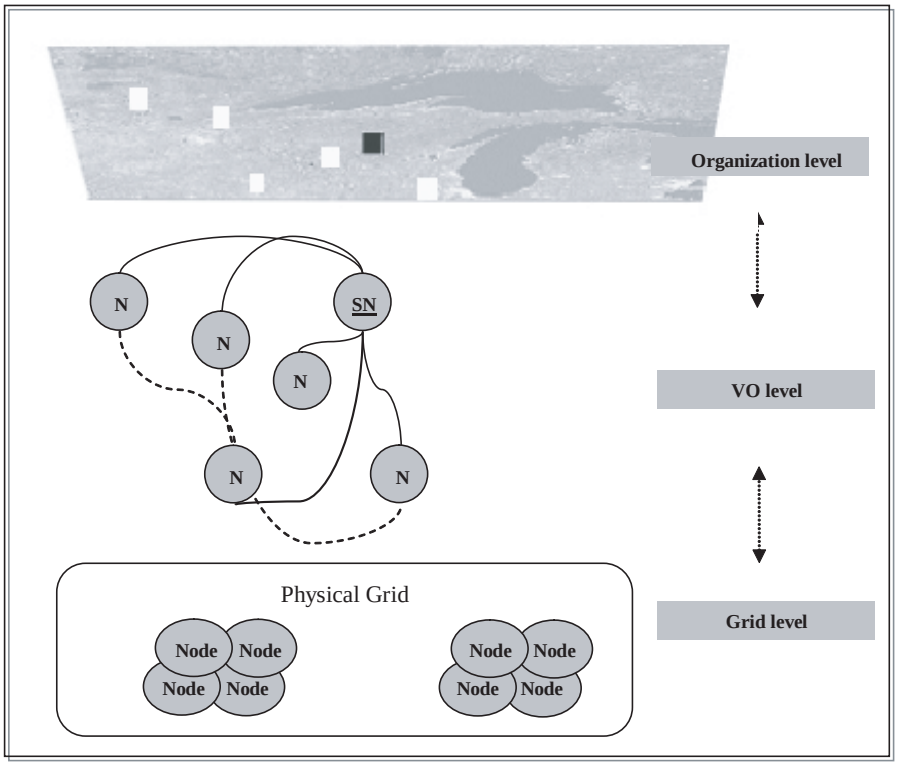
Table 2. Grid characteristics/functionalities vs. other technologies

Characteristic / Functionality	**Other Technologies**	**grid**
Transparency	*WWW*: simple collaboration	Enables advanced collaboration
Sharing	*P2P*: files	resources
Virtualization	*Virtualization tech*: single system	multiple systems
Grouping of computational resources	*Cluster computing*: homogeneous systems with geographical proximity	heterogeneous and geographically dispersed systems

grid now incorporates semantics (De Roure et al., 2005), Peer to Peer (Talia & Trunfio, 2003) and Web Services' (Cannataro & Talia, 2004) characteristics/functionalities. The use of these functionalities in the grid however is slightly different as described in Table 2.

The grid was originally applied in e-science but the framework it engendered to realize effective sharing of distributed heterogeneous resources is now being applied to many other areas, especially enterprise computing, e-commerce and recently higher education (Maldonado, 2004). The first application is extremely interesting since the grid has been proposed as a Virtual Organization (VO) enabler (Foster, Kesselman & Tuecke, 2001). In this model, grid technology is used for coordination, sharing and reuse of services, knowledge and data across the geographically dispersed nodes of the VO. VO nodes are mapped onto the physical grid nodes (computers or clusters of computers) and predefined workflows are used for service execution and data management (Figure 4). The next generation of grid solutions will increasingly adopt the service-oriented model for exploiting commodity technologies. Its goal is to enable as well as facilitate the transformation of Information into Knowledge, by humans as well as – progressively – by software agents, providing the electronic underpinning for a global society in business, government, research, science, educa-

Figure 4. An organization's central and regional centers mapped to a central (SN) and secondary (N) VO nodes, mapped to a grid



tion and entertainment. The most important benefit of this technology is its ability to meet major bursts of CPU demand using existing resources. By decoupling the services from the underlying h/w, resources are automatically allocated/deallocated as loads are increased/decreased. This increases ROI on h/w since idle computers are used instead of buying new ones.

TECHNOLOGY SELECTION

The grid computing paradigm is relatively new although it uses ideas and technologies from other, more mature disciplines. It has been recognized as a high risk technology that may provide significant benefits in special kinds of applications and especially in distributed coordination/cooperation. Although its potential has been recognized by the research and commercial community, the lack of grid-specific standards has been viewed as a serious uncertainty factor, prohibiting currently its wider adoption. Thus, projects that use grid technology bear a significant technological risk but also have a lot of potential.

The use of grid technology was chosen because the services anticipated were computationally expensive since a large number of users would be engaged in processes including massive calculations (e.g., online experiments). Furthermore, the grid had already shown significant advantages in collaboration and knowledge sharing for large teams. The organization's policy to adopt new technologies in order to minimize costs and maximize return on investment in all levels played a significant role in taking this decision. Improvements were anticipated in many levels, from administration to the delivery of education. In the latter case, new e-learning services were supposed to be designed in a way that:

- fully utilizes existing resources,
- could be expanded in order to support a large number of users while minimizing costs,
- would be flexible and able to respond to the rapidly evolving market,
- increases the power of core resources without creating the need for investing in additional hardware or hosting infrastructures,
- supports the University's dynamic policy for expansion.
- In this context, grid technology would play a significant role towards the direction of a compact, scalable and cost-efficient infrastructure for e-learning services. The use of grid technology was also necessary in order to fulfil real business needs:
- to combine resources on-demand for service provision,
- to intelligently allocate finite resources to the appropriate applications,
- to free expensive supercomputing resources for only the most high-end processing needs while still supporting the wide range of compute-intensive applications run by research groups throughout the University,
- to increase ROI on existing resources,
- to be able to support the ever-increasing number of users,
- to provide cross-domain functionality.

A grid infrastructure would permit transparent access to services and resources wherever they are located. Furthermore, special grid functions valuable for the efficient provision of services such as resource reservation, storage of large volumes of data and minimization of data transferred by using a super-node grid structure would be made available. In conclusion, the business value of using grid technology for this project stemmed from the possibility to leverage existing h/w and s/w investments and resources, reducing operational costs and creating a scalable and flexible infrastructure. It has been argued by some of the technical staff that acquiring CPU power and increasing storage capacity by using clusters of computers was both a cheap and efficient solution for supporting large numbers of users. This

was partially true, but the real problem was not purely technological (e.g., about CPU power) but business as well. If contemporary technology would be used then cost would be increased since the management and monitoring of different resources, would be difficult and expensive. Furthermore, expanding usually means buying and dedicating new hardware and software to new services, possible fragmented resources while ROI would not be maximized. It was also recognized that return of value would stem from educational, organizational and external benefits. Educational benefits would include more efficient learning models overcoming limitations problems of current computing infrastructure, provision of ubiquitous e-learning services. Exploiting potentials from the provision of consulting services to other academic institutions in the country concerning new e-learning models, and grid-enabled learning were also identified. Organizational benefits were supposed to include improvements in organizational efficiency (better information flow, savings in staff time, and improvements in service provision), a possibly, enhanced public profile of the institution and high return on investment: support more students, with less effort and reduced costs. Other secondary factors such as strategic partnering with external organizations (e.g., other higher education institutions, commercial, or community organizations), and the opening of new markets by providing new services through the collaboration with other academic organizations were identified.

CASE DESCRIPTION: DESIGN AND INITIAL STEPS

Before the design phase of the project was initialized, a strategic decision was taken concerning the identification of the project's stakeholders: the actors involved and/or affected by its implementation. It was decided to include:

- Students: students of the University. They indirectly provide the requirements for designing a new learning model. This means that their requirements should be assessed though their behaviour throughout the learning process.
- Teaching Staff: academic personnel including professors and tutors. They would provide input on the design and adoption of new learning models and evaluate the project's efficiency.
- Technical and support staff: including network and server administrators, special technical staff for supporting e-learning service provision.
- Laboratories: laboratories include laboratory staff, equipment, e-learning modules, services, knowledge.
- Academic Institutions collaborating with the University.
- Government: a potentially successful deployment of grid service architecture in the University would boost the adoption of new learning models and grid services in other higher education institutions. This would be made possible through the cooperation of the local Ministry of Education. Experts of the Ministry would evaluate the new methods and decide if the return of investment for its adoption in a wider scale would be feasible.

The actual role of the University was to produce specific requirements, test and evaluate solutions developed by the other partners. A proactive approach was adopted for the management of risk in the larger project. Project risks, especially the ones related to technological innovation, but also those related to budget, schedule and coordination were considerable but the Consortium was comprised of selected partners with strong research and development experience. The University's subproject purpose, referred in this communica-

Figure 5. The context of the case study

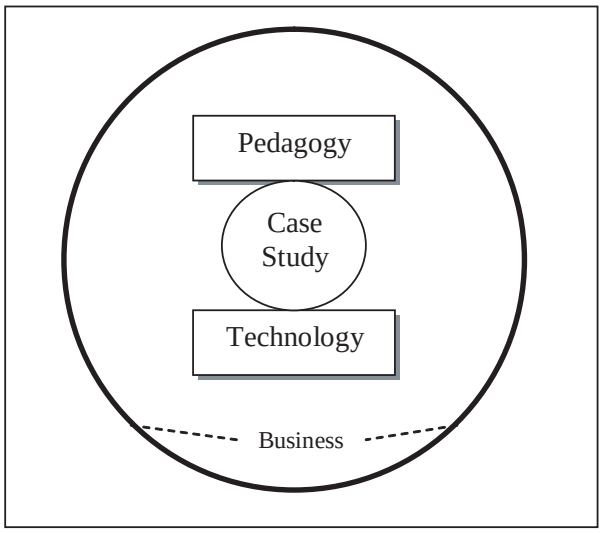
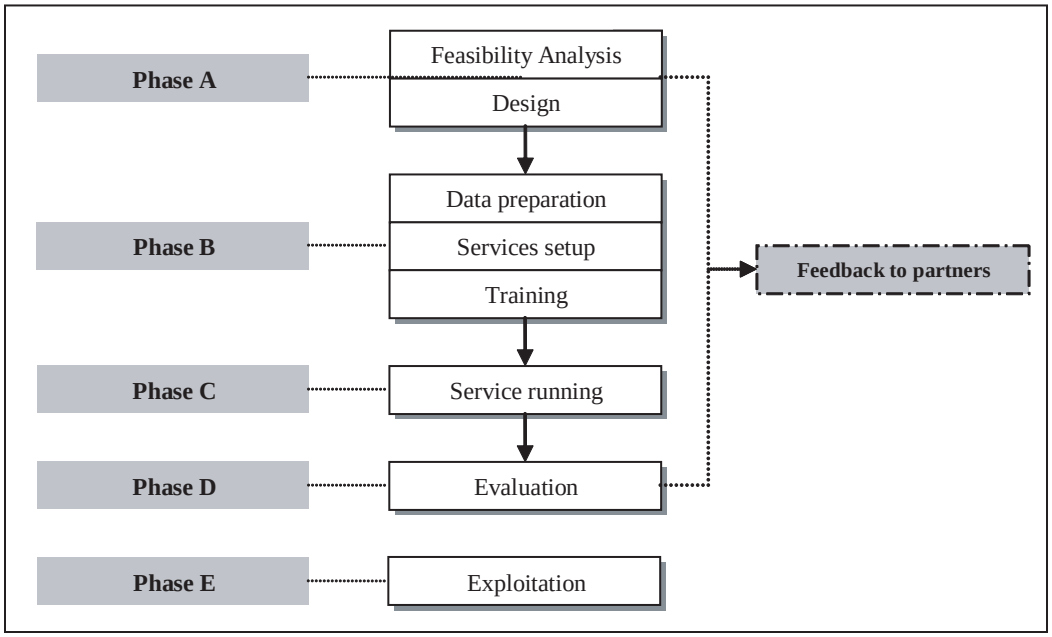


Figure 6. General approach to project completion



tion as *case study,* was to develop and gain insight into the processes involved from formulating pedagogic requirements to the implementing environments that meet these requirements. The case study and the University's goals were converging, since they were both aimed at a mixed pedagogic/economic/technological solution for advanced service provision (Figure 5).

The case study was anticipated to include five main phases, each of them decomposed in smaller tasks: design, development, deployment, evaluation, and exploitation (Figure 6). The first phase would check the feasibility of reaching the goals with the available funds and technology and define detailed requirements. During the development, phase data preparation, services setup and configuration would take place. Furthermore, tasks such as introduction, motivation and training of staff for operating the service were anticipated. Case study enactment would involve the actual running of the new services in real conditions for a short period of time. These services would be then evaluated by different user groups and conclusions would be communicated to the consortium.

FEASIBILITY ANALYSIS

Although the project is its initial phase, the project team is able to pinpoint the milestones for the upcoming phases. Phases are not necessarily sequential; two phases may be executed in parallel at a given moment. The first phase is relatively easy to decompose into smaller phases and set exact milestones, while for the others it is harder since one phase depends on the results of the previous ones. Assumptions however are possible (and useful) to make (Table 3).

As described previously, in the first phase requirements are defined. A feasibility study that takes into account technological, pedagogical, organisational and economical considerations would be of great help to define concrete and

Table 3. Milestones per project phase

Phase	Milestones
A: Design	• Feasibility study completed • Requirements are identified • Resources (human/financial) needed are committed • Initial plan for the first year has been agreed upon
B: Development	• Services are designed according to user requirements • The architecture of the infrastructure (e.g., exact technologies used) is defined • First prototype is made available • Second prototype is made available
C: Deployment	• Software (first prototype) is installed • Initial feedback is given to the developer team
D: Evaluation	• Evaluation methodology and plan has been agreed upon • Evaluation group formed • Feedback is provided to the developer team
E: Exploitation	• Demonstrators are made available • Licences (e.g., patents) are acquired

realistic objectives (Bates, 2000; Boucher, 1998). Subsequently, a detailed initial plan for the first period (e.g., the first year) can be drafted. Since this plan is based on the feasibility study, it is (theoretically) based on solid ground and it is as close to reality as possible. This study should consider the economic viability of the project. As mentioned previously, major cost savings of ICT introduction still remains in theory while it seems that its greatest pedagogical advantages are the most costly. Other, usually overlooked, costs may include courseware development costs, incremental capital and recurrent equipment costs, costs associated with provision of appropriate resources, infrastructure costs, maintenance, user support costs, costs of adoption, access costs, security costs, replacement costs and institutional overheads. This has lead Rumble (1999) to suggest that the cost of utilising advanced ICT services is nearly the same with face to face teaching. This assumption holds for complete distance learning solutions where traditional methods are completely replaced by ICT but in this blended learning situation it remains to be seen if this is the case.

The project team should be extremely careful in determining the resources needed and technologies used. Since grid technologies and Service-Orientation are relatively new fields, changes in frameworks or standards may have a significant impact on the project. For this reason, a technology-watch team should be formed in order to monitor changes and inform the partners.

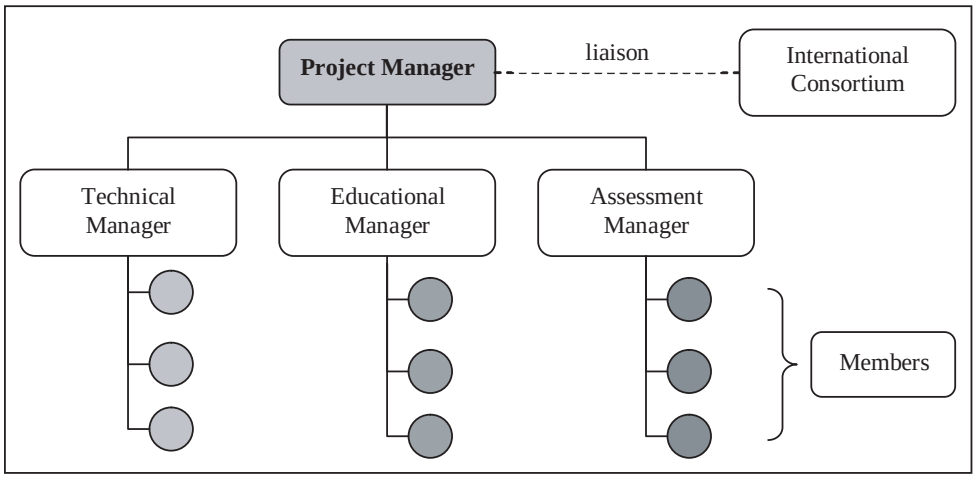
The milestones of the subsequent phases (B-E) can only be generally defined. They include the design and development of services and infrastructure, the availability of prototypes and demonstrators and finally, the evaluation results (performed by a specially selected user group).

Management Structure

A local project team was put together coordinated by an experienced project leader with background in both technology and pedagogy. Achieving a consensus between these two disciplines was important since the failure of addressing the requirements of one was automatically transformed to the failure of the other. The project leader was responsible for managing three sub-teams: the technological, the educational, and the evaluation team. Each of these teams would be managed by a corresponding leader and would have a relatively small number of members (3-4). As depicted in the team chart (Figure 7), there are many different roles that cover managerial, assessment, technological and educational criteria. Many of the actors involved, would work in the project for a specific, short period of time engaged in special duties (e.g., assessment).

The *Project Manager* would be responsible for the overall management of the case study. The project manager would coordinate the staff working in the project and would be responsible for the completion of individual tasks within the appropriate timeline. Project manager's responsibilities also included the final quality check of the work produced by the team. The *Technical Manager* would be responsible for the technical development of the project. This included the provision of technical requirements and guidance to experts during the installation, enactment and evaluation of the services. The *Educational Manager* had the responsibility to put together the requirements analysis, design, and integration of new e-learning models. Furthermore to design and support a concrete e-learning use case that would have an actual educational value for the University. Finally, the *Assessment Manager* was responsible for assessing the services during the installation and enactment phase. For this purpose, three tutors would be cooperating along with a number of students for assessing the final output of the project. Since assessment was going to take place at the final stages of the project, the names of the above mentioned actors were not made available from the beginning. The project team should cooperate smoothly with the much larger

Figure 7. The structure of the local project team



international team responsible for designing and producing the solutions.

The project manager and the members of the local project team interact with other members of the international consortium. This consortium is also organised so as to cope with a range of legal, contractual, ethical, financial and administrative tasks that may occur during the project life-cycle. The management structure of the consortium is divided into four bodies: project management (project manager, Project Coordination Board, Executive Board and Scientific Advisory Board), financial management (financial manager), Operations/Technical management and Quality management. Each of these bodies is comprised of separate roles and/or special boards with distinct authorities. These authorities are defined in a consortium agreement which is agreed upon and signed by each participant organisation before the start of the project. There are a lot of different configurations in these kinds of managerial structures; structure flexibility and the competence of the leading persons are the keys to successful management.

CURRENT CHALLENGES FACING THE ORGANIZATION

The project manager has a draft of a general implementation plan which defines major deliverables and deadlines. The project manager knows that this plan must be detailed and refined in the near future. For the time being, he is concentrated in the first phase of the project. One of the most important phases of project elicitation is feasibility and requirements analysis. The feasibility study to be contacted for the case study has the purpose of instigating cross-project understanding of requirements engineering, identifying and agreeing on common representations and modelling techniques for requirements engineering. The value of this analysis is very important since the adoption of a new technology like the grid in

such a difficult application domain as distance learning, is of high risk. The existence of a gap between technologists, educationists and final users is natural to exist but it should not exceed a certain threshold; otherwise the final solutions would fail to perform successfully in real situations (Xenos, Pierrakeas & Pintelas, 2002).

The University's project team needs to initiate an intense specification process to clarify, on the one hand, for the user groups the specific options in a given learning setting and on the other, to enable technologists to efficient communicate their views about emerging features and services useful for learning. This analysis process is demanding in time and resources since there are different views and a broad spectrum of ambitions and contexts inside the consortium.

The basic idea for the requirement analysis is to analyze the case study with a well experienced tool that provides sophisticated semantics for describing Service-Oriented architectures (SOA). The project leader needs to choose a general framework as a starting point for the requirements specification that also takes into account economic, technological and educational parameters. Furthermore, it should be oriented to the application of innovative technological solutions to traditional contexts.

A challenge that needs to be addressed in the first phase of the project is the methodology that should be followed for putting together the feasibility analysis plan. From his experience, there is no formal template or methodology for SOA cases. Most methodologies are either too technical or business/product oriented. But this project will produce services not products and furthermore, these services will be developed by a third party using an innovative, high risk technology. Services are different from tools; they have to be composed on the fly depending on needs (Rust & Kannan, 2003). They promote the so called "user-centered" approach for the design and development of software. Software as a service (SaS) has emerged as a response to a fundamental shift in enterprise business culture that started at the late 1990s (Foster & Tuecke, 2005). Traditional monolithic architectures are giving way to Service-Oriented Computing, which permits the utilization of large systems comprised of self-containing building blocks: services. Services may be made public, searched, reused, and combined to form complex business processes while in the same time retaining a significant level of flexibility. Since services are a relatively new concept, then the design of a SOA is in itself a challenge (Singh & Huhns, 2005). Again are the questions of what method is appropriate for system design and will classic approaches do the job?

The context also complicates things. Distance learning does not fall into any of the classic product categories, maybe it resembles e-commerce in some aspects, but there are differences: an online user does not behave as an e-learner does (De Bra, Brusilovsky, & Houben, 1999). How is it possible to identify and take into account business, technology and pedagogy requirements at the same time?

Anyway, before deciding on the methodology, the project team made some assumptions about the project, hoping that they would be the correct ones. First, that the design envisaged for the project could be developed in time and it would work within the context of the mission and goals that were set by the consortium and that the characteristics of the envisioned system would be among its high priorities. Secondly, that the organization's educational policy and legislative requirements would not prevent or circumvent key aspects of the system and those tools and technologies used so far can be integrated in the new infrastructure. Other assumptions should also be made on conceptual, legislative, stakeholder-related, and technological matters.

The administration of the institution also puts pressure on the project manager to consider the economic goals of the project, which closely relate to exploitation potential. A plan is needed to identify exploitation results and opportunities

during and after the completion of the project. Furthermore, the project manager needs to consider the quality aspects of both the procedure and the deliverables. The project manager is responsible for producing an output that is efficient and workable. In cooperation with the assessment manager, an expert in software quality and assessment, he is planning on putting together an initial *evaluation plan* to assess both methodology and services. This plan will be used as a guide to insure that quality is maintained at high levels. Priorities for evaluation must also be divided into categories and related to goals. Several indicators must be identified that can supply a measure of the success of the project in achieving the targets set in each of the above mentioned categories.

Expansion, a strategic business objective of the institution is one of the drivers of the project. It would seem thus, that this IT-focused project strengthens the strategic alignment between business and IT: the alignment of business and IT strategy in order for the organisation to stay competitive and gain profit from the use of advanced technology (Boar, 1994). In this case, the added value expected is (1) to provide more quality services to existing clients and (2) the ability to serve more clients efficiently. So the benefits are only connected to client services and not directly to organisational structure, internal infrastructure or processes; these will probably remain stable due to the academic nature of the organisation. One way to ensure strategic alignment is through assessments that is, setting performance goals for the processes affected. This should be dealt with both in the feasibility analysis of the design phase (phase A) and in the evaluation phase; software developed should be evaluated in the light of its impact on business strategy. For this reason cross-discipline metrics need to be defined during the design phase, metrics that measure the impact and relationships between IT and business strategy. These metrics should be evaluated at specific milestones and, if needed, corrective actions should be taken.

A set of assessment criteria must also be clearly identified for several categories of e-learning provision that the deployment of new services will affect. These criteria should then be de-composed, where possible, into smaller metrics that can be quantifiable — that is, a number can be assigned to them. All this "atomic" measurements, combined together using mathematical equations, will finally provide some kind of performance indicator. These criteria should try to establish whether the services create new possibilities compared to present tools and methodology, support the transition from classic educational models to advanced, student-centric methods and finally if they are cost-effective. It is apparent that different complexities (or levels) for these criteria exist and they need to be identified.

The case study's project management should also be based on a concrete *project plan* that anticipates risks stemming from the collaboration with many partners in a multi-cultural, international environment. Specifications are greatly affected by the technological solutions at hand, so strong cooperation with other partners need to exist in order to exchange information on what or what cannot be done. A risk that has been identified by all partners is the possibility that a different visions exists for different institutions in the consortium. Although the goals of each partner participating in the project may be different in an atomic level, the general goals should be the same and most importantly, well understood. The project manager and its colleagues at the institute are a little bit concerned about this but at the beginning of the project there is little thinks they can do. But they have in mind to follow a formal methodology in order to anticipate future misunderstandings or risks relating to these differences in vision, anticipations and business/technology culture between partners.

The formation of the project team is also a challenge. Its basic structure should include experienced sub-team leaders with proven abilities to cope with high risk research and development.

Although the four main members of the team (the managers) will work in the project from the beginning to its completion, other members will participate only in certain phases of the project. The project manager is concerned about the dynamicity of its team so he needs to put together a flexible *human resource management plan*.

After a discussion with the assessment manager, the project manager also decides to anticipate risk by developing a *risk assessment and contingency plan* that will monitor all ongoing management activities. The initial version of this plan should identify only high level, major risks and propose measures to overcome them. The challenge here is to correctly describe the impact of each potential problem on the project. The relevant coordinators and managers will review and apply risk management strategies. Risks will be re-evaluated at the regular consortium meetings. The project manager's opinion is that, in general, the strategy for reducing the risks related to project management is to reduce their probability of occurrence, rather than their impact. This necessitates a proactive style of management, with strong emphasis on planning, control, and corrective action.

REFERENCES

Bates, A. W. (2000). *Managing technological change*. San Francisco: Jossey-Bass.

Bender, T. (2003). *Discussion-based online teaching to enhance student learning: Theory, practice and assessment*. Vancouver: Stylus Publishing.

Berman, F., Fox C. G., & Hey, A. (2003). *Grid computing: Making the global infrastructure a reality*. New York: Wiley Press.

Boar, B. H. (1994). *Practical steps for aligning information technology with business strategy*. New York: John Wiley & Sons.

Boucher, A. (1998). Information technology-based teaching and learning in higher education: A view of the economic issues. *Journal of Information Technology for Teacher Education, 7*(1), 87-111.

Cannataro, M., & Talia, D. (2004). Semantic and knowledge grids: Building the next-generation grid. *IEEE Intelligent Systems, 19*(1), 56-63.

De Bra, P., Brusilovsky, P., & Houben, G. J. (1999). Adaptive hypermedia: From systems to framework. *ACM Computing Surveys, 31*(4), Article No. 12.

Demb, A., Erickson, D., & Hawkins-Wilding, S. (2004). The laptop alternative: Student reactions and strategic implications. *Computers & Education, 43*(4), 383-401.

De Roure, D., Jennings, N. R., & Shadbolt, N. R. (2005). The semantic grid: Past, present, and future. *Proceedings of the IEEE, 93*(3), 669-681.

Duffy, T. M., & Jonassen, D. H. (1992). *Constructivism and the technology of instruction: A conversation*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications, 15*(3), 200-222.

Foster, I., & Tuecke, S. (2005). The different faces of IT as a service. *ACM Queue, 3*(6), 27-34.

Haywood, J., Anderson, C., Coyle, H., Day, K., Haywood, D., & Macleod, H. (2000). Learning technology in Scottish higher education – A survey of the views of senior managers, academic staff and experts. *ALT-J, 8*(2), 5-17.

Jefferies, A., Thornton, M., Alltree, J., & Jones, I. (2004). Introducing Web-based learning: An investigation into its impact on university lecturers and their pedagogy. *Journal of Information Technology Impact, 4*(2), 91-98.

Li, M., & Baker, M. (2005). *The grid: Core technologies.* Chichester: Wiley and Sons.

Maldonado, M. F. (2004). Grid computing in higher education: Trends, values and offerings (IBM White Paper). Retrieved August 7, 2006, from http://www.ibm.com/grid/pdf/grid_computing_in_higher_ed.pdf

Rumble, G. (1999). Costs of networked learning: What have we learned. In *Proceedings of the Conference on Flexible Learning on the Information Superhighway,* Sheffield, England. Retrieved August 7, 2006, from http://www.shu.ac.uk/flish/rumblep.htm

Rust, R. T., & Kannan, P. K. (2003). E-service: A new paradigm for business in the electronic environment. *Communications of the ACM, 46*(6), 36-42.

Saunders, G., & Klemming, F. (2003). Integrating technology into a traditional learning environment. *Active Learning in Higher Education, 4*(1), 74-86.

Singh, P. M., & Huhns, M. N. (2005). *Service oriented computing, semantics, processes, agents.* Chichester: John Wiley and Sons.

Talia, D., & Trunfio, P. (2003). Toward a synergy between P2P and grids. *IEEE Internet Computing, 7*(4), 94-96.

Van Weert, T. J., & Pilot, A. (2003). Task-based team learning with ICT, design and development of new learning. *Education and Information Technologies, 8*(2), 195–214.

Xenos, M., Pierrakeas, C., & Pintelas, P. (2002). Survey on student dropout rates and dropout causes concerning the students in the course of informatics of the Hellenic Open University. *Computers & Education, 39*(4), 361-377.

This work was previously published in the Journal of Cases on Information Technology, edited by M. Khosrow-Pour, Volume 9, Issue 1, pp. 36-53, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 2.23
A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture

Hiroshi Wada
University of Massachusetts – Boston, USA

Junichi Suzuki
University of Massachusetts – Boston, USA

Katsuya Oba
OGIS International, Inc., USA

ABSTRACT

Service oriented architecture (SOA) is an emerging style of software architectures to reuse and integrate existing systems for designing new applications. Each application is designed in an implementation independent manner using two major abstract concepts: services and connections between services. In SOA, non-functional aspects (e.g., security and fault tolerance) of services and connections should be described separately from their functional aspects (i.e., business logic) because different applications use services and connections in different non-functional contexts. This paper proposes a model-driven development (MDD) framework for non-functional aspects in SOA. The proposed MDD framework consists of (1) a Unified Modeling Language (UML) profile to model non-functional aspects in SOA, and (2) an MDD tool that transforms a UML model defined with the proposed profile to application code. Empirical evaluation results show that the proposed MDD framework improves the reusability and maintainability of service-oriented applications by hiding low-level implementation technologies in SOA.

INTRODUCTION

A key challenge in large-scale distributed systems is to reuse and integrate existing systems to build new applications in a cost effective manner (Vinoski, 2003; Zhang, 2004). Service Oriented Architecture (SOA) addresses this challenge by improving the reusability and maintainability of distributed systems (Arsanjani, Zhang, Ellis, Allam, & Channabasavaiah, 2007; Bichler & Lin, 2006; Endrei, Ang, Arsanjani, Chua, Comte, Krogdahl, Luo, & Newling, 2004; Foster, 2005; Lewis, Morris, Brien, Smith, & Wrage, 2005; Papazoglou, 2003). It is an emerging style of software architectures to design applications in an implementation independent manner using two major abstract concepts: *services* and *connections* between services. Each service encapsulates the function of a subsystem in an existing system. Each connection defines how services are connected with each other and how messages are exchanged through the connection. SOA hides the implementation details of services and connections (e.g., programming languages and remoting middleware) from application developers. They can reuse and combine services to build their applications without knowing the implementation details of services and connections.

In order to make this vision of SOA a reality, this article focuses on a research issue of increasing the reusability of services and connections and addresses this issue by separating non-functional aspects (e.g., security and fault tolerance) of services and connections from their functional aspects. The separation of functional and non-functional aspects can improve the reusability of services and connections because it allows different applications to use services and connections in different non-functional contexts. For example, an application may unicast messages to a service and another may manycast messages to multiple replicas of the service to improve fault tolerance. Also, an application may send signed and encrypted messages to a service, when the messages travel to the service through third-party intermediaries, in order to prevent the intermediaries from maliciously sniffing or altering the messages. Another application may send plain messages to the service via unsecured connection when the service is hosted in-house. The separation of functional and non-functional aspects can also improve the ease of understanding application design and enable the two different aspects to evolve independently. This results in higher maintainability of applications.

This article describes a model-driven development (MDD) framework for non-functional aspects in SOA. The MDD framework consists of (1) a Unified Modeling Language (UML) profile to model non-functional aspects in SOA, and (2) an MDD tool that accepts a UML model defined with the proposed profile and transforms it to application code (e.g., program code and deployment descriptors). The proposed UML profile allows application developers to graphically describe and maintain non-functional aspects in SOA as UML diagrams (composite structure diagrams and class diagrams). Using the proposed UML profile, non-functional aspects can be modeled without depending on any particular implementation technologies. The proposed MDD tool, called Ark, transforms implementation independent UML models into implementation specific application code.

This article describes design details of the proposed UML profile and demonstrates how Ark transforms an input UML model to application code that runs with certain implementation technologies such as Enterprise Service Buses (ESBs) (Chappell, 2004), secure file transfer protocols and grid computing platforms. Empirical evaluation results show that the proposed MDD framework improves the reusability and maintainability of service-oriented applications by hiding implementation technologies in UML models.

CONTRIBUTIONS

This article offers the following three contributions to the design space of service-oriented applications.

- **Modeling support for non-functional aspects in SOA**: This work is the first attempt to investigate a UML profile to consistently model a wide range of non-functional aspects in SOA, although there exist several UML profiles for specific aspects (e.g., functional aspects and service discovery) in SOA. (See the Related Work section for more details.) The proposed UML profile covers the following four areas of non-functional aspects.
  1. **Service deployment semantics:** Service redundancy.
  2. **Message transmission semantics:** Messaging synchrony, message delivery assurance, message queuing, multicast, manycast, anycast, message routing, message prioritization, messaging timeout, message logging, and message retention.
  3. **Message processing semantics:** Message conversion, message split, message aggregation, message validation, and message filtering.
  4. **Security semantics:** Transport-level encryption, message-level encryption (entire/partial message encryption), message signature, message access control, and service access control.
- **Modeling support for regulatory compliance**: As regulatory compliance has been becoming an important factor in software development and maintenance, regulatory mandates (e.g., the Sarbanes-Oxley Act and HIPPA) dramatically increase the number of non-functional aspects that application developers need to consider (O'Grady, 2004). This work is the first attempt to investigate a visual modeling language to describe non-functional aspects derived from regulatory mandates. The proposed UML profile allows application developers (or compliance management staffs) to graphically specify and verify how their applications meet regulatory mandates. Currently, the proposed UML profile addresses data retention, data/process validation (e.g., consistency validation among an order, invoice, and payment) and security (e.g., access control and data integrity).
- **MDD support for service-oriented applications**: Non-functional requirements change during application lifecycle more often than functional aspects (Bieberstein, Bose, Fiammante, Jones, & Shah, 2005). It can be expensive to manage frequent changes in non-functional requirements. This results in escalating maintenance cost, in turn total cost of owning. When a non-functional requirement (e.g., security policy) changes in an application, the proposed MDD framework allows application developers to make the change in a UML model specifying the application's non-functional aspects and keep its functional part intact. The proposed MDD tool (Ark) generates non-functional code from the updated UML model and combines the generated code with existing functional code. Ark makes application's functional aspects reusable across the changes in non-functional requirements, thereby improving the productivity of application development and maintenance.

BACKGROUND AND A MOTIVATING EXAMPLE

UML is a modeling language to describe application designs as graphical diagrams. It specifies the syntax (or notation) and semantics of every model element that appears in diagrams (e.g.,

class, interface and association). The syntax and semantics are defined in the UML metamodel (Object Management Group, 2004), which is the grammar specification for standard (default) model elements in UML.

In addition to standard model elements, UML provides extension mechanisms (e.g., stereotypes and tagged-values) to specialize them to precisely describe domain or application specific concepts (Fuentes & Vallecillo, 2004). A stereotype is applied to a standard model element and specializes its semantics to a particular domain or application. Each stereotyped model element can have data fields, called tagged-values, specific to the stereotype. Each tagged-value consists of a name and value. A particular set of stereotypes and tagged-values is called a UML profile.

For example, a UML profile for Enterprise Java Beans (EJB) (Java Community Process, 2001) defines the stereotype `<<EJBEntityBean>>`, which extends `Class` in the UML metamodel. This means the stereotype can be applied to classes. Thus, a UML class stereotyped with `<<EJBEntityBean>>` indicates that the class is designed as an EJB entity bean. The stereotype `<<EJBEntityBean>>` has a tagged-value, called `EJBPersisitenceType`, to specify who provides persistence to an entity bean. The tagged-value can have a value `Bean` or `Container`. `Bean` indicates an individual entity bean is responsible for its own persistence, and `Container` indicates an EJB container takes care of persistence.

Figure 1 shows an overview of an example purchasing system across buyers, retailers, sup-

Figure 1. The structural architecture of an example purchasing system

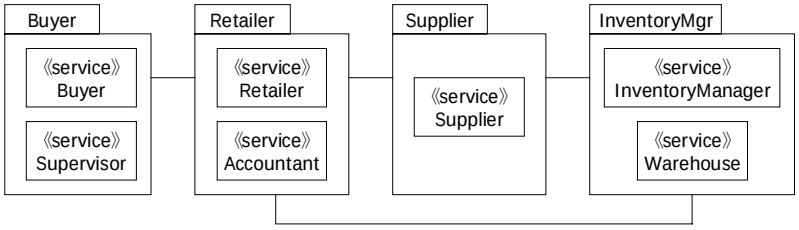
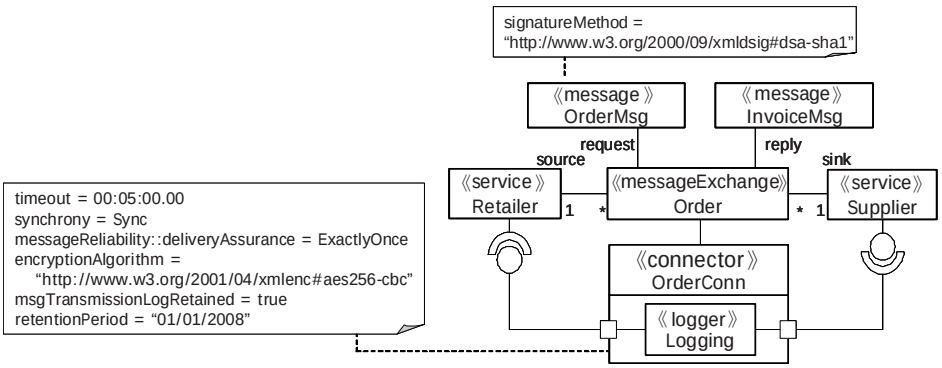


Figure 2. An example UML model



pliers, and inventory managers. All example models in this article focus on and define several particular parts of this system. In this example system, a Buyer purchases a product from a Supplier via Retailer. A Supervisor authorizes each order that a Buyer places. An Accountant performs accounting tasks for a Retailer. An InventoryManager manages a Retailer's inventory.

Figure 2 shows an example model built with the proposed UML profile. This model focuses on an interaction between a Retailer and Supplier in Figure 1, and defines an order processing scenario in which a Retailer places an order and a Supplier issues an invoice. In this example, two services (Retailer and Supplier) exchange messages. Each service is represented by a class stereotyped with <<service>>. These services exchange two types of messages (OrderMsg and InvoiceMsg), each of which is stereotyped with <<message>>. Each message can have multiple tagged-values to specify additional message transmission/processing semantics. In this example, the tagged-value signatureMethod specifies that an OrderMsg carries a digital signature created with DSA (Digital Signature Algorithm). Each pair of a request and reply messages is represented by a class stereotyped with <<messageExchange>>.

<<connector>> represents a connection that transmits messages between services. In this example, messages are delivered through a connector called OrderConn. Every message exchange is bound with a connector in order to specify which connector is used to deliver messages. A connector has a provided interface (represented as a "ball" notation) and a required interface (represented as a "socket" notation) to transmit messages between services. Services use the provided and required interfaces to send and receive messages, respectively. The two interfaces are intended to show how services use (connect with) a connector.

Each connector can have multiple filters inside. They are used to define message transmission/processing semantics in a connector. This example uses a Logger in the OrderConn connector. Logger logs messages that transmitted through the filter (OrderMsg and InvoiceMsg in this example).

Also, each connector can have multiple tagged-values to specify additional message transmission/processing semantics. In this example, OrderConn specifies the timeout of message transmissions (five minutes), the synchrony of message transmissions (synchronous), the assurance level of message delivery (exactly once) and the message encryption algorithm (Advanced Encryption Standard). Also, through the use of tagged-values msgTransmissionLogRetained and retentionPeriod, OrderConn specifies to retain the logs of message transmissions until a certain date.

As shown above, the proposed UML profile provides a visual and intuitive abstraction to model the architectures and non-functional aspects of service-oriented applications.

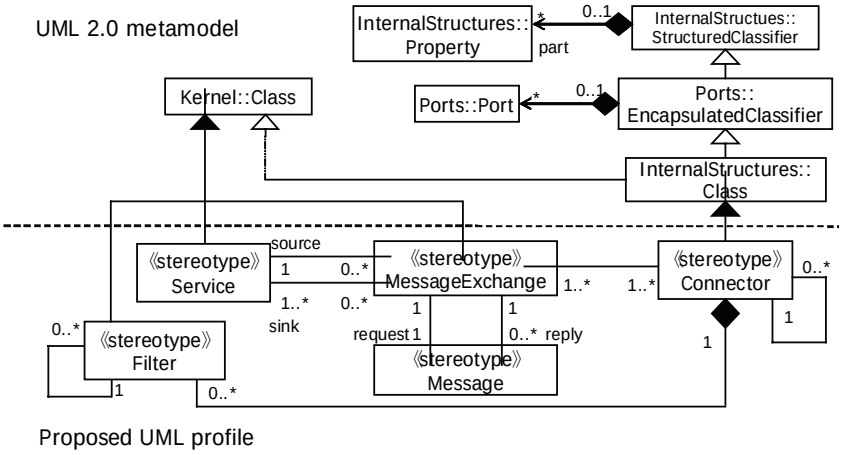
DESIGN OF THE PROPOSED UML PROFILE

The proposed UML profile provides key model elements to specify service-oriented applications: *service*, *message exchange*, *message*, *connector*, and *filter*, each of which is defined as stereotypes (Table 1). Figure 3 shows how the proposed profile defines these stereotypes by extending the UML metamodel. Each stereotype is defined as a metaclass stereotyped with <<stereotype>>[1]. Except Connector, four stereotypes inherit the Class metaclass in the Kernel package of the UML metamodel. Thus, they are applied to classes in user-defined models (see Figure 2). A Service can be a source or sink of each request/reply message. The source and sink are identified with source and sink, roles on two associations between a MessageExchange and Services (Figure 2). Each MessageExchange may have multiple reply messages per request message (Figure 3). Using multiplicity on two associations between a MessageExchange

Table 1. Key model elements (stereotypes) in the proposed UML profile

Stereotype	Description
<<service>>	Represents a service.
<<messageExchange>>	Represents a pair of a request and reply messages. Specifies which services send and receive the messages.
<<message>>	Represents a (request or reply) message.
<<connector>>	Represents a connection between services (i.e., message source and destination). Defines the semantics of message transmission and processing. Specifies which messages (message exchange) to transmit.
<<filter>>	Customizes the semantics of message transmission and message processing in a connector.

Figure 3. Definition of stereotypes



and `Services`, `MessageExchange` can indicate one-to-one (unicast) and one-to-many (multicast or manycast) message exchanges. For example, Figure 2 shows a one-to-one message exchange between a `Retailer` and a `Supplier`.

`Connector` is a stereotype extending the `Class` metaclass in the `InternalStructures` package of the UML metamodel (Figure 3). This metaclass defines a composite class, a special type of class, which can contain other model elements (e.g., inner classes)[2] and have `Ports` to specify how internal model elements interact with external elements. In the proposed UML profile, a `Connector` can contain `Filters` to specify the semantics of message transmission and message processing. The `Ports` connected with a `Connector` identify the `Messages` it receives and sends out, using association roles `input` and `output`. For example,

Figure 2 shows the `OrderConn` connector , which contains a filter (a `Logger`). This filter receives, records message's log, and sends out `OrderMsg` or `InvoiceMsg` messages.

Connector

`Connector` has 10 tagged-values (Figure 4). `timeout` is a mandatory tagged-value to specify the timeout period (in millisecond) in which a connector needs to deliver each message. If a message is not delivered to its destination (sink) within the timeout period, a connector discards the message. In Figure 2, the timeout period of the connector `OrderConn` is specified as five minutes.

`synchrony` is a mandatory tagged-value to specify the synchrony semantics of message transmissions between a message source and destination. Synchronous, asynchronous and oneway non-blocking semantics are defined as an enumeration in `Synchrony` (Figure 4), and each connector chooses one of them. In Figure 2, a `Retailer` and a `Supplier` exchange `OrderMsg` and `InvoiceMsg` messages synchronously.

`priority` is a mandatory tagged-value to specify the priority of each message that a connector delivers. The range of `priority` is from 0 to 255 (0 is the lowest and 255 is the highest), and the default value is 0 (Figure 4).

`inOrder` is a mandatory tagged-value to specify whether the order of messages that a service (message destination) receives is same as the order of messages that the other service (message source) sends out. The default value of `inOrder` is false.

`deliveryAssurance` is an optional tagged-value to specify the assurance level of message delivery. Three different semantics are defined as an enumeration in `DeliveryAssurance` (Figure 4), and each `Connector` chooses one of them at a time. `AtLeastOnce` means that a connector retries delivering a message until its destination receives the message. (A message retransmission is triggered with the `timeout` tagged-value.) However, the message may be delivered to its destination more than once. `AtMostOnce` means that a connector discards a message if the message has already been delivered to its destination; however, there is no guarantee of message delivery. `ExactlyOnce` satisfies the requirements of the above both semantics. It guarantees that a connector delivers a message to its destination without duplications. When `inOrder` is true, `ExactlyOnce` is implicitly (automatically) set to `deliveryAssurance` because duplicated or missing messages violate the `inOrder` semantics.

Figure 5 shows an example model using `inOrder` and `deliveryAssurance`. This example illustrates an extension to an order processing

Figure 4. Tagged-values of connector

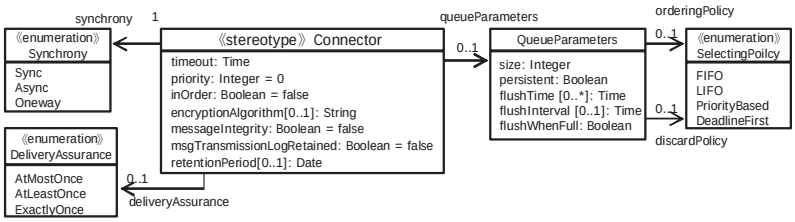
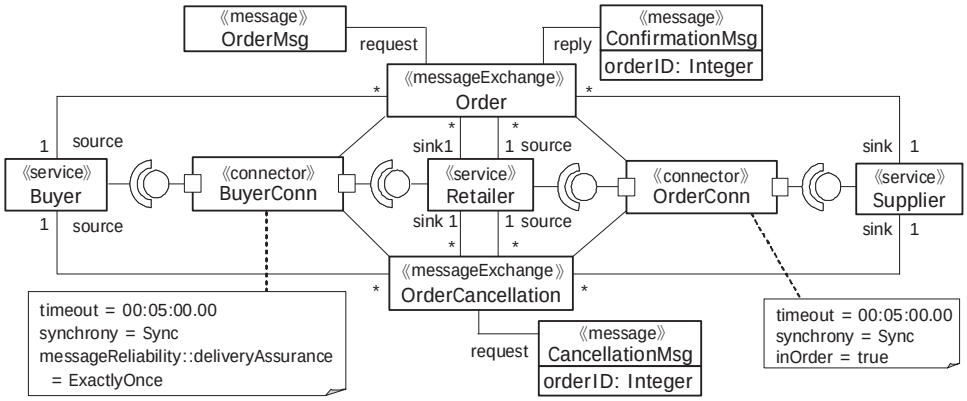


Figure 5. An example of inorder and delivery assurance



application in Figure 2. In this example, a Buyer transmits an `OrderMsg` to a `Supplier` via `Retailer` (see also Figure 1). After a `Retailer` forwards an `OrderMsg` from a Buyer to a `Supplier`, the `Buyer` can cancel the order by transmitting a `CancellationMsg` to the `Retailer`, and in turn, to the `Supplier`. In this example, the order of message transmissions is important between `Retailer` and `Supplier` because an order must be delivered to a `Supplier` before a corresponding order cancellation. Therefore, the `inOrder` semantics is assigned to the `OrderConn` connector. This semantics implicitly assigns `ExactlyOnce` to the `deliveryAssuarance` semantics in the `OrderConn` connector.

`encryptionAlgorithm` is an optional tagged-value used for transport-level encryption in a connector. This tagged-value defines an algorithm to secure a connection upon which request and response messages are transmitted. (See Figure 2 for an example.) The encryption algorithm is specified as a URI defined in the XML Encryption specification (World Wide Web Consortium, 2002). For example Triple DES is represented with `http://www.w3.org/2001/04/xmlenc#tripledes-cbc`, and AES-256 (Advanced Encryption Standard) is represented with `http://www.w3.org/2001/04/xmlenc#aes256-cbc`.

`queueParameters` is an optional tagged-value to deploy a message queue between services (i.e., message source and destination) and specify the semantics of message queuing between them. `size` specifies the maximum number of queued messages. `flushWhenFull` specifies whether queued messages are flushed from a queue to their destinations when the queue overflows. When `flushWhenFull` is false, the overflowing queue discards a message according to `discardPolicy` (Figure 4); discarding the oldest message (First-In-First-Out), the newest message (Last-In-First-Out), the lowest priority message or the closest deadline message. These four policies are defined as an enumeration in `SelectionPolicy` (Figure 4). `flushTime` and `flushInterval` specify when and how often a queue flushes messages, respectively. `orderingPolicy` specifies how to order messages in a queue: FIFO, LIFO, highest-priority-first or earliest-deadline-first. `persistent` specifies

whether a queue stores messages in a storage (e.g., a file or database) so that the queue can recover them when it crashes unexpectedly.

Figure 6 shows an example using `queueParameters`. It illustrates an inventory management application for retailers (see also Figure 1). Each `Retailer` transmits an `OrderMsg` to an `InventoryManager` when it has no or few products in stock. The `InventoryManager` receives `OrderMsg`s from multiple `Retailers` every two hours in a batch manner. The `OrderConn` connector implements a synchronous queue that stores and forwards

Figure 6. An example of queue

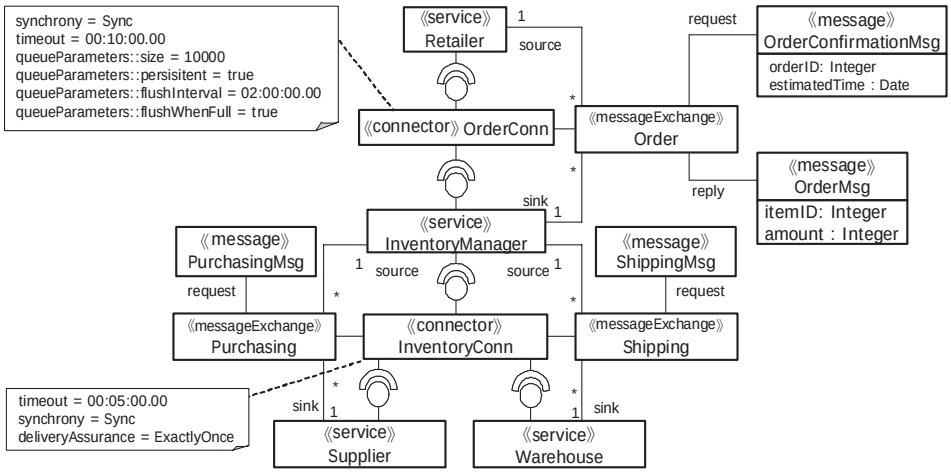
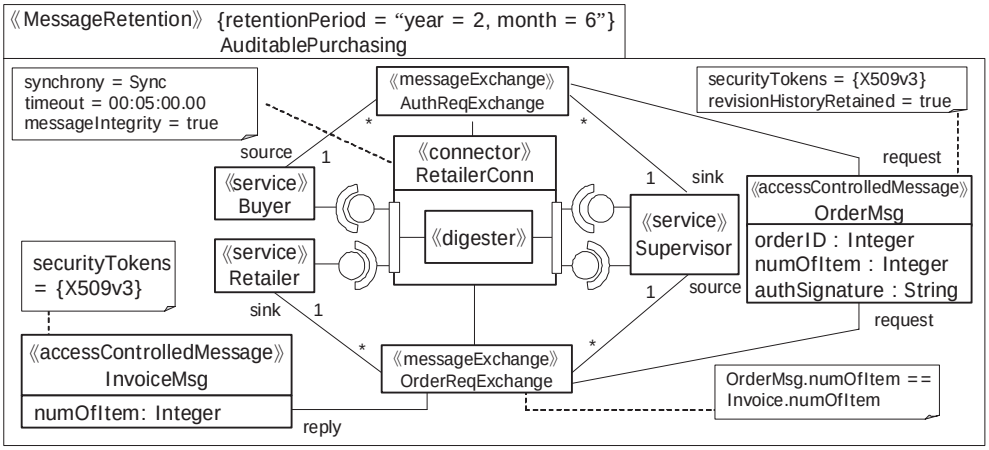


Figure 7. An example model for regulatory compliance



OrderMsgs. The InventoryManager schedules which warehouses deliver which products to which retail stores (every two hours), and based on the shipping schedule, sends ShippingMsgs to Warehouses. If a warehouse has a small inventory of a particular product, the InventoryManager orders the product by sending a PurchasingMsg to a Supplier.

msgTransmissionLogRetained is a mandatory tagged-value to specify whether to retain logs on message transmissions (see Figure 2 for an example). Regulatory mandates require applications to retain the logs and make them auditable for the third party organizations in the future. A connector with this tagged-value records (1) which messages are transmitted, (2) message source and destination (services), and (3) when the messages are transmitted. If msgTransmissionLogRetained is true, retentionPeriod must be specified to define the period to retain each message transmission log. The default value of msgTransmissionLogRetained is false. If it is false, retentionPeriod is ignored.

messageIntegrity is a mandatory tagged-value to specify whether to ensure the message integrity. The default value of messageIntegrity is false. A connector with this tagged-value checks whether messages are changed during their transmission.

Figure 7 illustrates an order processing application in which a Buyer places an order and a Retailer receives it via authorization by a Supervisor. By assigning a signature to the authSignature data field of an OrderMsg, Supervisor authorizes the message (order). Services are connected through a connector with the messageIntegrity semantics. This semantics ensures that OrderMsg messages are not altered during their transmission, and eliminates the possibility of malicious alteration.

A package stereotyped with <<messageRetention>> specifies that contained connectors have the msgTransmissionLogRetained semantics implicitly if the connectors omit it (Figure 7). Each connector follows the retentionPeriod specified in the package. When a connector specifies msgTransmissionLogRetained and retentionPeriod explicitly, they override the retentionPeriod specified in a package. Also, a package stereotyped with <<messageRetention>> enforces contained services and messages to log their message transmissions. Connectors, services, and messages retain their logs independently, and the third party organizations can discover fraud activities by checking the inconsistencies between their logs.

Application developers can specify constraints using the Object Constraint Language (OCL). Constraints are used to ensure the consistency among application data and check whether services work correctly. For example, in Figure 7, OrderReqExchange has a constraint that ensures the number of items in an order (OrderMsg) and a corresponding invoice (InvoiceMsg) are always equal. When this constraint is violated, fraud activities could be committed.

Filter

This article describes eight of the filters that the proposed UML profile defines. Filters are defined as stereotypes extending the Filter stereotype (Figure 8). New filters can be defined as its subclasses. This section shows six filters to specify message transmission semantics and two filters to specify message processing semantics.

The stereotypes Multicast, Manycast, Anycast, Router, Logger, and Digester are used to define the message transmission semantics in a connector. A Multicast filter receives a request message from its source and transmits it to multiple destinations (services) simultaneously (one-to-many message exchange). A group of destinations can contain different types of services. When the Multicast filter receives reply messages from the destinations, it sends them back to the source of the request message. Multicast is used to improve the efficiency of message transmissions.

Figure 9 shows an example that models an application for wholesale price notification using `Multicast`. A `Retailer` subscribes for the price changes of a particular supply, and a `Supplier` notifies (or publishes) any price changes to the `Retailer`. A `Retailer` transmits a `Subscription` message to a `Supplier` in a synchronous and exactly-once manner. A `Supplier` multicasts a `PriceNotificationMsg` message, which contains a supply's GTIN (Global Trade Item Number)[3] and price, to multiple `Retailers` in asynchronous and at-most-once semantics.

`Manycast` is used to improve fault tolerance by forwarding a request message to a group of replicated destinations (i.e., to the same type of services). The tagged-value `groupSize` specifies how many services are deployed as a group. `standby` specifies the operation of replicated services: *hot standby*, *warm standby*, or *cold standby*. In hot standby, all services in a group remain active to receive request messages. A `Manycast` filter sends a message to all services in a group. `Manycast` returns only one reply message to the source of a request message, out of multiple replies from

Figure 8. Tagged-values of filters

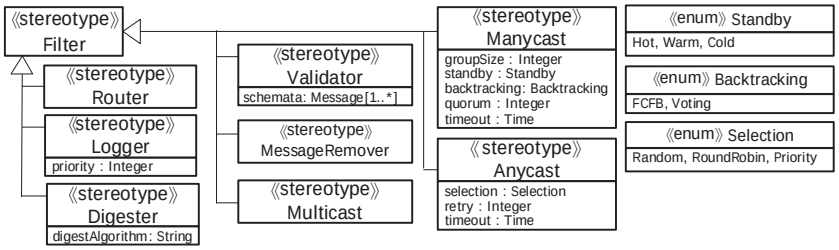
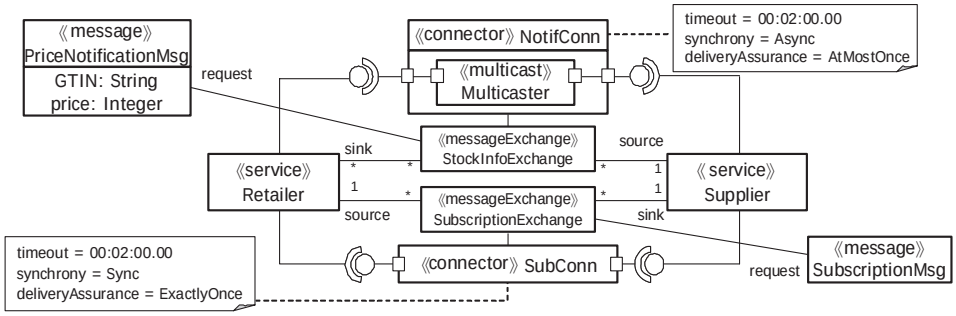


Figure 9. An example of multicast



services. `backtracking` defines two policies to decide which reply message to be returned. When `FCFB` (first-come-first-backtracked) is selected, a `Manycast` filter returns the first reply that it receives from destination services. When `Voting` is selected, the `Manycast` filter performs a voting process. It counts the number of reply messages and inspects their contents. If the number of replies that have the same content reaches `quorum`, the `Manycast` filter returns one of the replies. If the number does not reach `quorum` within `timeout`, the `Manycast` filter returns the reply that generates the highest voting count.

In warm standby, all services in a group remain active to receive request messages. A `Manycast` filter sends a message to all services in a group, but only one service returns a reply. In this case, `backtracking` is not used. In cold standby, only one service in a group is active, and a `Manycast` filter sends a message to the service. If the service does not respond within `timeout`, the filter activates another service in the group and sends a message to the service. In cold standby, `backtracking` is not used.

In an example model shown in Figure 10, a supplier sends an inquiry to a cluster of transaction record servers to obtain a transaction record containing a set of orders. A manycast filter, `Replicator`, is used in the connection `RecordConn`. The filter intercepts each `Inquiry` (request) message and sends it to three replicated instances of `TransactionRecServer`, which is maintained with the hot standby policy. `Replicator` returns a `TransactionRecordMsg` (reply message) to a `Supplier` on FCFB basis.

`Anycast` is a variation of the hot standby policy in `Manycast`. It forwards a request message to only one destination in a group of replicated services. This filter is used to balance workload placed on services. `selection` defines how to choose a destination from multiple services; randomly, on round robin or on destination's priority basis (the service with the highest priority in a group is selected). If an `Anycast` filter fails to deliver a request message within `timeout`, it retries to forward the request message. `retry` specifies the maximum number of retries. If the `Anycast` filter fails the maximum number of retries, it returns an error message to the source of the request message.

Figure 11 shows an example model describing a content delivery system, for example, for delivering contents among supplier's online catalogs of their supplies. (For simplicity, tagged-values of the connector `RedirectionConn` and

Figure 10. An example of manycast

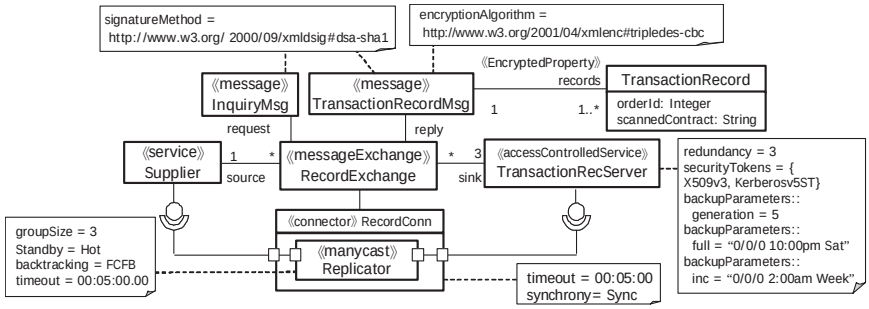
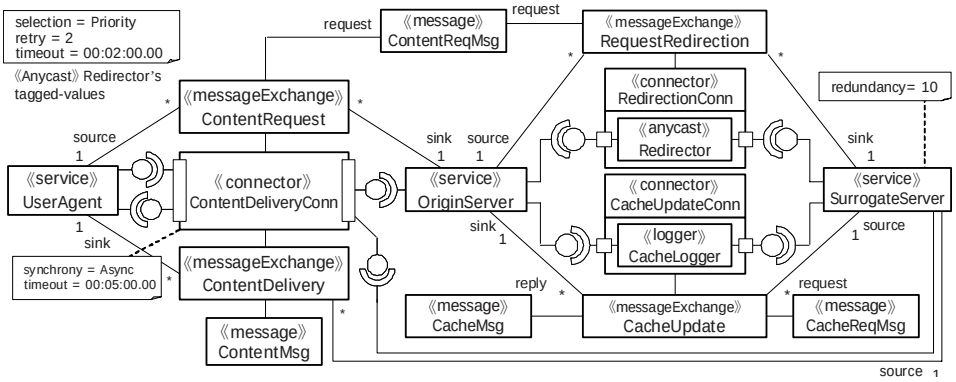


Figure 11. An example of anycast



`CacheUpdateConn` are omitted, but both have the synchronous semantics and their timeout is five minutes.) A user agent (`UserAgent`) sends a request (`ContentReqMsg`) to a content server (`OriginServer`) through a connector (`ContentDeliveryConn`). To balance workload, the content server redirects the request to a surrogate server (`SurrogateServer`). This model has a cluster of surrogate servers which consists of 10 replica servers. An `Anycast` filter in the connector `RedirectionConn` selects one of servers on their priority basis and redirects a request to it. (tagged-values of the `Anycast` filter is described on the upper left corner.) If a surrogate server does not have data to process a request, it sends a request (`CacheReqMsg`) to a content server to obtain cache data (`CacheMsg`). After processing a request, a surrogate server returns content (`ContentMsg`) to a user agent.

`Router` routes an incoming message to one or more destinations with certain criteria. Since UML does not provide a means to define rules, the proposed profile has no facility to specify routing rules at design time. Supporting tools transform a `Router` to a skeleton source code (e.g., in Java) or a rule description (e.g., in XPath) that performs message routing. Developers are expected to complete the skeleton code/description.

`Logger` records the transmission of each message whose priority value is higher than `priority`. When `priority` is omitted, all message transmissions are recorded.

`Digester` records digest values of all messages. This filter can be used to check whether a message is altered after its transmission. The digest algorithm is specified as a URI defined in the XML Encryption specification (The World Wide Web Consortium, 2002a). For example, `http://www.w3.org/2000/09/xmldsig#sha1` specifies SHA-1.

In addition to the stereotypes for message transmission semantics, the proposed UML profile provides two other stereotypes to define the message processing semantics in each connector: `Validator` and `MessageRemover` (Figure 8). `Validator` and `MessageRemover` validate incoming messages against the message schema specified in its tagged-value `schemaURI` and a given criteria (e.g., rules specifying valid data ranges) respectively, and transmit only validated messages. Since UML does not provide a means to define rules, the proposed profile has

no facility to specify message filtering rules for `MessageRemover` at design time. Supporting tools transform a `Validator` and `MessageRemover` to a skeleton source code (e.g., in Java) or a rule description (e.g., in XPath) that performs message filtering. Developers are expected to complete the skeleton code/description. When a connector is encrypted with `encryptionParameter`, `Validator` and `MessageRemover` in the connector cannot validate messages (all messages are transmitted to their destinations.)

Service

`Service` has six tagged-values (Figure 12). `timeout` is an optional tagged-value to specify the timeout period (in millisecond) of each message that a service issues. If a message is not delivered to its destination within this time period, a connector discards the message.

`priority` is an optional tagged-value to specify the priority of each message that a service issues. `Anycast` filter uses `priority` to select its destinations. Also, it is used to order messages in a message queue when a connector has `queueParameters`.

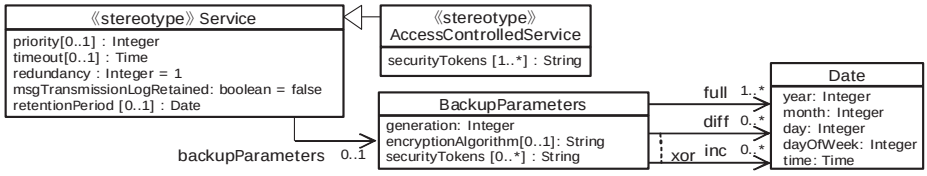
Each service is expected to have data fields corresponding to the `priority` and `timeout` tagged-values. Usually, class instances cannot read and write tagged-values because tagged-values are defined in a metamodel (see Figure 3) and used in a model. The data fields allow different class instances to have different tagged-values, and tagged-values specified in a model behave as default values of corresponding data fields.

`redundancy` is an optional tagged-value to specify the number of runtime instances of a service. This tagged-value must be specified when a service is accessed by `Manycast` or `Anycast` filters. In Figure 10, three instances of `TransactionRecServer` are used for fault tolerance with the manycast filter `Replicator`.

Same as a connector, a service with `msgTransmissionLogRetained` and `retentionPeriod` records information on its message transmissions. A package stereotyped with `<<messageRetention>>` specifies that enclosed services have the `msgTransmissionLogRetained` semantics implicitly if the services omit it (Figure 7). When a service specifies `msgTransmissionLogRetained` and `retentionPeriod` explicitly, they can overrides package's ones.

`backupParameters` is an optional tagged-value to specify service's backup policy. `full`, `diff`, and `inc` specify the time when full, differential, and incremental backups is performed respectively. The class `Calendar` can specify a specific time in point and a repetition time. For example, when `year` is omitted (value zero means omission), the backup is performed every year (the date and time to perform a backup is specified by other data fields in `Calendar`). Full backup stores all

Figure 12. Tagged-value of service



data in a service, differential backup stores all data which have been modified since the last full backup, and incremental backup stores all data which have been modified since the last full or incremental backup. Differential backup requires much amount of storage and longer time to perform than incremental backup, but it can restore data faster. Also, data redundancy in differential backup reduces the risk of data loss. Differential and incremental backups must be used with full backup, and full backup must be performed at least once before differential or incremental backups are performed. One backup policy can have either differential or incremental backup at a time (xor). If `diff` and `inc` are omitted, only full backup is performed. `generation` specifies the number of full backups retained in a storage. `encryptionAlgorithm` specifies an algorithm to secure backup data. `encryptionAlgorithm` specifies an algorithm to secure backup data. `securityTokens` specifies security tokens for the purpose of authentication (see below). In Figure 10, the service `TransactionRecServer` has a backup policy. The backup policy specifies the generation (five), the time when full and incremental backup are performed (10:00pm on every Saturday and 2:00am on week days respectively).

`AccessControlledService` is a stereotype extending the stereotype `Service` (Figure 12). It is a special type of service that enforces an access control policy. The tagged-value `securityTokens` is mandatory to specify security tokens (or certificates). The security tokens are used to authenticate entities (e.g., services) that access a message. This tagged-value can contain multiple values in order of precedence. The values use the names defined in the WS-SecurityPolicy specification (Organization for the Advancement of Structured Information Standards, 2005). In Figure 10, `TransactionRecServers` control accesses from `Supplier`s using X.509 certificates or Kerberos tickets. Since UML does not provide a good means to describe policies (or rules), the proposed UML profile does not define how to specify access control policies. `<<accessControlledService>>` is used only for indicating a service implements a certain access policy. A supporting tool transforms an `AccessControlledService` to a skeleton program code or an access control description in accordance with an implementation technology that an application designer chooses. Application developers are required to complete implementing access control policies.

In addition to the general type of service, the proposed UML profile provides three special types of services, `MessageConverter`, `MessageSplitter`, `MessageAggregator`, to define the message processing semantics. They inherit the `Service` stereotype.

`MessageConverter` converts an incoming message with a given rule. Similar to `Router`, supporting tools transform a `MessageConverter` to a skeleton source code or rule description (e.g., in XSLT) that performs message conversions, and developers complete the skeleton code/description.

`MessageSplitter` divides an incoming message into multiple fragments with a certain rule. Similar to `MessageConverter`, supporting tools transform a `MessageSplitter` to a skeleton code or rule description that performs message split, and developers complete the skeleton. In an example model shown in Figure 13, a `Retailer` sends an order message (`OrderMsg`) to a `MessageSplitter`, and the splitter divides the message into two fragments (`PurchasingMsg` and `AccountingMsg`), and sends them to different destinations (`Supplier` and `Accountant`). The destinations directly returns reply messages (`PurchasingConf` and `AccountingConf`) to the `Retailer`. The connector `OrderConn` encrypts all messages with Triple DES. Also, the message `OrderMsg` retains routing information, which includes source of a message. (i.e., it is auditable which customer sends which message.)

`MessageAggregator` combines multiple incoming messages. Figure 14 shows an example extending the model in Figure 2. In addition to `OrderMsg`, `Retailer` sends a message `AuthReqMsg`

Figure 13. An example of messagesplitter

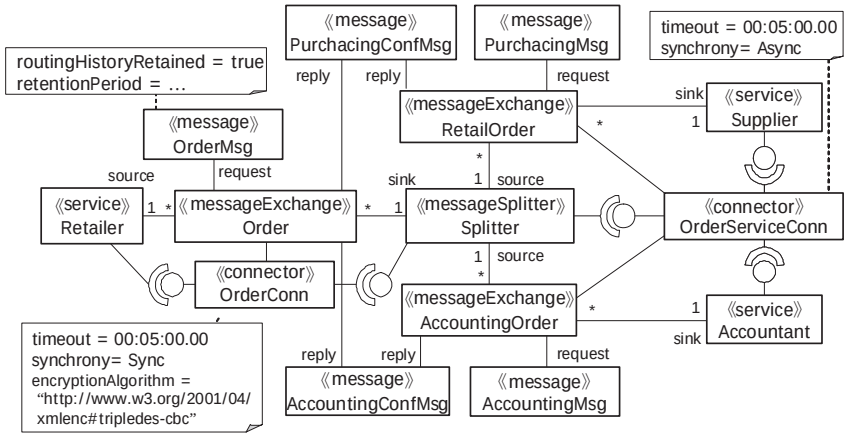
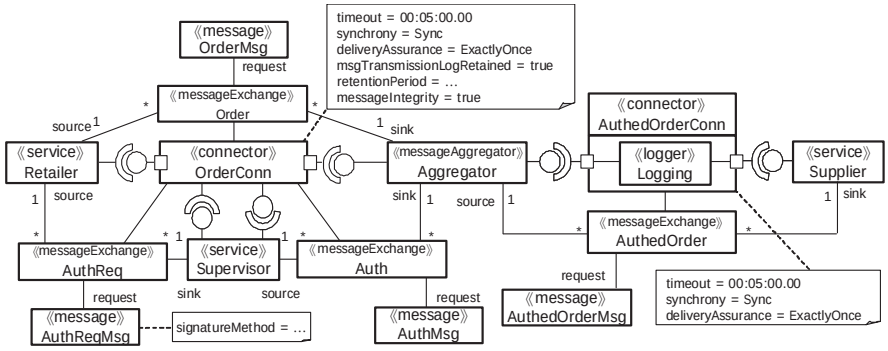


Figure 14. An example of messageaggregator



to ask the service `Supervisor` to authorize the order. `Aggregator` synchronizes and combines `OrderMsg` and `AuthMsg` (an authorization message from `Supervisors`), and it sends new message `AuthedOrderMsg` to `Supplier`. The connector `OrderConn` retains logs on message exchanges between `Buyer`, `Supervisor`, and `Aggregator`. It makes logs on order and authorization process auditable. Also, `OrderConn` ensure the integrity of messages.

Message

Message has seven tagged-values (Figure 15). schemaURI is a mandatory tagged-value to identify the schema of a message. The default value of schemaURI is message's qualified name (a combination of a package name and message's name).

priority and timeout are optional tagged-values to specify the priority and timeout period of messages. Connector has timeout, and Service also has those two tagged-values. The precedence is that Message's tagged-values override Service's ones, and Service's tagged-values override Connector's ones. Same as Service, each message is expected to have data fields corresponding to the priority and timeout tagged-values, and different message instance can have different priority and timeout.

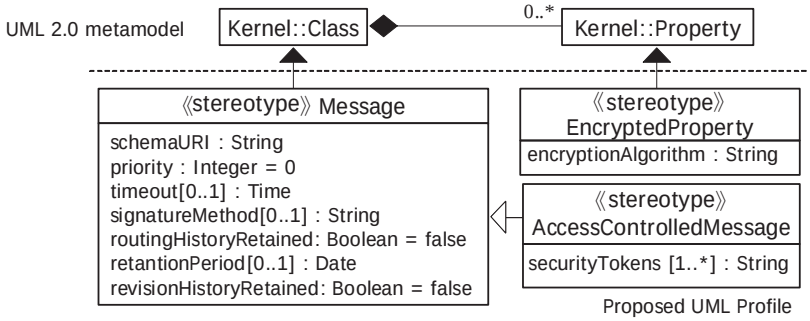
signatureMethod is used an optional tagged-value to ensure the integrity of a message. It specifies an algorithm for generating the message's digital signature. The algorithm is represented with a URI defined in the XML Signature specification (The World Wide Web Consortium, 2002b). For example, DSA (Digital Signature Algorithm) is represented with http://www.w3.org/2000/09/xmldsig#dsa-sha1. In Figure 10, each Inquiry and TransactionRecord message is signed with DSA. When signatureMethod is specified, each message is expected to maintain its signature in a data field called signature.

Same as a connector, a message with routingHistoryRetained and retentionPeriod records information on its message transmissions. A package stereotyped with <<messageRetention>> specifies that enclosed messages have the routingHistoryRetained semantics implicitly if the services omit it (Figure 7). When a message specifies routingHistoryRetained and retentionPeriod explicitly, they can overrides package ones.

revisionHistoryRetained is an optional tagged-value to specify whether to retain message's revision history (Figure 7). It makes the revision history to auditable in the future. A message with this semantics records 1) which data fields are revised, 2) how they revised (i.e., newly created, replaced, or deleted), 3) when they revised, and 4) who revised them. The tagged-value retentionPeriod is used to specify the period to retain the history. A package stereotyped with <<messageRetention>> specifies that enclosed messages have the revisionHistoryRetained semantics implicitly

The stereotype EncryptedProperty is used for message-level (end-to-end) encryption. It is

Figure 15. Tagged-values of message



defined as a stereotype extending `Property` in the UML metamodel (Figure 15). This stereotype is attached to data fields to be encrypted in a message. `EncryptedProperty` has a tagged-value, `encryptionAlgorithm`, to specify an algorithm used to encrypt a message. The semantics of this tagged-value is same as that of `encryptionAlgorithm` in `Connector`. An encryption algorithm is specified as a URI that the XML Encryption specification defines (World Wide Web Consortium, 2002a). Different data fields in a message can be encrypted with different encryption algorithms. In Figure 10, `orders` in `TransactionRecordMsg` are encrypted with Triple DES, which is represented with `http://www.w3.org/2001/04/xmlenc#tripledes-cbc`.

`AccessControlledMessage` is a stereotype extending `Message` (Figure 15). Similar to `AccessControlledService` it is a special type of message that enforces an access control policy. It removes the possibility of unauthorized accesses (i.e., altering messages by unauthorized users) and accidental altering (i.e. altering messages mistakenly by authorized users). The tagged-value `securityTokens` must be specified in `AccessControlledMessage` for the purpose of authentication. Since UML does not provide a good means to describe policies (or rules), the proposed UML profile does not define how to specify access control policies. `AccessControlledMessage` is used to indicate a message implements a certain access policy. A supporting tool transforms an `AccessControlledMessage` to a skeleton program code or an access control description in accordance with an implementation technology that an application developer chooses. Application developers are required to complete implementing access control policies.

APPLICATION DEVELOPMENT WITH THE PROPOSED MDD FRAMEWORK

This section describes a model-driven development (MDD) tool, called Ark, which accepts a UML model designed with the proposed UML profile and transforms the model into a skeleton of application code (program code and deployment descriptor). Currently, Ark implements transformations between the proposed UML profile and three middleware technologies: Mule ESB[4], ServiceMix ESB[5] and GridFTP[6] (Figure 16). UML models in this work are maintained with the metameta model of the Eclipse Modeling Framework (EMF; http://www.eclipse.org/emf/). The proposed UML profile is defined as an extension to the UML metamodel. Each application designer gives his/her UML model to Ark, and instructs Ark which transformation to use for generating skeleton application code.

Figure 17 shows the development process using Ark. (This figure assumes that generated application code uses Mule ESB.) Application designers define application models using the proposed UML profile (e.g., an example model in Figures 2). Ark Transformer, one of the components in Ark, takes the application models in the format of XML Metadata Interchange (XMI) and transforms the input models into application code compliant with Mule ESB.

Ark has been tested with MagicDraw[7], a visual UML modeling tool that can serialize UML models to XMI (Figure 18). Ark Transformer is implemented based on openArchitectureWare[8], a model transformation engine. Each input UML model (XMI file) is validated against the UML standard metamodel and the proposed profile's metamodel (see Figure 3), and transformed to application code for Mule ESB (Java programs and deployment descriptors in XML). A transformation rule between UML models and application

Figure 16. The Architecture of Ark

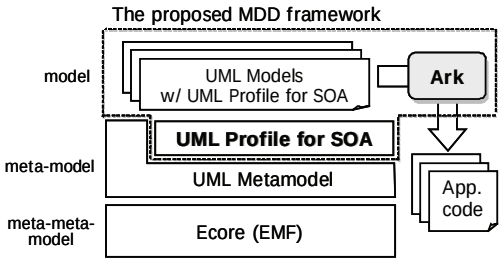


Figure 17. Application development with Ark

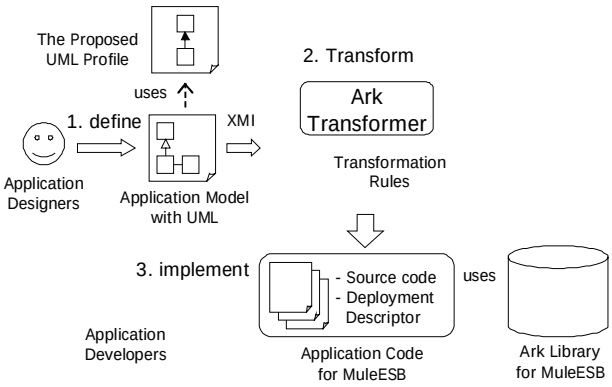
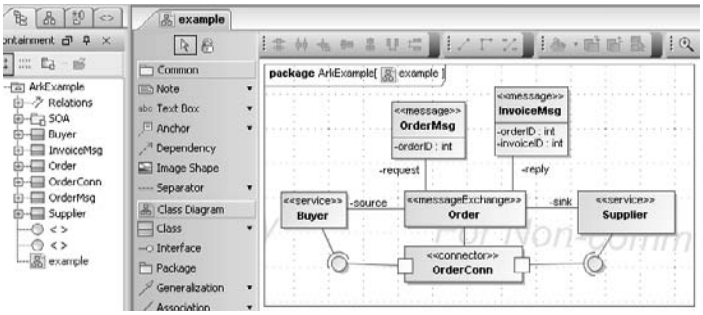


Figure 18. A UML model in magicdraw



code is implemented as a set of transformation templates, which define how to transform UML model elements to program elements in application code.

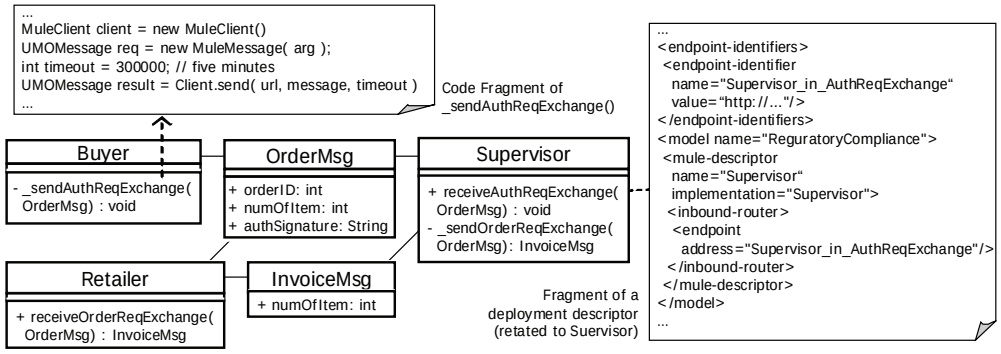
Transformation Rules for ESB Applications

Figure 19 shows some of the Java classes and deployment descriptors that Ark generates from the UML model in Figure 7 when Mule ESB is selected as middleware to operate applications. Table 2 shows the transformation between model elements in the proposed UML profile and program elements in Mule ESB. Ark transforms a UML class stereotyped with `<<message>>` to a Java class that has the same class name and the same data fields. The Java class implements the interface `Serializable`. This is required to implement messages exchanged with Mule ESB.

A UML class stereotyped with `<<service>>` is transformed to a Java class that has the same class name and the same data fields. Ark inserts several operations to the Java class, depending on whether its association role is `source` or `sink` against a message exchange. The operations are used to send and receive messages: `_sendX()` to send messages where X references the name of a message exchange, and `receiveX()` to receive messages. For example, in Figure 19, `Supervisor` has `_sendOrderReqExchange()` and `receiveAuthReqExchange()` to send and receive `OrderMsg` messages to `Supplier` and from `Retailer` respectively. `_sendX()` is supposed to be invoked by methods in the same service class. (This is why its visibility is private.) `receiveX()` is called by source services that have messages to deliver. A fragment of a deployment descriptor in Figure 19 specifies the URL of `Supervisor`. `<endpoint-identifier>` specifies a name of an end point (`name`) and its URL (`value`), e.g., when a service is deployed to be accessed via HTTP, `value` is `http://...`. `<mule-descriptor>` specifies the implementation (`implementation`) of a service (`name`), and `<inbound-router>` specifies the URL of a service by referencing an end point.

A template fragment of this transformation rule is shown below. `<<service.name>>` in the template represents the name of a UML class stereotyped with `<<service>>`. (Note that the variables and keywords in a transformation rule are embraced with << and >>.) `<<messageExchange.name>>` references the name of an UML class stereotyped with `<<messageExchange>>`. `<<requestMessage.name>>` and `<<replyMessage.`

Figure 19. Generated code for mule ESB



name>> represent the names a request and reply messages, respectively. Ark replaces each variable in a transformation template with the name of a UML model element (e.g., class name), and generates Java code. When a service transmits multiple pairs of request and reply messages, Ark generates corresponding sets of _ sendX( ) and receiveX( ). <<sinkID>> represents the logical name of a destination service. Each pair of a logical service name and its access point is specified in a deployment descriptor. For example, if a service is deployed to be accessed via HTTP, its access point starts with http://.

Table 2. Transformation between the proposed UML profile and mule ESB

Model Element in the Proposed UML Profile	Program Element in Mule ESB
<<service>> <<accessControlledService>>	A Java class with the same name.
securityTokens	A security filter implemented in Ark library
<<message>>	A Java class implementing Serializable interface
signatureMethod	A security filter implemented in Ark library
<<encryptedProperty>>	A property in a corresponding Java class
encryptionAlgorithm	A message transformer implemented in Ark library
<<messageExchange>>	Methods to send/receive messages
sink (Service's role)	Service's operations sending messages.
source (Service's role)	Service's operations receiving messages.
<<connector>>	A set of entities in a deployment descriptor
timeout	An operation's parameter to specify message's timeout period
synchrony	Different types of Mule ESB's operation used to send a message.
deliveryAssurance	A filter implemented in Ark library
queueParameters	JMS parameters specified in a deployment descriptor
encryptionAlgorithm	A message transformer implemented in Ark library
msgTransmissionLogRetained	A message transformer implemented in Ark library
routingHistoryRetained	A message transformer implemented in Ark library
messageIntegrity	A message transformer implemented in Ark library
<<messageAggregator>>	A class implementing AbstractEventAggregator in Mule ESB
<<messageConverter>>	A class implementing DefaultTransformer in Mule ESB
<<messageSplitter>>	A class implementing AbstractMessageSplitter in Mule ESB
<<logger>> <<messageFilter>> <<router>> <<validator>>	Filters provided by Mule ESB
<<multicast>>	A filter implemented in Ark library
<<manycast>>	A filter implemented in Ark library
<<anycast>>	A filter implemented in Ark library

```
public class <<service.name>> {
    private void
    _ send<<messageExchange.name>>(
  <<requestMessage.name>> request){
   MuleClient muleClient =
       new MuleClient();
    String endpointName =
    <<sinkID>>;
     UMOEndpoint url =
       MuleManager.
        getInstance().
        lookupEndpoint(
            endpointName );
    int timeout =
    <<connector.timeout>>000;
    <<IF connector.synchrony
        == Sync>>
    UMOMessage result =
        muleClient.send(
        url, message, timeout );
  <<ELSEIF connector.synchrony
          == Oneway>>
    // generating code for
    //an oneway call
    <<ELSE>>
    // generating code for
    //an asynchronous call
    <<ENDIF>>
    }

    public void
    receive<<messageExchange.name>>(
        <<replyMessage.name>>reply){
    }
}
```

UML classes stereotyped with `<<messageExchange>>` and `<<connector>>` are not transformed to particular Java classes. The message transmission/processing semantics specified in a UML model is implemented in Java classes of message sender and destination. For example, in Figure 7, a `Buyer` sends an `OrderMsg` message to a `Supervisor` synchronously. Therefore, Ark generates a Java code to send the message synchronously using Mule ESB's API[9], and embeds the code in `_sendAuthReqExchange()` of `Buyer`. `<<connector.synchrony>>` in the above transformation template references the synchrony of a connector, and Ark interprets it to generate Java code to send messages to a destination service. (`<<IF>>`, `<<ELSEIF>>` and `<<ENDIF>>` are the reserved keywords for branching statements.) Ark also generates Java code to handle timeout using Mule ESB's API (`<<connector.timeout>>` references the timeout period specified in a UML model.), and embeds the code in `_sendAuthReqExchange()` of `Buyer` (see also Figure 19).

As Figure 7 shows, a connector has the `messageIntegrity` semantics. To support this semantics, Ark provides a pair of message transformers to generate and verify a message's hash value. (These transformers are implemented as a part of Ark Library; see Figure 19.) In Mule ESB, each service can have an arbitrary number of message transformers as the classes implementing the interface `org.mule.transformer.UMOTransformer`. Message transformers are invoked (or hooked) when a service sends/receives a message. At a message source, a transformer (`edu.umb.cs.MessageIntegrityGenerator`) generates a message's hash value and embeds it into the message's header. At a message sink, a transformer (`edu.umb.cs.MessageIntegrityVerifier`) verifies the message's integrity using the hash value. Ark Library also implements the `msgTrasmissionLogRetained` and `routingHistoryRetained` semantics as message transformers `edu.cs.umb.TransmissionLogger` and `edu.cs.umb.RoutingHistoryLogger`).

When a UML model specifies a connector as a message queue, Ark generates application code that uses Java Message Service (JMS) because Mule ESB supports message queues through the use of JMS. For example, in Figure 6, `OrderConn` is specified as a message queue. Ark generates a corresponding deployment descriptor to configure

and establish JMS connector that exchanges `OrderMsg` and `OrderConfirmationMsg` between `Retailer` and `InventoryManager`.

When a UML model uses the `MessageSplitter` or `MessageAggregator` filter (e.g., Figures 14 and 15), Ark generates application code that uses corresponding class in Ark Library. Corresponding to `MessageSplitter`, Ark generates a class implementing the interface `org.mule.routing.outbound.AbstractMessageSplitter`. In Mule ESB, the implementation class can be attached to arbitrary services in order to split an outgoing message into fragments and route them to different services. When Ark transforms a UML model in Figure 14, the implementation class is attached to a `Retailer` for intercepting an `OrderMsg` message from the `Retailer` and spliting it to a `PurchasingMsg` and `AccountingMsg`.

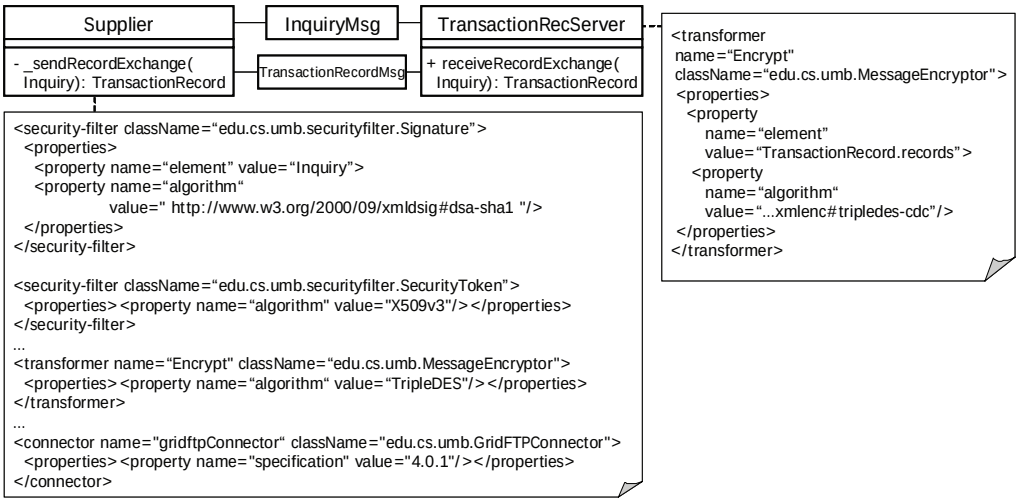
Similarly, corresponding to `MessageAggregator`, Ark generates a class implementing the interface `org.mule.routing.outbound.AbstractEventAggregator`. The implementation class can be attached to arbitrary services to aggregate an incoming message into a single message. In order to transform a UML model in Figure 15, Ark attaches the implementation class to a `Supplier` to aggregate a `OrderMsg` and a `AuthMsg` to a `AuthedOrderMsg` and pass the aggregated message to the `Supplier`.

The `Logger`, `MessageFilter`, `Router`, and `Validator` filters are transformed to and implemented with corresponding classes built in Mule ESB. Those classes are attached to services to perform message loging, filtering, routing, and validation functionalities as specified in an input UML model.

Transformation Rules for Secure and Broadband File Transfer Applications

When an application designer chooses GridFTP to operate his/her application, the application is deployed on Mule ESB and configured to use GridFTP as a message transport. Figure 20 shows some of the Java classes and deployment descriptors that Ark generates from a UML model in Figure 10.

Figure 20. Generated code for mule esb and gridFTP



As Figure 10 shows, the data field `records` is encrypted in `TransactionRecordMsg`. Since Mule ESB does not support message-level encryption, Ark Library provides a pair of message transformers to encrypt and decrypt data fields in messages (`edu.cs.umb.MessageEncryptor` and `edu.cs.umb.MessageDecryptor`). Ark generates a deployment descriptor to configure services so that they use those encryption/decryption transformers when they send/receive messages. Figure 20 shows a fragment of generated deployment descriptor for `TransactionRecServer`. It configures `TransactionRecServer` to use a message encryption transformer (`edu.cs.umb.MessageEncryption`) to encrypt the data field `records` in `TransactionRecordMsg` using Triple DES.

As Figure 10 shows, each `InquiryMsg` and `TransactionRecordMsg` message is signed with DSA, and `TransactionRecServer` performs authentication with X.509 or Kerberos. Since Mule ESB does not support DSA signatures and X.509/Kerberos security tokens, Ark Library provides a set of security filters to write/read signatures and security tokens by implementing the interface `org.mule.umo.security.UMOEndpointSecurityFilter`. Similar to message transformers, security filters are invoked when a service sends/receives a message. Ark generates a deployment descriptor that configures services to use the security filters Ark provides. Figure 20 shows a fragment of generated deployment descriptor for `Supplier`. It configures `Supplier` to include a DSA signature and an X.509 security token in each `Inquiry` message using two filters (`edu.cs.umb.securityfilter.Signature` and `edu.cs.umb.securityfilter.SecurityToken`).

In Figure 10, a `TransactionRecordMsg` is expected to contain a huge amount of data (e.g., scanned contract). When this example application uses GridFTP as a message transport to improve its throughput, Ark generates a deployment descriptor that configures `Supplier` and `TransactionRecordServer` to use GridFTP to transmit `InquiryMsg` and `TransactionRecordMsg` messages (Figure 20). Although Mule ESB does not support GridFTP, it provides a plug-in mechanism to implement arbitrary message transports. Ark Library implements a plug-in for GridFTP (`edu.cs.umb.GridFTPConnector`) so that services can use it in Mule ESB.

Extensibility of the Proposed MDD Framework

The proposed MDD framework (i.e., the proposed UML profile and Ark) is designed and implemented extensible. For example, application developers can change the default transformation rules that Ark provides. They can also integrate arbitrary implementation technologies with Ark in addition to currently-supposed three middleware (e.g., other ESBs and databases). These extensions can be made by changing the default set of transformation templates.

Moreover, the proposed MDD framework allows application developers to introduce arbitrary non-functional aspects that it does not support currently. Since the proposed UML profile is built on the UML standard metamodel with the standard extension mechanism (i.e., stereotypes and tagged-values), application developers can add new stereotypes and tagged-values representing their own non-functional aspects. This extension can be made by defining a set of transformation rules for new stereotypes and tagged-values. These newly-defined stereotypes/tagged-values and transformation rules have no effects on existing UML models and Ark itself (e.g., existing transformation rules, Ark Transformer and Ark Library).

Another type of extensibility of the proposed MDD framework is the ability to support arbitrary UML modeling tools. As described earlier, MagicDraw has been used as the default UML modeling tool; however, Ark can accept UML models from any modeling tools that serialize them in XMI. Choices of modeling tools have no effects on existing models and Ark.

EVALUATION

This section evaluates how the proposed MDD framework (i.e., the proposed UML profile and Ark) improves the reusability and maintainability of service-oriented applications. Given its two properties, the proposed MDD framework allows UML models (i.e., non-functional models built with the proposed profile) to be reusable across different implementation technologies. The first property is that the proposed UML profile allows application developers to model non-functional aspects in their applications in an implementation independent manner by abstracting away low-level details of implementation technologies (e.g., ESBs). As the second property, Ark can map a single UML model to different implementation technologies by switching transformation rules, even if those technologies are very different with each other. For example, Ark currently supports very different ESBs as implementation technologies: Mule ESB and ServiceMix ESB; their APIs and deployment descriptor schemata have no compatibility. The following code fragments are Java classes that Ark generates from the `Supervisor` class in Figure 7 to Mule ESB and ServiceMix ESB. In Mule ESB, a service can be implemented as a simple Java class.

```
public class Supervisor {

public void
receiveAuthReqExchange(
 OrderMsg reply){
  //…
 }

private void
_ sendOrderReqExchange(
OrderMsg request){
MuleClient muleClient =
 new MuleClient();
String endpointName = …
UMOEndpoint url = …
Int timeout = …
FutureMessageResult result =
 muleClient.sendAsync(
     url, request, timeout );
}}
```

On the other hand, in ServiceMix ESB, a service is implemented as a class that extends the `ComponentSupport` class and implements the `MessageExchangeListener` interface. Messages are received through the `onMessageExchange` method.

```
public class Supervisor
extends ComponentSupport
implements MessageExchangeLis
 tener {
public void
onMessageExchange(
 MessageExchange exchange)
     throws MessagingException {
     if (exchange.getRole() ==
     Role.CONSUMER) {
    ServiceEndpoint ep =
 exchange.getEndpoint();
    if (ep.getServiceName()
 .getLocalPart()
 .equals(RETAILER)) {
receiveAuthReqExchange(exchange)}}}
private void
receiveAuthReqExchange(
 MessageExchange exchange)
     throws MessagingException {
  // ...
 }

 private void
_ sendOrderReqExchange(
OrderMsg orderMsg){
InOut inout =
 createInOutExchange(
 SUPPLIER, null, null);
  NormalizedMessage msg =
 inout.createMessage();
```

Table 3. Generated program elements and their LOC

Model Elements in the Proposed UML Profile	Program Elements and their LOC in Mule ESB	Program Elements and their LOC in ServiceMix ESB
<<service>> <<accessControlledService>>	A Java class (8) An endpoint identifier in DD (1) A service entry in DD (7)	A Java class (9) A service entry in DD (6)
securityTokens	An in-bound filter in DD (3)	An in-bound filter in DD (3)
<<message>>	A Java class (2)	A Java class (2)
signatureMethod	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
<<encryptedProperty>>	An attribute in a Java class (1)	An attribute in a Java class (1)
encryptionAlgorithm	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (10)
<<messageExchange>>	In-bound and out-bound routers in DD (6)	A routing conf. in DD (14)
sink (Service's role)	A method to send in Java (10)	A method to send in Java (10)
source (Service's role)	A method to receive in Java (2)	A method to receive in Java (12)
<<connector>>	No code generated (0)	No code generated (0)
synchrony	Java code in Mule ESB API (1)	Java code in ServiceMIX API (1)
deliveryAssurance	A configuration entry in DD (3)	A configuration entry in DD (6)
queueParameters	A configuration entry in DD (14) A JMS configuration file (20)	A configuration entry in DD (6) JNDI configuration in DD (7) A JMS configuration File (20)
encryptionAlgorithm	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (10)
msgTransmissionLogRetained	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
routingHistoryRetained	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
messageIntegrity	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
<<messageAggregator>>	A Java class (4) An In-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<messageConverter>>	A Java class (4) An out-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<messageSplitter>>	A Java class (4) An out-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<logger>>	An out-bound filter in DD (3)	An out-bound filter in DD (3)
<<messageFilter>>	An out-bound filter in DD (3)	A filter conf. in DD (7)
<<router>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<validator>>	An out-bound filter in DD (3)	An out-bound filter in DD (3)
<<multicast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<manycast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<anycast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)

```
  // ...
  inout.setInMessage(msg);
  sendSync(inout);
}}
```

By making UML models (i.e., non-functional models) reusable across different implementation technologies, the proposed MDD framework allows application developers to reuse or repurpose services without knowing the details of implementation technologies.

Table 3 shows the program elements (Java code and/or deployment descriptors: DD) that Ark generates for Mule ESB and ServiceMix ESB from a single UML model element. Table 3 also shows the lines of code (LOC) of each generated program element. (LOC is shown in parentheses.) As this figure illustrates, a single model element represents multiple program elements in the proposed MDD framework. For example, `queueParamters` represents 34 LOC in Mule ESB and 33 LOC in ServiceMix ESB. This contributes to improve the maintainability of service-oriented applications by freeing application developers from manually and carefully dealing with many lower-level program elements in a consistent manner.

RELATED WORK

This article is a set of extensions to the authors' prior work (Wada, Suzuki, & Oba, 2006a, 2006b, 2006c). As one of the extensions, this work investigates new non-functional aspects for regulatory compliance, which were beyond of the scope of the prior work. Another extension is that Ark currently supports a wider range of implementation technologies. As a result, the proposed MDD framework now allows application developers to model SOA's non-functional aspects through hiding the implementation differences across two of the most major ESBs (Mule ESB and ServiceMix ESB). Given these extensions, this article fully discusses the updated details in the design and implementation of the proposed MDD framework. Moreover, unlike the prior work, this work empirically evaluates how the proposed MDD framework contributes to the reusability and maintainability of service-oriented applications.

There are several UML profiles proposed for SOA. Marcos, Castro, and Vela (2003) and Amsden, Gardner, Griffin, and Iyengar (2005) propose UML profiles to specify functional aspects in SOA. Both profiles are designed based on the XML schema of Web Service Description Language (WSDL). Each profile provides a set of stereotypes and tagged-values that correspond to the elements in WSDL, such as `Service`, `Port`, `Messages` and `Binding`[10]. Since WSDL is designed to define only functional aspects of web services, non-functional aspects are beyond of the scope of Marcos et al. (2003) and Amsden et al. (2005). Ermagan and Krüger (2007) propose and Object Management Group (2006b) standardizes UML profiles for functional aspects in SOA. Unlike the above profiles, the proposed profile focuses on specifying non-functional aspects in SOA.

Amir and Zeid (2005) propose a UML profile to describe both functional and non-functional aspects in SOA. This profile is generic enough to specify a wide range of non-functional aspects. For example, the stereotypes for non-functional aspects include `<<policy>>` and `<<permission>>`. However, their semantics tend to be ambiguous. This profile does not precisely define what non-functional aspects developers can (or are supposed to) specify and how to represent them with tagged values in accordance with given stereotypes. Ortiz and Hernández (2006) also propose a generic UML profile to describe various non-functional aspects (called extra-functional properties). Arbitrary non-functional aspects can be defined as stereotypes extending the `<<Extra-Functional Property>>` stereotype. However, it is ambiguous how to define particular non-func-

tional aspects with user-defined stereotypes and tagged-values. The World Wide Web Consortium (2006) standardizes the WS-Policy specification, a generic XML format to specify arbitrary non-functional aspects of web services. No explicit principles and guidelines are available on how to define particular non-functional aspects with XML document elements. Unlike the above three schemes, the proposed UML profile carefully and precisely defines a variety of stereotypes and tagged-values for non-functional aspects in SOA so that the proposed MDD tool (Ark) can interpret and transform models to code in an unambiguous manner.

Vokäc (2005) proposes a UML profile for data integration in SOA. It provides data structures to specify messages. Application developers can use the data structures for building dictionaries that maintain message data used in existing systems and new applications. This profile separates data integration as a non-functional aspect from functional aspects, and enables specifying data integration in an implementation independent manner. This UML profile and the proposed profile focus on different issues in SOA. Data integration is beyond of the scope of the proposed profile, and Vokäc (2005) does not consider non-functional aspects in message transmission, message processing, security and service deployment.

Heckel, Lohmann, and Thöne (2003) propose a UML profile for dynamic service discovery in SOA. This profile provides a set of stereotypes (e.g., `<<uses>>`, `<<requires>>` and `<<satisfies>>`) to specify relationships among service interfaces, service implementations and functional requirements. For examples, a relationship can specify that a service *uses* other services, and another relationship can specify that a service *requires* other services that *satisfy* certain functional requirements. These relationships are intended to aid dynamic discovery of services. Rather than service discovery, the proposed UML profile focuses on non-functional semantics in message transmission, message processing, security, and service deployment.

Object Management Group (2007) standardizes a UML profile for Data Distribution Service (DDS). DDS is a standard specification for publish/subscribe middleware, and it supports several non-functional aspects in real-time messaging. OMG's UML profile for DDS allows UML models to specify these non-functional aspects. In contrast, the proposed profile is not limited to real-time messaging, but supports a wider range of non-functional aspects. Moreover, OMG's profile is designed to be mapped into only DDS implementations. In contrast, the proposed profile is designed in an implementation independent manner; it can be mapped to arbitrary implementation technologies.

Gardner (2003), List and Korherr (2005), Johnston (2004) and Object Management Group (2005a) define UML profiles to specify service orchestration and map it to Business Process Execution Language (BPEL) (Organization for the Advancement of Structured Information Standards, 2003). These profiles provide a limited support of non-functional aspects in message transmission, such as messaging synchrony. The proposed profile does not focus on service orchestration, but a comprehensive support of non-functional aspects in message transmission, message processing, security and service deployment.

Lodderstedt, Basin, and Doser (2002) propose a UML profile, called SecureUML, to define role-based access control for network applications. SecureUML provides stereotypes to assign roles (`<<security.role>>`) and access control permissions (`<<security.constraint>>`) to applications (e.g., UML interfaces and classes). SecureUML uses Object Constraint Language (OCL) to define access control. Jürjens (2002) propose another UML profile, called UMLsec, to define data encryption (`<<data security>>`) and secure network links (`<<encrypted>>`). Wang and Lee (2005) and

Nakamura, Tatsubori, Imamura, and Ono (2005) also propose UML profiles to define security aspects. In addition to security aspects, Soler, Villarroel, Trujillo, Medina, and Piattini (2006) propose a UML profile extending the Common Warehouse Metamodel (Object Management Group, 2003) in order to define regulatory audit policies in data warehouses. For example, the profile provides stereotypes to specify whether a data warehouse retains logs to access data sources. Gönczy and Varró (2006) propose a formal definition of reliable messaging mechanisms as a metamodel. These profiles/metamodels are parallel to the proposed UML profile in terms of the ability to describe security aspects, audit policies, and reliable messaging in network applications. However, the proposed UML profile covers not only security, auditing or reliable messaging aspects but also many other non-functional aspects in SOA (e.g., message queuing, message validation/filtering, and message.

Zhu and Gorton (2007) and Zou and Pavlovski (2006) propose UML profiles to visually define non-functional requirements such as desirable response time and throughput. However, they do not consider model transformation to map non-functional requirements to certain implementation technologies. In contrast, the proposed profile is designed to consider model transformation, although non-functional requirements are beyond of the scope of the proposed profile.

There are several specifications and research efforts to investigate implementation techniques for non-functional aspects in SOA (Baligand & Monfort, 2004; Mukhi, Konuru, & Curbera, 2004; Organization for the Advancement of Structured Information Standards, 2003, 2004a, 2004b; Wang, Chen, Wang, Fung, & Uczekaj, 2004). Each specification and technique provides a means to implement non-functional requirements in, for example, performance, reliability, and security and to enforce services to follow the requirements. Rather than investigating specific implementations of non-functional aspects in SOA, the proposed MDD framework is intended to provide a means for application developers to model and maintain non-functional aspects in an implementation independent manner so that they can be mapped on different specifications or implementation technologies.

CONCLUSION

This article proposes a model-driven development (MDD) framework for non-functional aspects in SOA. The proposed MDD framework consists of (1) a UML profile to graphically specify and maintain SOA non-functional aspects in an implementation independent manner and (2) an MDD tool that accepts a UML model defined with the proposed profile and transforms it to application code (program code and deployment descriptors). This article presents design details of the proposed UML profile and describes how the proposed MDD tool uses the profile to develop service-oriented applications that can run with different implementation technologies such as Mule ESB, ServiceMix and GridFTP. Empirical evaluation results show that the proposed MDD framework contributes to improve the reusability and maintainability of service-oriented applications by hiding the details of implementation technologies.

Several extensions to the proposed MDD framework are planned as future work. As described in the Related Work section, there are several other UML profiles for SOA. The proposed profile will be co-used or integrated with some of them (e.g., Oba, Hashimoto, Fujikura, & Munehira, 2005; Object Management Group, 2005b) in order to investigate a more comprehensive development framework for SOA.

Another extension is to integrate the proposed UML profile with a modeling language for business processes such as Business Process

Modeling Notation (Object Management Group, 2006a). The proposed profile is designed to specify applications from a structural point of view; it does not consider a viewpoint of processes or workflows. Therefore, as the size of a model (application) increases, it becomes harder to understand how messages are exchanged among services and define non-functional aspects along with message flows. For example, in order to specify secure messaging for a certain business process (e.g., order processing process), it can be time-consuming and error-prone to find all the services associated with the process and define a security aspect for the connections among those services. The integration with a business process modeling language can make non-functional modeling more intuitive by providing both structural and process viewpoints.

The proposed MDD framework will be evaluated in several different application domains[11]. One of them is service-oriented system integration in a natural gas utility company. The proposed UML profile and Ark are planned to be used in a system integration project, and their design and implementation will be enhanced through the project experience. Another application domain is eco-informatics. The proposed framework has been used to design and maintain ecological observation systems (Wada & Suzuki, 2006d). Ecological observation systems monitor ecosystems, record various observation data (e.g., a niche of a particular species and weather in the niche), and help ecologists understand and predict the observation of ecosystems. Currently, ecological observation systems are often implemented monolithic; their extensibility and customizability are limited. SOA is expected to overcome this issue by decomposing an observation system into multiple services, implementing the system as a combination of services, and extending/customizing it through a recombination of services (Bermudez, Bogden, Bridger, Creager, Forrest, & Graybeal, 2006). The proposed MDD framework has been used to separate functional and non-functional aspects in an ecological observation system and model/implement non-functional aspects in the system. Through this practice, the proposed MDD framework will be enhanced to improve its generality.

ACKNOWLEDGMENT

This work is supported in part by OGIS International, Inc.

REFERENCES

Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., & Foster, I. (2005). The Globus striped GridFTP framework and server. *ACM Super Computing.*

Amir, R., & Zeid, A. (2004). A UML profile for service oriented architectures. *ACM Object-Oriented Programming, Systems, Languages, and Applications Poster session.*

Amsden, J., Gardner, T., Griffin C., & Iyengar, S. (2005). UML 2.0 profile for software services. IBM developerWorks.

Arsanjani A., Zhang L., Ellis M., Allam A., & Channabasavaiah K. (2007) S3: A service-oriented reference architecture. *IT Professional, 9*(3).

Baligand, F., & Monfort, V. (2004). A concrete solution for web services adaptability using policies and aspects. *Proceedings of UNITN/ Springer International Conference on Service Oriented Computing.*

Bermudez, L., Bogden, P., Bridger, E., Creager, G., Forrest, D., & Graybeal, J. (2006). Toward an ocean observing system of systems. *MTS/IEEE Oceans.*

Bichler, M., & Lin, K. (2006). Service-oriented computing. *IEEE Computer, 39*(6).

Bieberstein N., Bose S., Fiammante M., Jones K., & Shah R. (2005). *Service-oriented architecture (SOA) compass: Business value, planning, and enterprise roadmap.* IBM Press.

Chappell, D. (2004). *Enterprise service bus.* O'Reilly.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., et al., (2004). *Patterns: Service-oriented architecture and Web services.* IBM Red Books.

Ermagan, V., & Krüger, H. (2007). A UML2 profile for service modeling. *Proceedings of ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.*

Foster, I. (2005). Service-oriented computing. *Science, 308*(5723).

Fuentes, L., & Vallecillo, A. (2004). An introduction to UML profiles. *The European Journal for the Informatics Professional, 5*(2).

Gardner, T. (2003). UML modeling of automated business processes with a mapping to BPEL4WS. *Proceedings of European Conference on Object-Oriented Programming Workshop on Object Orientation and Web Services.*

Gönczy, L., & Varró, D. (2006). Modeling of reliable messaging in service oriented architecture. *Proceedings of Telcert International Workshop on Web Services Modeling and Testing.*

Heckel, R., Lohmann, M., & Thöne, S. (2003). Towards a UML profile for service-oriented architectures. *Proceedings of Workshop on Model Driven Architecture: Foundations and Applications.*

Java Community Process. (2001). UML profile for Enterprise Java Beans.

Johnston, S. (2004). UML 1.4 profile for software services with a mapping to BPEL 1.0. *IBM developerWorks.*

Jürjens, J. (2002). UMLsec: Extending UML for secure systems development. *Proceedings of ACM/IEEE International Conference on Unified Modeling Language.*

Lewis, G., Morris, E., Brien, L., Smith, D., & Wrage, L. (2005). *Smart: The service-oriented migration and reuse technique.* Technical report, Software Engineering Institute, Carnegie Mellon University.

List, B., & Korherr, B. (2005). A UML 2 profile for business process modelling. *Proceedings of ACM International Conference on Conceptual Modeling Workshop on Best Practices of UML at the International Conference on Conceptual Modeling.*

Lodderstedt, T., Basin, D., & Doser, J. (2002). SecureUML: A UML-based modeling language for model-driven security. *Proceedings of ACM/IEEE International Conference on Unified Modeling Language.*

Marcos, E., Castro, V., & Vela, B. (2003). Representing Web services with UML: A case study. *Proceedings of UNITN/Springer International Conference on Service Oriented Computing.*

Mukhi, N., Konuru, R., & Curbera, F. (2004). Cooperative middleware specialization for service oriented architectures. *Proceedings of ACM International World Wide Web Conference.*

Nakamura, Y., Tatsubori, M., Imamura, T., & Ono, K. (2005). Model-driven security based on a Web services security architecture. *Proceedings of IEEE International Conference on Services Computing.*

Oba, K., Hashimoto, M., Fujikura, S., & Munehira, T. (2005). The status quo and challenges of service-oriented architecture based application

design. *Proceedings of IPSJ Workshop on Software Engineering.*

Object Management Group. (2003). Common warehouse metamodel, version 1.1.

Object Management Group. (2004). UML 2.0 super structure specification.

Object Management Group. (2005a). Business process definition metamodel.

Object Management Group. (2005b). UML profile for modeling quality of service and fault tolerance characteristics and mechanisms.

Object Management Group. (2006a). Business process modeling notation, version 1.0.

Object Management Group. (2006b). UML profile and metamodel for services, request for proposal.

Object Management Group. (2006c) UML profile for data distribution service, request for proposal.

Object Management Group. (2007) Data Distribution Service for Real-time Systems, version 1.2.

O'Grady, S. (2004). *SOA meets compliance: compliance oriented architecture*. White paper, RedMonk.

Organization for the Advancement of Structured Information Standards. (2003). Web services business process execution language.

Organization for the Advancement of Structured Information Standards. (2004a). Web service reliability 1.1.

Organization for the Advancement of Structured Information Standards. (2004b). Web service reliable messaging.

Organization for the Advancement of Structured Information Standards. (2005). Web services security policy language.

Ortiz, G., & Hernández, J. (2006). Toward UML profiles for Web services and their extra-functional properties. *Proceedings of IEEE International Conference on Web Services.*

Papazoglou, M. (2003). Service-oriented computing: concepts, characteristics and directions. *Proceedings of IEEE International Conference on Web Information Systems Engineering.*

Roberts, D., & Johnson, R. (1997). *Evolving frameworks: A pattern language for developing object-oriented frameworks.* Pattern Languages of Program Design 3, Chapter 26. Addison Wesley.

Soler, E., Villarroel, R., Trujillo, J., Medina, E., & Piattini, M. (2006). Representing security and audit rules for data warehouses at the logical level by using the common warehouse metamodel. *Proceedings of IEEE International Conference on Availability, Reliability and Security.*

Vinoski, S. (2003). Integration with Web services. *IEEE Internet Computing, 7*(6).

Vokäc, M. (2005). Using a domain-specific language and custom tools to model a multi-tier service-oriented application: Experiences and challenges. *Proceedings of ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.*

Wada, H. & Suzuki, J. (2006). *Designing ecological observation systems using service oriented architecture*. Elsevier/ISEI International Conference on Ecological Informatics, poster paper.

Wada, H., Suzuki, J., & Oba K. (2006a). A model-driven development framework for non-functional aspects in service oriented grids. *Proceedings of IEEE International Conference on Autonomic and Autonomous Systems.*

Wada, H., Suzuki, J., & Oba K. (2006b). Modeling non-functional aspects in service oriented

architecture. *Proceedings of IEEE International Conference on Services Computing.*.

Wada, H., Suzuki, J., & Oba K. (2006c). A service-oriented design framework for secure network applications. *Proceedings of IEEE International Conference on Computer Software and Applications Conference.*

Wang, G., Chen, A., Wang, C., Fung, C., & Uczekaj, S. (2004). Integrated quality of service (QoS) management in service-oriented enterprise architectures. *Proceedings of IEEE Enterprise Distributed Object Computing Conference*.

Wang, L., & Lee, L. (2005). *UML-based modeling of Web services security.* IEEE European Conference on Web Services, poster paper.

World Wide Web Consortium. (2002a). XML encryption syntax and processing.

World Wide Web Consortium. (2002b). XML signature syntax and processing.

World Wide Web Consortium. (2006). Web services policy framework.

Zhang, Z., & Yang, H. (2004). Incubating services in legacy systems for architectural migration. *Proceedings of IPSJ/IEEE Asia-Pacific Software Engineering Conference.*

Zhu, L., & Gorton, I. (2007). UML profiles for design decisions and non-functional requirements. *Proceedings of ACM/IEEE International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent.*

Zou, J., & Pavlovski, C. (2006). Modeling architectural non functional requirements: From use case to control case. *Proceedings of IEEE International Conference on e-Business Engineering.*

ENDNOTES

1. According to the UML specification, the first letter of a stereotype's name is capitalized when the stereotype is defined (Figure 3). However, it is not capitalized when the stereotype is used in UML models (Figure 2).
2. Precisely, a composite class can contain any *classifiers*, defined in the UML metamodel.
3. http://www.gtin.info/
4. http://mule.mulesource.org/
5. http://servicemix.apache.org/
6. An extension to FTP for transmitting files of large size (Allcock, Bresnahan, Kettimuthu, Link, Dumitrescu, Raicu, & Foster, 2005)
7. http://www.magicdraw.com/
8. http://www.openarchitectureware.org/
9. Mule ESB provides three different APIs to send messages in synchronous, asynchronous and oneway (non-blocking) manners.
10. In WSDL, a `Service` defines an interface of a web service. A `Port` specifies an operation in a `Service`, and `Messages` defines parameters for a `Port`. A `Binding` specifies communication protocols used by `Port`s.
11. A software engineering discipline suggests investigating at least three applications on a framework in order to examine the framework's generality and reusability. (Roberts & Johnson, 1997)

This work was previously published in the International Journal of Web Services Research, edited by L.J. Zhang, Volume 5, Issue 4, pp. 1-31, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 2.24
An Incremental Functionality–Oriented Free Software Development Methodology

Oswaldo Terán
ENDITEL; Centro de Micro Electrónica y Sistemas Distribuidos and Centro de Simulación y Modelos, Universidad de los Andes, Venezuela

Johanna Alvarez
CENDITEL, Venezuela

Blanca Abraham
CEMISID Universidad de los Andes, Venezuela

Jose Aguilar
CENDITEL; Centro de Micro Electrónica y Sistemas Distribuidos, Universidad de los Andes, Venezuela

ABSTRACT

This chapter presents a methodology used as reference model for a free software factory that is part of the National Centre for Free Technologies in Venezuela. This centre is oriented at promoting free software development for serving mostly the public sector in order to promote endogenous development and technologic autonomy. Under this strategy, strengthening the software small and medium size enterprises and cooperatives, by allowing them to participate in different projects (improving their know-how) and providing them with a methodology for increasing their capabilities and software quality, is necessary and urgent. This methodology plans the development process incrementally, based on a prioritisation of the software functionalities development in accordance to the functionalities risks, development urgency, and dependencies. It combines aspects of the two styles of free software development,

namely cathedral and bazaar. The development process is centralised, in essence collaborative, and continuously allows source code release.

INTRODUCTION

The Free Software Factory (FSF) of CENDITEL (Venezuelan national centre for promoting free technologies) has been conceived and created as part of the efforts of the Venezuelan State aiming at increasing endogenous development and technological sovereignty. In particular, it intends to strengthen the national software sector, especially the small and medium software enterprises (including the cooperatives), by allowing them to access the technology and participate in the software market, on one hand, and to increase their capabilities and software quality, on the other hand.

Two styles exist for developing free software: the cathedral style and the bazaar style. In the cathedral mode, software is developed from a unified *a priori* project that prescribes all the functions and the features to be incorporated in the final product. Programmers' work is centrally coordinated and supervised in order to assure the integration of various components. On the other hand, in the bazaar style, software emerges from an unstructured evolutionary process. Starting from a minimal code, groups of programmers add features and introduce modifications and patches to the code. There is no central allocation of different tasks; developers are free to develop a given program in directions they favor.

This chapter presents an attempt at building a free software development methodology having many characteristics of the cathedral style but keeping certain principle of the bazaar mode. The methodology has been developed at a public organisation which responds to public sector free software necessities and requirements that must be satisfied in a limited time period. Because of this, it is necessary to adopt the cathedral mode of work while taking key advantages of the bazaar style. For instance, it is allowed that developers from outside the organisation contributes with software coding, testing, and so forth; these external developers do not follow a centrally controlled process; and the software code is made public as soon as it is tested.

This methodology assumes an organisational structure oriented towards specific processes. The processes dedicated to software development are:

- *Process # 1*: Free Software Project Management
- *Process # 2*: Specific Project Administration
- *Process # 3*: Free Software Application Development

Actions to be carried out in these processes are classified in steps and activities. In particular, steps and activities in the third process are implemented by the following six phases. This methodology has taken ideas from diverse software development methodologies, methods, and models such as the extreme programming method (Beck, 2004), the rational unified process (Kruchten, 2000; Pollice, 2001; Probasco, 2000), the watch method (Montilva, 2004; Montilva, Hamzan, & Ghatrawi, 2000), and the model of processes for software development (MoProSoft) (Oktaba et al., 2005). Due to the fact that these models and methods, except extreme programming, have been proposed proprietary software development, it has been necessary to adapt the hints, ideas, or procedures taken from them to the free software development needs.

The methodology to be proposed has been validated at the FSF of the Foundation for Science and Technology of the Mérida State in Venezuela (FUNDACITE-Mérida). This factory has permitted us to understand better, empirically, the real needs of a free software development process and has also been a source of interesting ideas. The

proposed structure will allow planning and control activities which are required in the management and administration of software projects. In addition, the free software application development process is iterative and incremental, in terms of the software application functionalities. On the other hand, the design of the application is based on component architecture, which allows software reuse. Each process will be explained in detail in the main body of this chapter. For facilitating each process, some free software tools will be suggested.

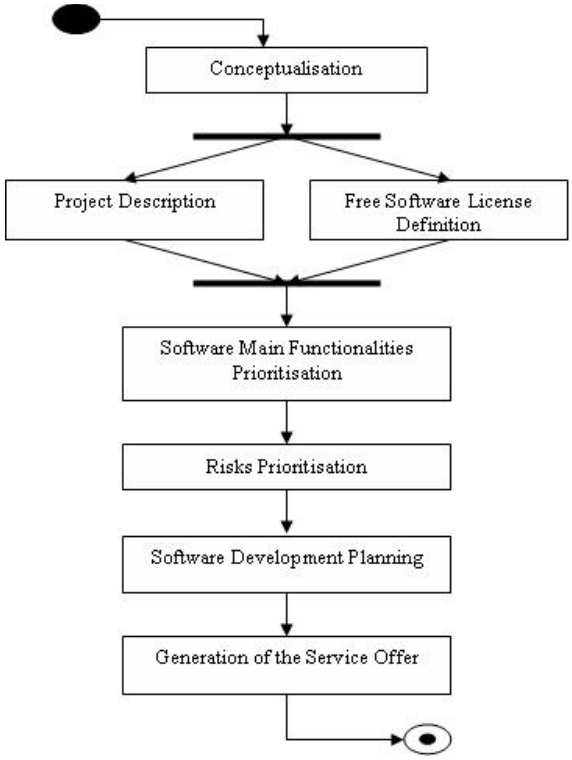
FREE SOFTWARE DEVELOPMENT METHODOLOGY

Process # 1: Free Software Project Management (FSPM)

This process is responsible for managing all projects being carried out by the organisation, that is, both internal projects (projects for the organisation) and external projects (projects for other organisations) are managed. Specifically, in this process, a "service offer" for the project to be developed is generated. This offer must include a conceptualisation, a description, and a (general) development plan for the project, as well as the definition of the free software license. Figure 1 shows the steps as a workflow for the FSMP process. Subsequently, these steps will be described.

a. **Conceptualisation**
   Description: Specific user needs and/or problems must be identified in order to define the scope of the project. In case of a high complexity of the project, for instance, the scope of the problem goes beyond a software need and involves organisational issues, methodologies such as technologic prospective and strategic planning, and/or tools such as fish bone (a technique commonly used in operation research and in quality control) are recommended. In this case, the study would suggest a set of solutions and organisational changes, of which software needs would be only part of the whole answer.
   People Responsible: According to free software philosophy, all people associated with the project (project manager, project administrator, developers, users, and so forth) must be involved from the very early stages of it, even when new actors can be incorporated to any phase of the project.
   Techniques: Prospective analysis, strategic planning, or any other useful technique.
   Products: (1) Client/user's needs and/or problems; (2) scope of the project.

Figure 1. Free software project management process



b. **Project Description**
   The main point in this phase is to achieve a detailed description of the project. Each actor must analyse the client's need and problem,

as well as the project's scope, in order to contribute to the description of the project.
People Responsible: Clients, users, project manager, project administrator, developers, assessors and other people interested in the project participation.
Products: Project description document.

c. **Free Software License Definition**
Description: In this step, the free software license to be adopted for the development is defined. It might be the case that a license from the market satisfies the client requirements and then is chosen or, in case that there is not an existent license matching the user's requirements, a new license is defined.
People Responsible: Clients, users, project manager, project administrator, assessors.
Products: Project license.

d. **Software Main Functionalities Prioritisation**
Description: In this step, the goal is to describe and classify the software functionalities in accordance with the implementation urgency required by the client.
People Responsible: Clients.
Products: Functionalities prioritised.

e. **Risk Prioritisation**
Description: To identify, prioritise, and associate the risks to software application functionalities. The risks are prioritised in accordance to their impact on the application development.
People Responsible: Clients, users, project manager, project administrator, developers, assessors, and other people interested in the project.
Products: Risks prioritised.

f. **Software Development Plan**
Description: To build the development plan, the implementation order of the functionalities of the application must be established, in accordance to the functionality priorities defined by the client, and the risks prioritised associated to the functionalities. This must allow determining the number of cycles or iterations required for the development of the application. A cycle is responsible for developing a certain number of functionalities (taken into account their priority order). A development plan can be modified after an iteration, as a result of work reorganisation, in line with the dynamic of the project.
People Responsible: Project manager, project administrator, and developers.
Products: Development plan.

g. **Generation of the Service Offer**
Description: The service offer is completed in this step. It specifies all important issues of the project, such as the goal, scope, and description of the project; the development plan; the due dates for deliverables; the work team; the project cost; and the operation platform.
People Responsible: Project manager, clients.
Techniques and Tools: Service offer forms.

Note: In accordance to the chosen software license, the products achieved in this process must be published in a collaborative development platform. This will facilitate the interested people access to the software products and their documentation.

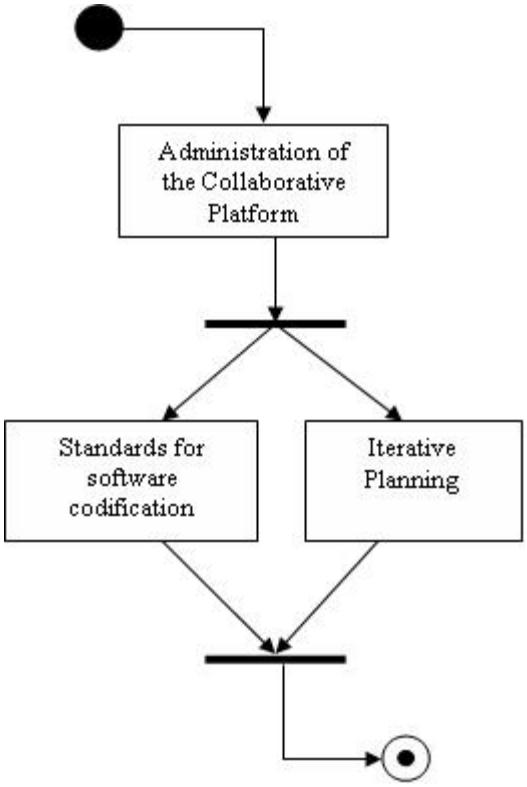
Process #2: Administration of Specific Projects (ASP)

The administration of specific projects leads the developer team of a software application (it is assumed that a software application development corresponds to a software project). In this sense, each software project must have at least one project administrator. The project administrator is responsible for organising and planning the activities corresponding to each iteration defined in the development plan. Additionally, the project administrator must assure the software quality, manage the system configuration, and the collaborative technical platform, as well as supervise

and control the project development and the administration of subcontracts. Figure 2 shows the main steps as a workflow for the ASP process. Subsequently, these steps will be described.

a. **Administration of the Development Collaborative Platform**
Description: The processes related to the ASP phase are facilitated by using a collaborative platform. A software developing team has its own necessities; accordingly it is important to select the correct tool for collaborative software management considering such necessities. In this phase, the collaborative platform is set up. The collaborative platform permits any interested person to collaborate with and share ideas, source code, documentation, testing, and so forth. This is a very important aspect of the free software development. However, as part of the administration of this platform, the project administrator must approve and then publish the software versions and the associated documentation in accordance with the free software license assumed.
People Responsible: Project administrator.
Tools: GFORCE, and so forth.
Products: Collaborative development server.

Figure 2. Process for the Administration of Specific Projects



b. **Standard for Software Codification**
Description: This step establishes the standards for code generation and for the documentation to be used during the software development. These standards allow a quick and simple reading of the code, facilitating the work of the whole group, including the client, the user, and other actors.
People Responsible: Project administrator.
Products: Coding and documentation standards.

c. **Iterative Planning**
Description: The activities of the iteration to be carried out in accordance with the functionalities to be developed are planned. After an iteration is performed, the next iteration is planned and takes into account functionalities that could not be implemented and problems found during the previous iteration.
People Responsible: Project administrator.
Techniques: Gantt, Pert/CPM
Tools: Planner, GFORCE, XPTracker, SourceForge, and so forth.
Products: Plan for the iteration to be developed.

Note: Products accomplished in this process must be placed at the collaborative platform in accordance with the adopted free software license.

Process #3: Free Software Application Development

The software application is constructed by the sequence of iterations or cycles in an incremental and iterative way, allowing that users and clients can check the advances of the work and give feedback useful for improving the development and testing processes. The methodology presents a general reference framework or structure for the activities to be planned at each iteration of this process (see Figure 3). In each cycle, one activity receives the main attention while the others are secondary. The whole set of functionalities is developed during the cycles.

Any person can access and execute the source code stored in the collaborative platform. In this way, everyone can contribute to the project. Experiences show that the more people use and test the software, the more quickly the errors and bugs will be found and solved.

It is important to mention that during the software application construction, not only must the code be published but also all associated documentation. In this manner, new programmers and collaborators can be easily incorporated. As mentioned earlier, the code and associated documentation publication depends on the software license established. Next, in Figure 3, the development phases carried out during this process will be presented.

a. **Application Domain Analysis**
Description: This phase is considered one of the most important in the software development process, since the domain environment and context where the application will operate are analysed and understood. Such analysis is carried out in the first iteration but can be upgraded in the subsequent iterations. The activities workflow for this phase is shown in Figure 4. Following this figure, the main activities will be described.

- ***Domain Description***
Description: Establishes and validates the application domain and its organisational scope.
People Responsible: System analyst internal to the developer organisation, users, clients, programming team. In this chapter, the phrase "*internal to the organization*" means a person who works for the organization that develops the software, as opposed to "external

Figure 3. Free software development process

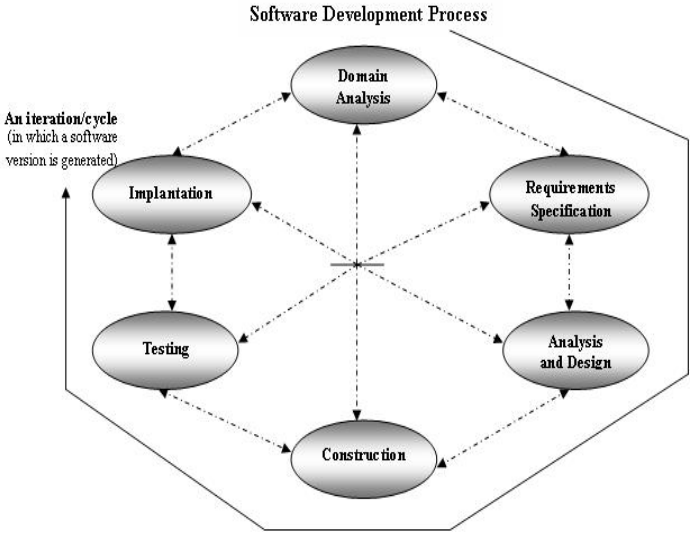
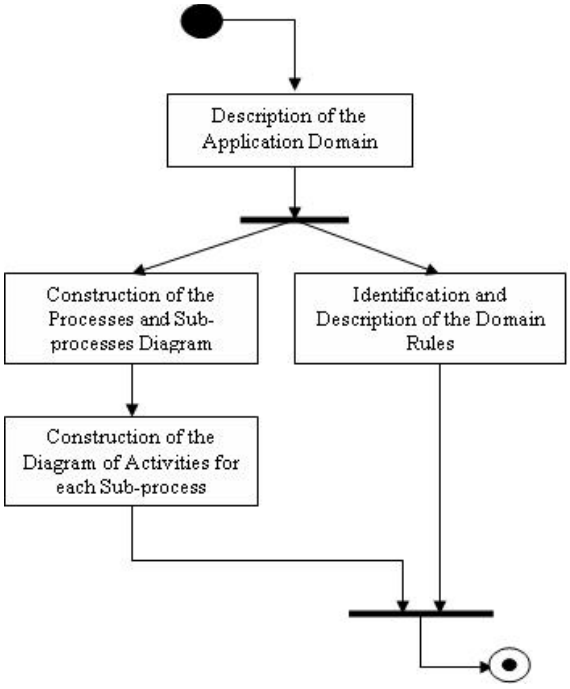


Figure 4. Steps for the application domain analysis phase



to the organization," which means a person who is not actually working for the developer organization.

Techniques: Domain engineering, interviews, revision of documents, and bibliography.

Products: Domain application definition.

- ***Construction of the Processes and Subprocesses Diagram***

  Description: This activity must identify processes and subprocesses related to the application domain, as well as events associated to all these, in order to generate the domain processes and subprocesses diagram. Finally, this diagram must be validated.

  People Responsible: System analyst (internal to the organisation), users, clients.

  Techniques: The processes diagram given by UML.

  Tools: Umbrello, ArgoUML, caseUML.

  Products: Domain processes and subprocess diagram.

- ***Construction of the Diagram of Activities for Each Subprocess***

  Description: Generates and validates the activities diagram for each subprocess.

  People Responsible: System analyst (internal to the organisation), users, clients.

  Techniques: Activities diagram offered by UML.

  Tools: Umbrello, ArgoUML, caseUML.

  *Products: S*ubprocess activity diagrams.

- ***Identification and Description of the Domain Rules***

  Description: Domain rules regulating the application domain must be identified and studied.

  People Responsible: System analyst (internal to the organisation), users, clients.

  Techniques: the activities diagram offered by UML.

  Tools: Umbrello, ArgoUML, caseUML.

*Products: S*ubprocess activity diagrams.

b. **Requirements Specification**

Description: In this phase, the functionalities to be developed in the planned iteration are specified, and the nonfunctional requirements are defined or upgraded. Generally, the nonfunctional requirements are defined in the early iterations. The requirement specification document will be upgraded from iteration to iteration. It is important to notice that in this phase the user or the client can modify, change, include, or eliminate requirements and risks, which, in turn, might entail updates of the development plan. The activities workflow

for this phase is shown in Figure 5. Following this figure, the main activities for this phase will be described.

- ***Description of Requirements Related to the Actual Iteration***

  Description: A detailed description of the functional requirements for the present iteration is generated, and the nonfunctional requirements are defined or upgraded. These will allow generating and validating the requirements definition document. It is important to mention that only the definition of such requirements related to the present iteration are validated, since those requirements related to previous iterations were validated during the corresponding iterations.

  People Responsible: System analyst (internal to the organisation), users, clients.

- ***Specification of Requirements Related to the Actual Iteration:***

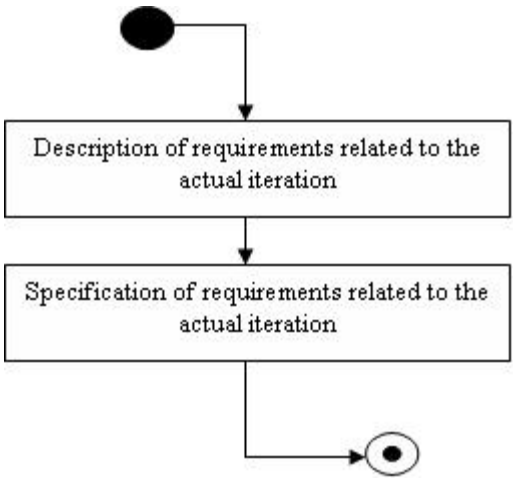
  Description: To create or upgrade the requirements specification document, including the use cases describing the functional requirements associated with the actual iteration. In this methodology, it is understood that the requirements specification is made in terms of diagrams and textual descriptions of the use cases. Only the specification of those requirements associated to the present or actual iteration must be verified and validated in this step.

  People Responsible: System analyst (internal and/or external to the organisation), users, clients.

  Techniques: the use cases diagram offered by UML.

  Tools: Umbrello, ArgoUML, caseUML.

Figure 5. Steps for the requirements specification



c. Analysis and Design

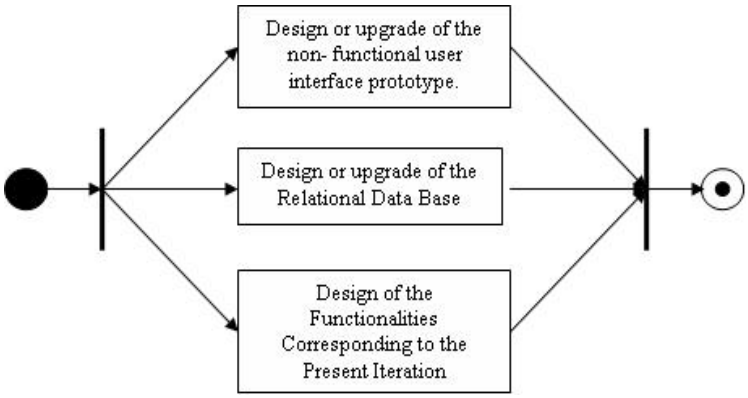
Description: In this phase, the specification of requirements is translated into a design specification, based on a set of architectonic views, which represent the system architecture. In this phase also, the user interfaces and databases are designed. The application architecture, like the requirements, is enriched or upgraded as the subsequent iterations are carried out, since each iteration add functionalities to the software application been developed. All this gives flexibility to the design, permitting that change in the client's viewpoint about desired functionalities be taken into account without great difficulties. The activities workflow for this phase is shown in Figure 6. Afterwards, the main activities are going to be described.

- ***Design or Upgrade of the Nonfunctional User Interface Prototype***

  Description: Create or update the nonfunctional user interface prototype. This design includes the hierarchic diagram of windows, taking into account the user requirements. This design must be validated.

  People Responsible: System analyst (internal or external to the developer organisation), programmers, users, clients.

Figure 6. Steps for the analysis and design phase



Tools: UX, DIA.
Products: Design of the nonfunctional user interface prototype.

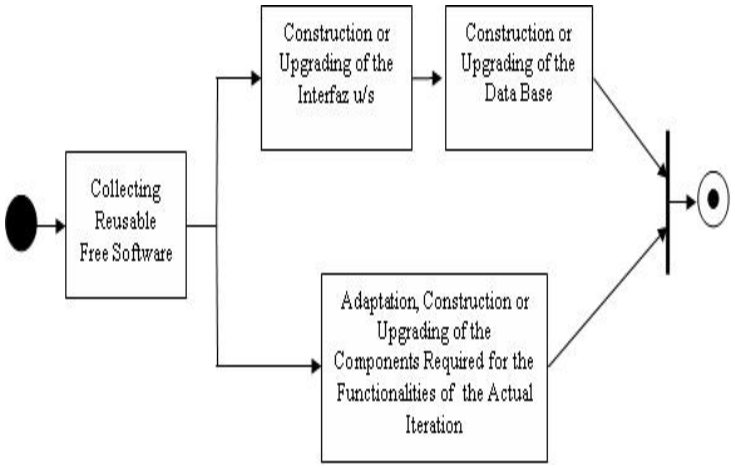
- ***Design or Upgrade of the Relational Database:***
  Description: The database design document is generated or upgraded. This document must contain the diagram entity/relation and the relational scheme, for the actual iteration. For these diagrams, the entities of the database and their attributes, as well as the primary and the foreign keys, must be defined. The entities of the database are identified by using the use cases for the present iteration. Finally, the database administrative procedures (i.e., backup, security, recovery, etc.) must be defined, and the database design document must be validated.
  People Responsible: System analyst (internal and external to the organisation), programmers.
  Techniques: Normalisation formulas.
  Products: Entity relation diagrams.
- ***Design of the Functionalities Corresponding to the Present Iteration***
  Description: The architectonic views must be generated or upgraded. It is constituted by the logic, the implementation, the behaviour, and the conceptual views. The logic view is defined by the class diagrams of the software application. It is created or upgraded by: (a) deriving from the use cases (associated to the actual iteration) the objects of the application, (b) generating the sequence diagrams for the "methods" or functions involved in the realisation of the use cases for the actual iteration. The implementation view is generated or upgraded from the components diagrams. The behaviour view is created or upgraded from the interaction relations among the components. The conceptual view is defined or upgraded from the use case diagrams corresponding to the actual iteration.
  People Responsible: System analyst (internal or external to the organisation).
  Techniques: Class, components, and interaction diagrams
  Tools: Umbrello, ArgoUML, caseUML.
  Products: Architectonic view of the software application.

d. **Construction**
Description: For the actual iteration or cycle, the user interface, the database, and the functionalities of the application are constructed or upgraded in this phase. For that,

the software application source code for the actual version is developed. The activities workflow for this phase is shown in Figure 7. Afterwards, the main activities of this phase will be described.

- ***Collecting Reusable Free Software*:**
  Description: Reusable free software components, abstract data type, classes, functions, and whole systems, useful for the software application, are searched for and collected.
  People Responsible: Programmers (internal and/or external to the organisation). This is the first activity where external programmers participate in the software development.
  Tools: Some are available at Web sites such as www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.
- ***Construction or Upgrading of the User Interface U/S***
  Description: The reusable user interface components corresponding to the design of the interface associated to the actual iteration are adapted and, when required, those of the previous iterations are upgraded.
  People Responsible: Programmers (internal and/or external to the organisation).
- ***Construction or Upgrading of the Database***
  Description: Build or upgrade the database using information from the database design document for the actual iteration. Additionally, components of the user interface must be integrated along the database.
  People Responsible: Programmers (internal and/or external to the organisation).
  Techniques and Tools: Web sites like www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.
- ***Adaptation, Construction, or Upgrading of the Components Required for the Functionalities of the Actual Iteration***
  Description: For the components, abstract data types, classes, and functions requited for the functionalities of the actual iteration: (a) adapt those reusable already collected, (b) construct those that could not be found, (c) update those useful from previous iterations.

Figure 7. Steps for the construction phase



People Responsible: Programmers (internal and/or external to the organisation).

Tools: Some are available at Web sites such as www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.

e. **Testing**

Description: In this phase, the unitary, integration, functional, and nonfunctional tests, for the components corresponding to the functionalities associated to the actual iteration, are designed or upgraded, and applied. The nonfunctional tests are applied only in the last version of the software application, which is obtained in the last iteration. The installation tests are also designed in this phase, but are applied in the implantation phase. It is important to say that code modified by developers external to the developer organisation must also be appropriately tested. Only after these tests are successfully completed can the project administrator publish the code. Figure 8 shows the workflow for this phase. Since this figure sufficiently explains each step, in the text following the figure, only the people responsible, techniques, tools, and products for a test design/upgrade or for a test application will be specified.

- **Test design/upgrade:**

  People Responsible: Tester (internal and/or external to the organisation).

  Techniques: White and black box tests.

  Products: Test plans.

- **Test application:**

  People Responsible: Tester (internal and/or external to the organisation).

  Tools: Test, Check, Junit, Cppunit.

  Products: Test reports.

f. **Implantation**

In this phase, the actual iteration version is released to the client so that the client can validate this version while other functionalities are developed in the next iterations. The migration strategies towards the new application are defined, the user is trained for using the delivered version, the actual version is installed, the installation test is applied, and the software application manuals are generated or upgraded, and verified. Finally, the actual version of the software application is integrated along with the organisation activities. Figure 9 shows the workflow for this phase. Given that the figure sufficiently explains each step of this phase, a detailed description for each step will not be given. However, it is important to mention that: (a) the people responsible for these steps are

Figure 8. Steps for the testing phase

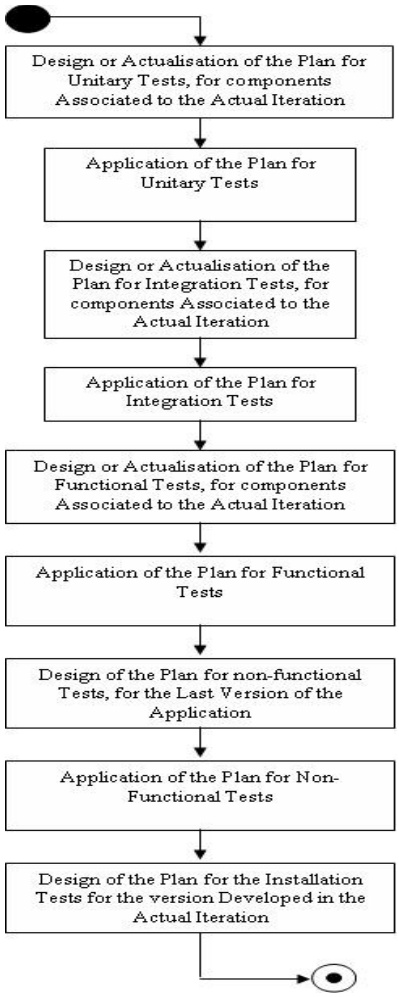
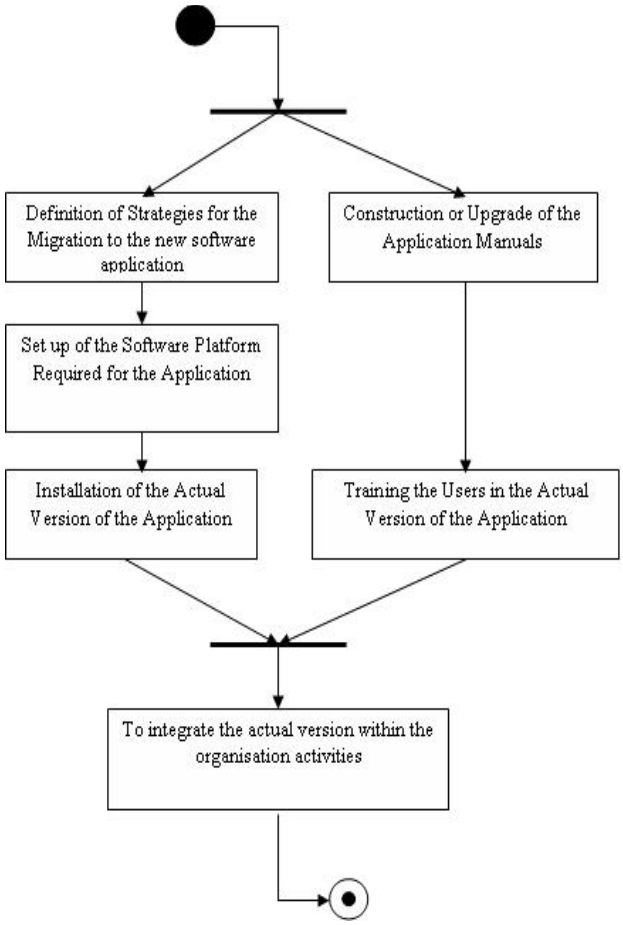


Figure 9. Steps for the installation phase



programmers and testers (both internal to the developer organisation) and (b) the main products of this phase are the system manuals, the training material, and the installation test reports for each installed version.

STUDY CASE

The presented methodology has been shaped and updated recently and has been implanted partially only in two projects. The implantation process will continue during the present year (2007) in order to apply the whole methodology to all projects. The two projects involved in the implantation process until now are:

1. The Dis-centralised Administrative System.
2. The Automation of the FSF. This means the automation of the free software project management, administration of specific projects, and software application development processes.

The Dis-centralised Administrative System had already been started and was entering the test phase at the moment the methodology began implantation. Because of this, until now, the methodology has been applied in this project only for part of the software application development process, more specifically, for the unitary tests of the test step. On the other hand, since

the automation of the FSF is still in course, the methodology has been applied up to the point the project has reached at present. However, the methodology has been applied from the beginning of the project. The following processes of the methodology have been implanted: the free software project management**,** administration of specific projects, and some aspects of the software application development process. Next, details about the application of the methodology to both of these projects will be presented.

Case 1: Automation of the Free Software Factory

Process # 1: Free Software Project Management

a. **Conceptualization**
Results of carrying out this step are summarised in a set of filled templates, which are stored in the GForge server (see Alvarez, 2007, sections 1 and 2). These templates show needs and problems, and scope of the project of implementation of the FSF. Those problems and needs include:
- Lack of a database of digitalised templates for documenting the development processes.
- The dynamics of the demand requires urgent development.
- Need of a knowledge base for learned lessons.

The scope of the project delimitates the system to be developed in terms of which processes and which steps will be covered.

b. **Project Description**
This gives an overview of how the automation of the FSF project is being carried out.

c. **Free Software License Definition**
There is not any licence defined for this project. All material developed in this project will be available from the GForge server.

d. **Software Main Functionalities Prioritisation.**
All functionalities to be covered by the automation system of the factory were defined and prioritised. Results of this phase are presented in Alvarez (2007, section 3). Among these functionalities, we have:
- Digitalise the functionalities and the risks prioritisation templates.
- Select analysis and design tools.
- Choose testing tools.
- Automate the project plan template.
- Integrate templates and design tools.
- Integrate design and programming phases (generation of automatic code)
- Develop a knowledge base.

These functionalities received a weight, as the methodology states.

e. **Risk Prioritisation**
At this step, the more important development risks were defined (see Alvarez, 2006, section 4 for more details). Among these risks, we have:
- Scarcity of free automation tools and FSF's development team's lack of knowledge and capacities for building, design, and test tools for free software development.
- Few people dedicated at testing and short experience in testing.
- Low experience in following methodologies.

As above, a weight is associated to these risks.

f. **Development Priorities Definition**
A prioritisation of the functionalities was performed (Alvarez, 2006, section 5), by following this formula:
Total functionality F_1 weight = ($\sum VR_i$ para F_1) * PR + VF_1 * PF , where,
- the VR_i are the risks for the functionality F_1;

- VF_1 is the weight for the functionality F_1;
- PR and PF are the relative weights-of-the-factors, in this case, between the total sum of risk weights, factor 1, and the functionality weight, factor 2.

Following this procedure, one of the most important functionalities resulted to be automated design and testing facilities. Tools for these tasks were selected from those available on the Internet.

g. **Software Development Plan**
This plan presents the development schedule, indicating the functionalities to be developed at each iteration; there were seven defined iterations after considering the functionality dependencies, size of the development team, and the functionalities development prioritisation (for more details, see Alvarez, 2007, section 5).

h. **Generation of the Service Offer**
The service offer indicates (Alvarez, 2007, section 6), for instance:
   - The offer proposal: to develop a system to automate the free software development processes.
   - The project scope: to automate in some degree, by integrating and digitalising (and in some cases completely automating, when the complexity of these tasks allows it) tools for implementing the FSF.
   - The release schedule (to the client).

Process #2: Administration of Specific Projects

a. Administration of the Collaborative Platform.
GForce was installed as the collaborative Platform.

b. Standards for software codification.
The codification standard is being defined at present.

c. Iterative planning.
This step is performed by using a GForce scheduling facility (Alvarez, 2007, section 7).

Process #3: Software Application Development

The software application development process consists basically on programming on top of GForge, in order to adapt it and add functionalities, to permit carrying out the software development activities required by the FSF processes. Some of the adaptations already implemented are:

- Digitalisation of templates, among which we have: client's needs and problems; scope of the project; service offer; test plan; test reports.
- Automation of the project plan.

Case 2: Dis-Centralised Administrative System

Process #3: Software Application Development

As mentioned previously, for this project, the methodology has been implanted to perform the unitary test plan (Alvarez, 2007, section 8).

CONCLUSION

The presented methodology pretends strengthening the software national sector, especially the small and medium software enterprises (including the cooperatives), by allowing them to access the technology and participate in the software market through a collaborative development of software for the public administration (main goal of the FSF), on one hand, and to increase their capabilities and software quality, on the other hand.

In this sense, the development process presents certain specific characteristics and numerous advantages (as it is stated in the methodology). As said before, a fundamental aspect is the collaborative development by iteration: a particular development group may enter or leave to collaborate at any iteration in accordance to the group interests. Additionally, the software developments and upgrades coming from any involved group are open to the community via a collaborative platform. Consequently, the development groups get benefits from a methodological framework, which establishes the ways and moments for participation, forms to recovery versions of the development, development rules and tools, and so forth. All these practices on the bases of the development framework allow any small or medium enterprise to share/communicate with partners in the free software development community, in accordance with the free software development philosophy. The proposed methodology has been partially validated at the FSF of the Foundation for Science and Technology of the Mérida State in Venezuela (FUNDACITE-Mérida). In addition, this factory has permitted to understand better, empirically, the real needs of a free software development process and has also been a source of ideas.

ACKNOWLEDGMENT

This chapter has been developed inside the project: "Process Improvement for Promoting Iberoamerican Software Small and Medium Enterprises Competitiveness – COMPETISOFT" (506AC287) financed by CYTED (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo)

REFERENCES

Alvarez, J. (2007). *Resumen del avance de la aplicación de la metodología desarrollada para la Fábrica de Software Libre* (Tech. Rep. No. 001-2007). Fundacite, Merida: Fábrica de Software Libre. Retrieved December 17, 2007, from http://www.funmrd.gov.ve/drupal/files/technicalReportJohanna.pdf

Alvarez, J., Aguilar, J., & Teran, O., et al. (2006). *Metodología para el Desarrollo de Software Libre: Buscando el Compromiso entre Funcionalidad y Riesgos* (Tech. Rep. No. 001-2006). Fundacite, Merida, Venezuela: Fábrica de Software Libre.

Beck, K. (2004). *Extreme programming explained: Embrace change* (2nd ed.). Addison-Wesley Professional.

Corredor IIMI. (2006). *Evaluación de MoProSoft como alternativa metodológica de organización de empresas de desarrollo y mantenimiento de software*. Tesis de Pregrado, Escuela de Ingeniería de Sistemas-Universidad de Los Andes, Mérida, Venezuela.

Kruchten, P. (2000). *The rational unified process: An introduction* (2nd ed.). Addison-Wesley.

Montilva, J. (2004). Desarrollo de Aplicaciones Empresariales: El Método WATCH. Mérida, Venezuela: Jonás Montilva.

Montilva, J., Hamzan, K., & Ghatrawi, M. (2000, July). The watch model for developing business software in small and midsize organizatios. In *Proceedings of the IV World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*, Orlando, FL.

Oktaba, H., Alquiara, C., Su, A., Martinez, A., Quintarilla, A., Ruvalcaba, M. (2005). *Modelo de Procesos para la Industria de Software* (MoProSoft, Versión 1.3). México. Retrieved December 16, 2007, from http://www.software.net.mx

Pollice, G. (2001). *Using the rational unified process for small projects: Expanding upon eXtreme programming* (White Paper TP 183). Rational Software.

Probasco, L. (2000). *The ten essentials of RUP: The essence of an effective development process* (White Paper TP177). Rational Software.

This work was previously published in Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies, edited by H. Oktaba & M. Piattini, pp. 242-257, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 2.25
Agile Outsourcing to India:
Structure and Management

Boris Roussev
University of the Virgin Islands, USA

Ram Akella
University of California, USA

ABSTRACT

The combination of low labor costs, technological sophistication, project management skills, and successful software establishment makes India a particularly attractive location for software production outsourcing. Furthermore, in most situations, information and communication technologies render virtual presence practically equivalent to physical presence, thus enabling effective communication and cooperation in a distributed mode. This chapter introduces a project structure creating agile conditions for large outsourcing software projects. The agility advantage is achieved by scaling down a large project into a number of small-sized projects working in agile settings. We divide the work into R&D activities, located onsite, and production activities, located offsite. The proposed approach makes Agile applicable to the stressed condition of outsourcing environments without compromising the quality and the pace of the software development effort. Creating a context congenial to agile methods hinges on maintaining a good balance between the functions and sizes of onsite and offsite teams, on redefining the developers' roles, and on reorganizing the information flow between the different development activities to compensate for the lack of customer onsite, team co-location, and tacit project knowledge.

INTRODUCTION

We live in a digital world, where any activity not requiring a "physical presence" can be outsourced to any place that is connected. Even the term "physical presence" comes with a qualification. Information and communication technologies (ICTs) enable cooperation in a distributed mode. Technologies, such as groupware and video-conferencing, are increasingly becoming feasible for

organizations to use in international projects. In addition, these technologies are changing the way we perceive presence and absence. The largely digital nature of software development allows changing its geography of provision. Advances in ICT have been essential for loosening the spatial constraints on software development.

The combination of low labor costs, technological sophistication, project management skills, and successful software establishment makes India a particularly attractive location for software production outsourcing. From 1996-1997 to 2003-2004, the export software sales of the Indian IT industry had an average annual growth rate of 38.7% to reach a total of US$12.2 billion in 2003-2004 (Nasscom, 2004).

Even though India has had a qualified labor pool and the enabling technologies, along with the great pressure exerted on firms in developed nations to lower costs, the Indian software industry still operates in the low value-added segments, typically in applications development, testing, and maintenance, while the high-end work, such as developing the IT strategy, building the software architecture, designing the system, integrating the project with enterprise packages, and designing custom components are all discharged by firms in developed countries (Dossani & Kenney, 2003).

Agile methods (Beck, 1999) are popular software development processes designed to be used on small- to mid-sized software projects. The Agile movement started in the mid-1990s. The first major project to apply an agile method was the Chrysler Comprehensive Compensation system, a payroll system developed in 1996. This method, called extreme programming, was described in Beck (1999) and became the foundation for the Agile Alliance Manifesto (Agile Alliance, 2001).

Agile methods are based on the notion that object-oriented software development is not a rigidly defined process, but an empirical one that may or may not be repeated with the same success under changed circumstances.

Agile methods are based in four critical values—simplicity, communication, feedback, and courage—informing a set of key practices (Pollice, 2004), which will be considered later on. Boehm and Turner (2004) define agile methods as "very light-weight processes that employ short iterative cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation."

Many of the agile practices are incompatible with the context of outsourcing, for example, customer onsite, team co-location, short lifecycle, and embracing change. But above all, agile methods can be applied only to small-sized projects.

In the past, there have been several attempts to reproduce the conditions for agility in large-sized projects. To the best of our knowledge, no such attempt has been made for outsourcing projects. Pollice (2001), Evans (2004), and Boehm and Turner (2004) propose to scale up agile methods by balancing agility and discipline. Pollice and Evans, for instance, look out for common grounds between Agile and RUP (Jacobson, Booch, & Rumbaugh, 1999), while Boehm and Turner try to get the best of both agile and plan-driven (waterfall) worlds. In contrast, Kruchten (2004) proposes to scale down large projects to meet the Agile "sweet spot," based on experience reported in Brownsword and Clements (1996) and Toth, Kruchten, and Paine (1994).

In this chapter we show how to reengineer large-sized (and small-sized) outsourcing projects to benefit from the "sweet spot" of Agile, while avoiding its "bitter spot." The proposed approach makes Agile applicable to the stressed condition of outsourcing environments, without compromising the quality of the software development effort. Creating a context congenial to agile methods hinges on maintaining a good balance between the functions and sizes of onsite and offsite teams, on redefining the developers' roles, and on reorganizing the information flow between the different development activities.

The rest of the chapter is structured as follows. First, we examine the state of the Indian software industry and show the problems experienced by Indian software suppliers. Next, we present the structure of the Agile outsourcing project, customized to the needs of the Indian outsourcing context. We then elaborate the inception and architectural activities, giving enough detail of how large-sized projects can be decomposed to a number of relatively independent agile projects and showing how the resulting builds are integrated. A discussion of many of the issues likely to be faced by Indian suppliers when applying the proposed approach follows, and the final section concludes and outlines plans for future work.

OUTSOURCING TO INDIA

Brief History

The genesis of outsourcing to Indian software firms can be traced back to the 1970s. By the late 1990s, India had become a leading supplier of contract software programming due to its combination of skilled, low-cost labor and project management skills (D'Costa, 2003). Indian software firms such as HCL, Infosys, and Wipro have become globally competitive. Further, their interaction with the global economy has contributed to the development of executive and managerial talent capable of securing overseas contracts, managing the interface with foreign customers, and migrating software development activities across national and firm boundaries.

Multinationals in developed countries and domestic firms quickly understood that there was a significant opportunity to undertake labor-cost arbitrage offshoring to India and moved, beginning in the 1990s, to establish Indian operations. Because the economics were so compelling, Indians living in the U.S. and the UK soon followed suit by establishing outsourcing firms in the U.S. and the UK, respectively, which discharged the work to India.

Drivers

The foremost reason for outsourcing is the potential cost savings, with quality commensurate or even better than that currently provided by the developed nation software developers. India-based outsourcers estimate that, in general, savings on a given activity would have to be at least 40% to make the relocation worthwhile (Dossani & Kenney, 2003). We group the rest of the drivers for outsourcing into technological and economic.

During the last decade, the cost of data transmission has dropped by more than 50%. The lowered telecom costs make it feasible to undertake even communication-intensive tasks outside of the U.S.

Another technological enabler is the ongoing revolution in the world of documents. Today's industrial-strength scanners digitize documents at the rate of 400 pages per minute. The digitized documents can be viewed anywhere in the world on a computer with a high-speed connection.

The last technological development we would like to consider is the widespread use of standard software platforms in corporate information systems, for example, database systems and ERP systems. Standard software platforms facilitate outsourcing since domain knowledge becomes uniform and transferable across industries, or branches thereof. This means that employees ought to acquire only standard, portable skills, thus lessening risks for both suppliers and clients.

Technology is necessary, but not sufficient to convince companies that they should outsource software development to India. Clients need assurance that the process would not be to their detriment and would bring in good return on investment. India has a very successful software establishment, with a proven track record

in satisfying international customers in the area of application programming and maintenance. This creates the necessary comfort levels in the clients. The comfort-building process is assisted by the visible presence of large multinationals like IBM, Dell, Microsoft, General Electric, and British Airways, who have established Indian operations over the past years.

Another very significant factor driving software development outsourcing is the profitability crisis in the developed countries. With revenues largely stagnant for the last four years, firms are under intense pressure to cut costs while retaining quality.

Analysis of the Indian IT Sector

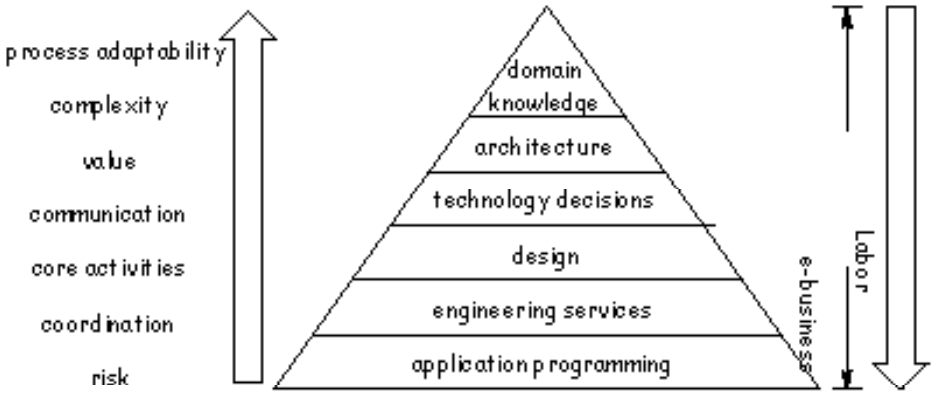
In their analysis, Akella and Dossani (2004) divide the Indian software export into two functional areas: services and products. The authors conclude that, in countries like India, the ability to develop global (but not local) products is more difficult relative to producing services. They observe that, in the Indian software export, services dominate products, and also that services in particular are supplied mostly to non-IT firms in the core, non-critical components of their activities.

A second division is between serving clients from the IT sector and clients outside the IT sector. The ratio between the two sectors offers an insight into the maturity of a software industry: the greater the numbers and the greater the relative weight for the non-IT sector, the more mature the software industry is.

Akella and Dossani classify the Indian software industry output into the following nearly exhaustive list of categories: 1) Web-based support; 2) data processing and management; 3) embedded software; 4) maintenance and legacy systems; and 5) application software development and maintenance. The complexities of the activities involved in all five categories fare low in the complexity pyramid of labor, as shown in Figure 1. Moreover, all five categories predominantly serve clients from the IT sector.

Both IT and non-IT client activities can be divided into core and non-core on the one hand, and critical and non-critical on the other hand. Core activities are manufacturing and engineering services that differentiate a firm from its competi-

Figure 1. Activities and pyramid of labor



tors, and they are key to its continued growth. Non-core activities are the firm's non-competitive services, such as payroll and human resources. The boundary between core and non-core is not clearly cut. Within both core and non-core activities, there are activities crucial to their functioning, termed critical activities.

From Indian suppliers' point of view, non-core and non-critical activities are the easiest to obtain because they tend to be the most standardized across industries, and clients are most willing to outsource them because they do not involve knowledge transfer. Indian suppliers commonly work on applications at the next more difficult level—core, non-critical activities, and rarely on core, critical activities (Akella & Dosssani, 2004).

Strengths of Indian Software Firms

The substantial share of services provided to non-IT clients suggests that Indian firms have expanded their project management capabilities and started to acquire domain knowledge. This could explain the trend of moving away from consultants to direct supplying of end users.

The resource of skilled and motivated people is the most important component of any project. The level of experience and educational qualification of staff at Indian software firms is commensurate to that of most developed countries. The major Indian corporations, for example, are overwhelmingly qualified for ISO 9001, a European standard on quality assurance, and most of them have SEI/CMM certification with a high average score of 4.2 (out of 5) (Nasscom, 2004). SEI/CMM is the Carnegie Mellon University-based Software Engineering Institute's Capability Maturity Model (Paulk, Curtis, Chrissis, & Weber, 1993).

Availability of skilled workers is complemented by a rising supply of CS/EE graduates, which was expected to reach about 99,000 new graduates in 2004-2005 (Nasscom, 2004). Furthermore, Akella and Dossani's study implies that Indian suppliers have the skills to manage projects remotely.

It has been observed that the ratio of onsite/offsite workers increases with firm size. This trend reflects the growing capabilities of larger Indian firms to provide more sophisticated consulting services. The approach proposed in this work relies on an increased onsite presence.

Outsourcing Software Development Activities

What Can Be Outsourced?

When the software development process is considered in its totality, it appears to resist relocation because software production requires face-to-face interactivity with clients (and among co-developers), for example, user requirements elicitation and testing. The development workflow has to be parsed into activities requiring different levels of interactivity, and the less client-communication-intensive activities can be potentially outsourced.

The software complexity chain is frequently described as a series of processes, each of which is less complex than the earlier one. The initial processes are less labor intensive than the later ones. The complexity of a process is roughly inverse-proportional to its labor intensity. The pyramid model in Figure 1 gradates the processes in terms of labor, complexity, risk, and communication intensity.

Theoretically, outsourcing is possible for all the levels of the complexity pyramid, but there are important differences of how outsourcing for each level is likely to be implemented. The processes at or closer to the pyramid's apex, such as domain knowledge acquisition, architecture design, and technology determination, are objectively more difficult to outsource.

Moving up the complexity pyramid entails establishing an intimate client-supplier rapport and acquiring knowledge about the client's core

and critical activities. The higher the pyramid level is, the more communication-intensive the activities become, and the bigger the demand grows for domain, architectural, and design knowledge. Adopting agile methods can help Indian suppliers move up the "value chain" because agile methods address the issues enumerated above.

Developing domain expertise, however, is difficult and risky because the firm becomes dependent on a single industry sector or process. And yet, specialization offers the potential of finding a niche outside the ferocious competition in highly commoditized sectors.

Smaller software firms are at a disadvantage because they lack domain skills and acquiring domain knowledge may be risky. The step transforming the business proposition from simple labor cost arbitrage to significant value addition involves acquiring deep-enough domain expertise.

Interactivity

Interactivity always comes across as a stumbling block for outsourcing. Interactivity has two dimensions—interaction among co-developers and interaction with clients. Requirements elicitation and acceptance testing are the activities requiring the most active involvement on the part of the client, which makes them impossible to offshore.

The greater the need of co-developers to interact across a set of different activities of the software process, the higher the risk threshold of outsourcing a subset of the activities is. Outsourcing the entire set of activities might be considered as a way of retaining interactivity at the new location. But, if some activities cannot be outsourced because that would disrupt the interaction with the client, then outsourcing the others might need rethinking.

Savings from Concentrating Activities in One Location

Often, a number of teams distributed across multiple time zones and reporting to different companies work on a large common project. There might be different reasons for having multiple teams with an inefficient spatial posture cooperating on a project. Most commonly, as companies expand, they outgrow the local labor pools and must establish teams in other regions. It might be too expensive to consolidate these teams in a single location in a developed country, especially for multinationals (e.g., a company with teams in France, Switzerland, Canada, and the Silicon Valley).

The advantages of concentration by relocating to India stem from the larger team size, which relates to economies of scale, such as retaining interactivity at the new location, standardized management practices, and tacit knowledge dissemination. Furthermore, a larger team is likely to have experts with precious domain knowledge in different industry sectors or technologies. In addition, an outsourcing supplier could pool many clients' businesses. A big firm in India can offer guarantees of quality that smaller domestic software firms could not match, owing to the larger labor pool and the division of labor.

For foreign equity participation companies (FEPs)—Indian firms whose foreign equity funding is 100%—the software development process can be reengineered during the act of outsourcing, and inefficient and ineffective practices can be easily abandoned. Very often such practices are legacies of earlier methodologies, such as waterfall lifecycle, iteration planning, and project controlling, that were not eliminated as the development process evolved. As Arthur (1994) has observed, all too often processes evolve in a path-dependent manner, which may not be the most efficient configuration. These inefficiencies can be addressed during the transfer without

disrupting work patterns, since the developers at the new location would not have the outsourcing company culture.

Rethinking of Earlier Cost-Benefit Decisions

The lower cost of more highly skilled personnel calls for rethinking of established cost-benefit decisions. The much lower cost of software engineers in India compared to the U.S. makes it possible to increase the density of software inspections or to dispel managers' doubts about the feasibility of pair programming. Other activities that are candidates for reconsideration are regression and integration testing, iteration planning, and metrics for tracking project progress. In all of the above-mentioned practices, lower labor costs can change the break-even point, and thus create new sources of revenue.

There may also be diseconomies of scale. There are quite naturally risks associated with centralizing the development process at one location. The most significant of these is the danger of losing touch with the national environment(s). In this respect, the effectiveness of the communication process between supplier and outsourcer is critical.

Problems Experienced by Indian Suppliers

The following issues are likely to have an impact on outsourcing. The major challenge faced by Indian software firms is shortage of domain expertise in some function areas. According to the Outsourcing Institute (2004), domain knowledge and expertise are needed in subject areas that are new to India, such as finance, insurance, healthcare, and logistics. However, Indian software firms have acquired sufficient expertise in accounting and banking, which have already been successfully outsourced to India. Maintaining a seamless relationship between the outsourcing supplier and the outsourcing organization back in the developed country is always a challenge.

The problems experienced by Indian suppliers can be "objective" or "subjective." For instance, it is objectively difficult to communicate effectively with a remote client. The lack of expertise in project scheduling is, on the other hand, subjective, particular because it is not universally true. More often than not, a lot more could be done to alleviate the subjective problems rather than the objective ones.

Outsourcing problems fall into three categories: 1) communication—relationship with remote clients; 2) technical know-how—domain expertise, experience in building software architectures, design experience, and inadequate quality thereof; and 3) management—project coordination and project scheduling.

In order to compete successfully on the global market for software products and long-lasting assets (reusable domain solutions), Indian software firms need to resolve the communication, technical know-how, and management issues listed above.

THE AGILE OUTSOURCING PROJECT

In this section, we present an agile method of software development geared toward the context of outsourcing.

Core Agile Practices

Agile methods are iterative and incremental. In an iterative process, the activities that were performed earlier in the process are revisited later. Revisiting activities provides developers in areas such as requirements engineering and software architecture with the opportunity to fix mistakes they have made. The iterative model implies more

interactions among the various management and technical groups. It is a means of carrying out exploratory studies early on in a project when the team develops the requirements and discovers a suitable architecture.

Some important agile practices are the following (Beck, 1999):

- **Embracing Change:** Respond to change quickly to create more value for the customer.
- **Fast Cycle:** Deliver small releases frequently, implement highest priority functions first, speed up requirements elicitation, and integrate daily.
- **Simple Design:** Strive for a lean design, restructure (refactor) the design to improve communication and flexibility or to remove duplication, while at the same time preserve the design's behavior.
- **Pair Programming:** To quote from Beck, "If code reviews are good, we'll review code all the time." With pair programming, two programmers collaborate side by side at one machine. This quality control practice also helps disseminate tacit knowledge among the team members.
- **Test-Driven Development:** Quality control technique, where a developer first writes a test, and then writes the code to pass that test.
- **Tacit Knowledge:** Preference for project knowledge in team members' heads rather than in documents.
- **Customer Onsite:** Continuous access to a customer to resolve ambiguities, set priorities, establish scope and boundary conditions, and provide test scenarios.
- **Co-Location:** Developers and onsite customer work together in a common room to enhance tacit project knowledge and deepen members' expertise.
- **Retrospection:** A post-iteration review of the work done and work planned. This reflective activity facilitates learning and helps improve the estimates for the rest of the project.

The enumerated practices (or disciplines) vary with the method. The disciplines can be divided into three broad categories: 1) communication, such as pair-programming and tacit knowledge; 2) management, such as planning game, scrums (short daily planning sessions, in which the whole team takes part), short cycle, and frequent delivery; and 3) technical, such as test-driven design, simple design, and refactoring. These categories correspond to the three groups of problems experienced by Indian suppliers, as previously discussed.

The fast cycle of agile methods gives business stakeholders multiple and timely feedback opportunities, which makes agile methods explorative (aid in architecture discovery and requirements verification) and adaptive to change. There is a broad consensus that agile methods outperform other software development methods for small projects in dynamic environments, such as in e-commerce.

What we see as a problem with all agile methods (e.g., extreme programming) is that they do not provide guidelines for building high-quality software, which is a natural consequence of their informal nature.

The feasibility-impact grid in Figure 2 shows how the core agile practices can alleviate the Indian suppliers' concerns.

Structuring the Agile Outsourcing Project

The main question we address below is how to reproduce the conditions ideal for agility in an outsourcing project.

Even a quick look at the core agile practices above would reveal that some of them are incompatible, while others are not entirely consistent with the context of outsourcing, for example,

Figure 2. Impact of agile practices on Indian suppliers' concerns

Practices\ Concerns	Domain expertise	Architecture	Design	Scheduling	Communication	Coordination	Quality
Customer onsite	√				√		
Fast cycle		√	√		√		√
Embracing change		√		√			√
Test-driven development		√					√
Refactoring			√	√			
Simple design			√				√
Retrospection		√			√	√	√
Risk mitigation				√			
Tacit knowledge			√		√	√	
Co-location			√		√	√	
Planning game				√		√	
Scrums				√		√	
Pair-programming			√				√
Immediate customer feedback	√	√			√		√

customer onsite, co-location, short lifecycle, and embracing change. Above all, agile methods can be applied only to small projects.

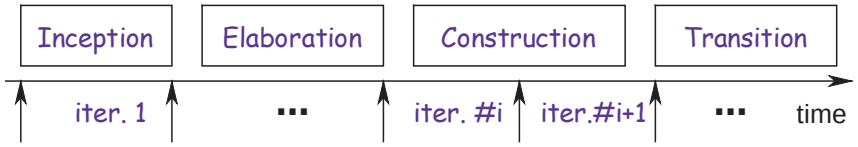
Kruchten (2004) proposes to scale down large projects to meet the Agile "sweet spot." The author describes the organization of a large project as an evolving structure, starting out as one, co-located team, which over time is transformed to a team of teams (with a tree-like structure).

Kruchten suggests organizing the iterative process into four typical RUP phases, as shown in Figure 3. Each phase shifts the focus of the development effort onto a different activity. A phase consists of one or more iterations, where iterations can be thought of as mini waterfalls.

The structure of the agile outsourcing project is based in Kruchten's approach. However, in an agile outsourcing project, the primary goal is not to slice a big project into multiple agile subprojects (which might be required anyway), but to outsource the project to one or more agile teams, which are co-located at a remote site and share common culture and educational background.

The structure of the development process is illustrated in Figure 4. The phases of the lifecycle are separated between "research and development" (R&D) activities and "production" activities, as suggested in Royce (2002). R&D is carried out onsite—close to the client—while production takes place offsite (the supplier's site). Elaboration

Figure 3. Typical RUP phases of the iterative process



is split between R&D and production. The two new phases are called architectural elaboration and production elaboration.

A team comprising requirements engineers and architects starts the development process. Initially, this team focuses on the business case, vision, and requirements. The team works closely with the client in a typical agile setting. The team's objective, however, is not to deliver executable code (a must in agile methods), but to set up an architectural prototype, including prioritized system-level use cases.

When the team has got enough clarity on key architectural choices, it can set about building and testing an architectural prototype.

Towards the end of the architectural elaboration phase, when the architecture stabilizes, additional teams are created offsite. Each new team is seeded preferably with two members of the architecture team. For large projects, the early architectural prototype is used to slice the project into smaller, considerably independent agile projects. Each agile project "owns" part of the system architecture.

The "seed developers" transfer domain expertise from the client to the supplier's site. The "seed developers" take on several roles. They lead the teams, serve as local architects, act as customer surrogates, communicate with the initial architecture team, and if necessary, communicate directly with the client. This organization creates near-perfect conditions for agility in each offsite team.

Each agile offsite team works on a subsystem and develops its detailed subsystem use case model—hence the second elaboration phase, named "production elaboration" in Figure 4.

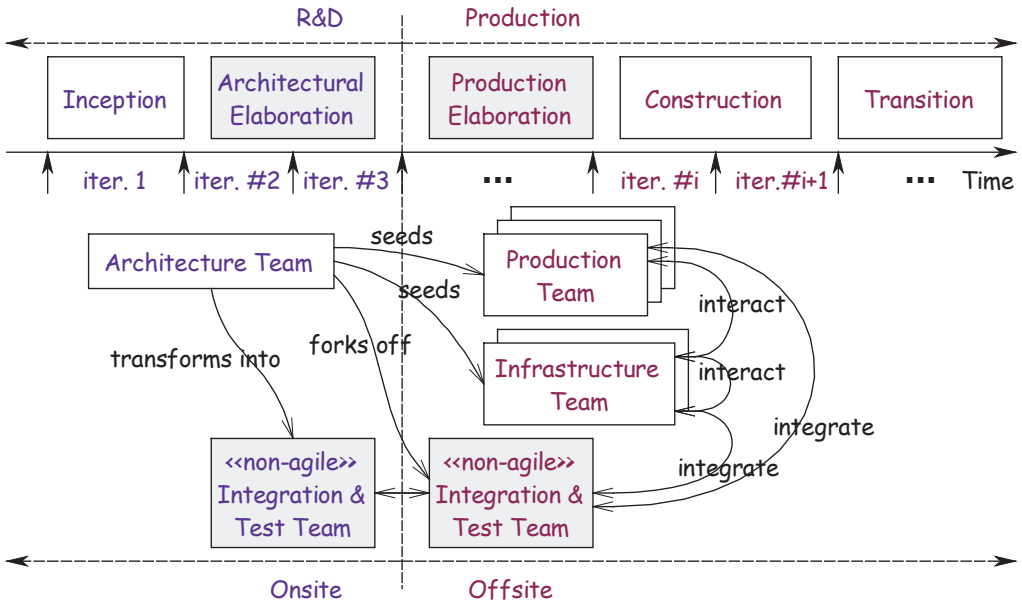
For small projects, one offsite production team will suffice to develop the complete code. For large projects, however, several production teams might be necessary to do the job. In addition, one or more infrastructure teams must be set up to develop common elements such as middleware, services, and any reusable assets. The customers for the infrastructure teams are all other teams. Thus, no matter how many teams are spawned offsite, they all work in agile conditions, even though predominantly with customer surrogates.

It is important to use a common software tool to reduce the risk of miscommunication, to track and communicate project requirements, to identify replicated modules, to map test cases to requirements, and to successfully integrate the outputs from the agile teams' builds.

The context diagrams in Figure 5 model the environments in which the different offsite teams operate. Note the dual nature of "seed developers." On the one hand, a "seed developer" impersonates a customer, and on the other hand, he/she is part of the team.

The interactions of offsite teams with the real customer are supposed to be infrequent, and to

Figure 4. Team structure of agile outsourcing



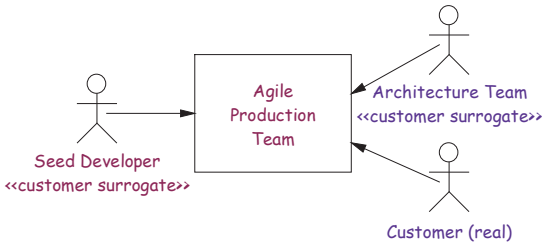
be mediated by the "seed developers" and the architecture team. For large projects, somebody needs to put together the builds delivered by the production and infrastructure teams, and to test the assembled system. This job is assigned to the Integration & Test Team, or integration team for short. The testing engages the client so that the client's feedback situates the project and steers the future effort.

The problem with the integration team is that it gets input from the offsite teams, but receives feedback from the client. Normally, the input and the feedback are coming from the same place—the customer. To account for this anomaly, we split the integration team into two teams, one located onsite and the other offsite (see Figure 4). Both integration teams derive directly from the initial architecture team. This guarantees that the developers in charge of integration and testing thoroughly understand the client's needs.

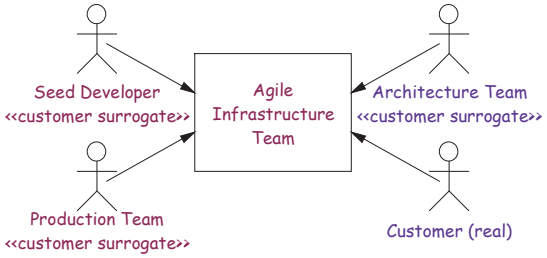
A sore issue stemming from the proposed organization is that of fault propagation. For example, a defect committed by one of the production or infrastructure teams can propagate through the integration process before the client detects it. Owing to the late stage of its detection, isolating and removing such a defect is expensive.

We have provisioned for two floodgates preventing fault propagation: 1) the "seed developers" in each offsite team; and 2) the test engineers in the integration team. Since they all come from the primary architecture team, it is very likely that they would be able to detect many of defects, which are normally revealed with help from customers.

Figure 5. Context diagrams for offsite teams

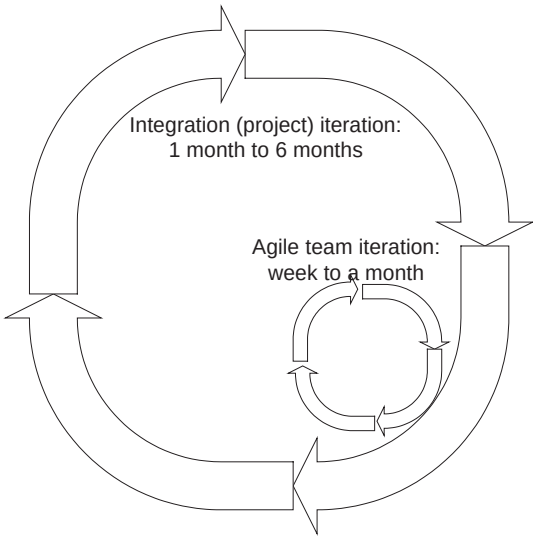


(a) Production team.



(b) Infrastructure team.

Figure 6. Nested iterations



The only two teams operating in non-agile, but still iterative and incremental mode, are the onsite and offsite integration teams.

The iterations (heart beats) of both the agile and integration teams can be best illustrated with the dual beat structure shown in Figure 6. Striking a balance between the lengths of the agile and integration iterations is the underlying objective of the management team.

The management team, composed of all local team managers, is led by the managers of the integration teams. We do not show the management team as a separate box in Figure 4 because management is thinly distributed across all teams.

Since all offsite teams reside in the same country, and most probably in the same city, say Bangalore, there are no cultural differences to overcome, and communications among production, infrastructure, and integration teams are not as ineffective as they are with geographically distributed teams. The management team is in a favorable position because good communication is a prerequisite for effective coordination.

ACTIVITIES AND WORKFLOW

In this section, we give more detail about the activities in an agile outsourcing project, as well as their workflow and information artifacts. We focus on inception, architectural elaboration, and production elaboration because they are critical to creating the agile contexts for the offsite teams.

Inception Phase

During inception, the emphasis is on the user requirements. A user requirement is a specification of what the system must do. As user requirements are elicited, they are organized into use cases. Very briefly, use cases are dialogical descriptions of system usage. By coalescing related transactions and scenarios into identifiable chunks, use case models focus on the dynamic behavior of a system at a macro level.

Use case descriptions can smoothly scale to large and small systems alike. They promote refinement and traceability from system-level usage goals down to low-level (subsystem, component, and instance) usage goals. Use cases are sufficiently flexible to be used in highly iterative and incremental development environments, as the one proposed in this work.

If a system consists of several subsystems, use case analysis can be applied recursively to the subsystems as well. This defines clearly the requirements and responsibilities of each subsystem. Subsystem-level use cases are derived from the system-level use cases and the system architecture (the architectural decomposition of the system into subsystems).

A system is decomposed into subsystems of related use cases, for example, use cases linked by <<include>> and <<extend>> dependency relationships or use cases sharing common actors. The <<include>> relationship is used to extract out a coherent part of a use case, typically for the purpose of reuse. It can also be used to decompose a system-level use case into part use cases to be realized by different subsystems.

The following technique is helpful in mapping a system-level use case to two or more subsystem use cases. The high-level use case is decomposed into a partially ordered set of activities on an activity diagram, where the different subsystems are modeled as swim lanes. Concurrent activities can be mapped to different swim lanes. Each such activity forms a subsystem use case. The actors of the subsystem use cases may be the original actors or other subsystems.

Architectural Elaboration

The onsite team carries out the architectural elaboration. The goals of the architecture team are to partition the system into multiple semantic

domains centered around different subject matters, to define the architectural decomposition of the system into subsystems, and to map system-level use cases to subsystem use cases.

The architecture team defines the model organization of the system being developed, and thus determines the allocation of jobs to teams. Appropriate model organization would allow teams to work effectively on a common project. The two main issues in model organization are how to allow different teams to reuse parts of the projects they do not "own" and how to efficiently implement a build process.

A UML package is a container for modeling elements, and as an organizational unit, it is a natural choice for a configuration item (CI) (a piece of ownership) in a configuration management (CM) tool. A package defines a namespace for the modeling elements it contains. Since UML offers no guidelines as to what constitutes a package, the question is what modeling elements should go into one package versus another.

Domain Modeling

A domain is a subject area with a shared vocabulary (Mellor & Balcer, 2002), for example, user interface (UI) or payment transaction management. Each domain contains many classes organized around a single subject matter. Most domains require specialized expertise, such as experience and knowledge in UI design or in payment transaction management. It makes sense to allocate the modeling of a domain to a developer with domain knowledge in that particular subject area.

Since a domain model captures precisely the conceptual entities of a single subject matter, it can be said that domain models are logical models. Logical models are in sharp contrast to subsystem models, which are pieces of the physical system. Typically, a physical subsystem is constructed from instances of several logical models. For example, a collaboration realizing a system-level use case would involve instances from a UI domain, a business logic domain, a transaction management domain, a persistent storage domain, and a security domain.

At first, it might seem that domains add yet another concern to a software development initiative, but it is all about economic numbers. Constructing domain models reduces production cost by simplifying the development process and by improving product quality.

The underlying philosophy is that a domain-specific model captures precious domain knowledge (e.g., persistence). Domain modeling leverages scarce domain knowledge normally limited to very few team members and shields the rest of the team from the domain implementation detail. For example, to make an object persistent, a developer needs only to mark its class or one of its attributes as persistent, and a persistent software entity is automatically generated at compile time. The result is a simplified development process, where only a few developers need detailed knowledge about domain technicalities.

Henderson-Sellers (1996) and Bettin (2004) observe that software quality degrades faster when software is treated as capital cost (i.e., maintenance only delays the inevitable obsolescence). Treating software as capital cost is incompatible with incrementally building software assets[1] and strategic software[2] assets reused across a large number of enterprise applications, software assets whose value appreciates rather than depreciates. Since domain modeling leverages domain-specific knowledge captured in the form of domain models, strategic software assets are built at no extra cost. Domain models do not degenerate into liabilities[3] over time because they are aligned with the business process architecture of the enterprise and they have the potential to achieve mass customization[4] (Bettin, 2004).

Domains, unlike objects, are not elemental, but just like objects they are cohesive. The classes and components in a domain are tightly coupled

and interdependent, and yet the domain is autonomous—its classes and components are decoupled from entities lying outside the domain boundary. Once constructed, domain models have greater longevity than an application because they evolve independently of other domain models out of which the application is built; that is, they become corporate assets and the biggest units of reuse.

Companies want to be able to adapt their software systems to the constantly changing business environment with minimum effort. It is easy to cope with the intra-domain effect of a change because the domain model naturally fits to the domain's subject matter, and the effect of the change is expressed in the language of the domain (i.e., in familiar domain concepts). The latter taken to the extreme is called end user programming. This scenario works only if no changes are made to the modeling notation. Changing the modeling notation leads to domain engineering and software product line engineering (SEI, 2004).

Domain modeling also alleviates the problem with the inter-domain effect of a change. The semantic autonomy of each domain implies that a domain can be replaced by another one using different conceptual entities or different implementation without affecting the application.

Domain modeling is a pragmatic approach that avoids heavy up-front investment and long-term design degradation (Bettin, 2004). It provides an incremental path of building sustainable domain-specific assets. Software products built out of domain models are not liabilities designed as one-off systems.[5]

Partitioning the project into domains promotes the development of domain models, with externalized interface definitions serving the goal of active dependency management. This is especially true for a domain model that has been developed by domain experts over several years. The market for such software assets has not matured yet, but it is projected to grow up (Booch, 2004; Mellor et al., 2004).

Structured Classes, Subsystems, and Components

In UML2.0, developers represent the containment hierarchy of a system through structured classes. A structured class may contain internal structured classes and instances, each of which may be further decomposed into structured classes and instances, ad infinitum.

The concept of a structured class is based on decomposition and encapsulation. The parts contained in the structured class define the decomposition aspect. Of course, the structured class is more than a simple container for the parts. A structured class has parts connected with connectors, publishes services via provided interfaces, provides runtime connections via ports, and imposes demands on other structured classes (servers) via required interfaces.

Components are structured classes constituting the primary replaceable units of software. In addition, components have an <<artifact>> section used mainly to specify a unit of deployment, such as a .dll file (dynamic library).

In UML 2.0, a subsystem is defined as a subordinate system within a larger system. The subsystems define the large-scale physical architecture of the system. Subsystems are specialized components (i.e., structured classes) used to decompose a system into parts. A subsystem has one compartment for its specification and one compartment for its realization. Either or both of these compartments could be empty or omitted. Developers can use ports and interfaces to show how subsystems relate to other subsystems. The components and objects within a subsystem cooperate to realize a common set of use cases. Ports and interfaces aid in encapsulating the subsystem's internal structure.

Model Organization

Use Case-Based Model Organization for Small-Sized Systems

For small-sized systems, the package structure can be organized by use cases. The system model is divided into packages of related use cases. This model organization is straightforward and allows tracing requirements easily to model elements. The downside of the use-case-based model organization is that it does not scale up well and encumbers reuse. Developers are forced to reinvent similar classes in different use case collaborations.

A quick remedy is to add a framework package for common elements such as usage points (services) and extension points (classes to subclass in use case packages). Regardless of this remedy, the use-case-based model organization still works well only for small-sized systems. Identifying commonalities in large systems leads to an explosive context and inter-team communication growth.

Domain-Based Model Organization—Large-Sized Systems

We propose to derive the package structure and contents for large-sized systems from the requirements model, the architecture model, and the domain model. The top-level packages of the domain-based system model are:

- System use cases and actors package
- Domains package
- Infrastructure package
- Subsystems package
- Builds package

The system use cases package contains system-level use cases and their actors. The domain package has one sub-package for each domain.

The infrastructure domain is a special type of domain. Infrastructure domains contain services and extension points for system communication and infrastructure, and they are dependent on the selected implementation technology (e.g., RMI/J2EE or CORBA). The infrastructure package contains one sub-package for each infrastructure domain. An infrastructure domain evolves into a self-contained subsystem package, which is assigned to an infrastructure team. Several production teams may use the classes and components, realizing the services of the infrastructure package.

Each subsystem package contains the classes and components of a subsystem, the largest-scale pieces of system functionality. If any of the subsystem packages is large enough, it can be recursively divided into sub-packages until the size of the leaf packages becomes manageable. A leaf package is a unit of ownership, such as a CI in a CM tool. A subsystem package normally refers to classes of several domain packages.

The builds package is divided into sub-packages, one per prototype to allow for easy management of incremental builds. This package also includes the system test model, such as test cases, test procedures, and test scripts, along with the drivers, stubs, and collaborations realizing the test model.

The domain-based model organization scales up well to large-sized projects for three main reasons. First, subsystem package decomposition can be applied recursively, resulting in subsystems realizing subsystem use cases. Second, classes from different domains may be reused in different deployment settings of the designed system, and third, domain models are assets that can be reused across projects.

The following are the major problems teams may experience with the domain-based model organization. Developers tend to blur the distinction between domain models and subsystem models. There is an overhead associated with maintaining

two separate types of models: domain models (logical) and subsystem models (physical).

If reuse is not a primary objective or if the system being designed is a small-sized one, a use-case-based model organization might work better because of the lower added overhead associated with model organization.

Production Elaboration

All strategic decisions for the overall organization structure of the system have been made in the architectural elaboration phase. Production elaboration drills down inside the subsystems to specify the semantic objects whose collaborations deliver the system's structure and behavior.

The demarcation line between architectural and production elaborations is the divide between logical and physical models, and between system-level use cases and subsystem use cases.

In production elaboration, the subsystem use cases are detailed and mapped to collaborations of components and objects. The following technique can assist in elaborating a system-level scenario. The developer starts out with the high-level sequence diagram for the scenario, and then adds lifelines for the instances of the identified classes. In this way, the developer can trace the elaborated scenario back to the original one and verify the design. Communication diagrams are extensively used to represent the stabilized semantic object structure. The measure of goodness at this level is benchmarked by whether the architectural design can realize the usage scenarios defined at the system level.

The described approach scales up well to many levels of subsystem decomposition. As mentioned earlier, the seed developers and production teams serve as customer surrogates.

Packages	Team
System use cases package	Architecture team
Domains package	Reused if already developed or assigned to the architecture team
Infrastructure domain packages	Infrastructure teams
Subsystem packages	Production teams
Builds package	integration and test teams

Next we show how packages are assigned to teams, for example, the architecture team is in charge of developing the system use cases, and therefore "owns" the systm use cases package.

DISCUSSION

Challenges to Agile Outsourcing Projects

Managers of agile outsourcing projects ought to be aware of the following challenges looming ahead.

The architecture team is the key to project success. It is of paramount importance that this team be a good mix of requirements analysts and architects. According to Akella and Dossani (2004), the total percent of technical staff in Indian software firms is about 85% and of MBAs about 4.9%. In contrast, in Silicon Valley, 28% of the Indian IT engineers have MBA degrees, which is a significant comparative advantage. According to the same study, Indian suppliers are mostly "pure-IT" firms, that is, they have limited domain knowledge outside the IT sector, and the average percentage share of architecture/technology consulting amounts to about 2.9%.

In this context, we see the formation of a balanced architecture team as a major challenge to agile outsourcing projects, for members of this team elicit requirements, build the primary system architecture, and subsequently take the roles of surrogate customers, local managers, integration and test engineers, and communicate with the customer throughout the entire lifecycle.

Inter-team dependencies reduce teams' abilities (especially affected is the infrastructure team) to test code and deliver builds. Production teams should provide early on the infrastructure team(s) with stubs, so that the work of the infrastructure team(s) proceeds smoothly.

In large projects, it is difficult to balance teams' workloads. Adaptive scheduling algorithms in grid computing have proven superior to static scheduling algorithms (Blumofe & Leiserson, 1999; Rousev & Wu, 2001). We propose employee stealing as an adaptive technique for load balancing. An overloaded team (thief) picks up a victim team at random and tries to steal a developer from the victim. If the victim is under-loaded, the stealing succeeds. Otherwise, a new attempt to steal a developer is made.

The random choice of a victim team and a developer makes the technique dynamic and adaptive. The geographical proximity of the offsite teams and the agile method applied are essential prerequisites for employee stealing because they facilitate fast learning.

Employee stealing is a knowledge dissemination technique, which can be viewed as a generalization of pair programming and developer rotation in extreme programming. Employee stealing may be applied as a routine practice, even with balanced projects, in order to improve inter-team communication and coordination, to arbitrate in problems straddling two teams, and to preclude loss of common goal.

With multiple teams, there is always the danger of replicating functionality across teams. The proximity and common culture of the offsite teams work against the chances of duplicating functionality. The participation of the members of the architecture and later integration teams in design reviews and software inspections helps detect replication early. Employee stealing is another effective mechanism to curb duplication.

Why Agility is the Right Path for Indian Suppliers

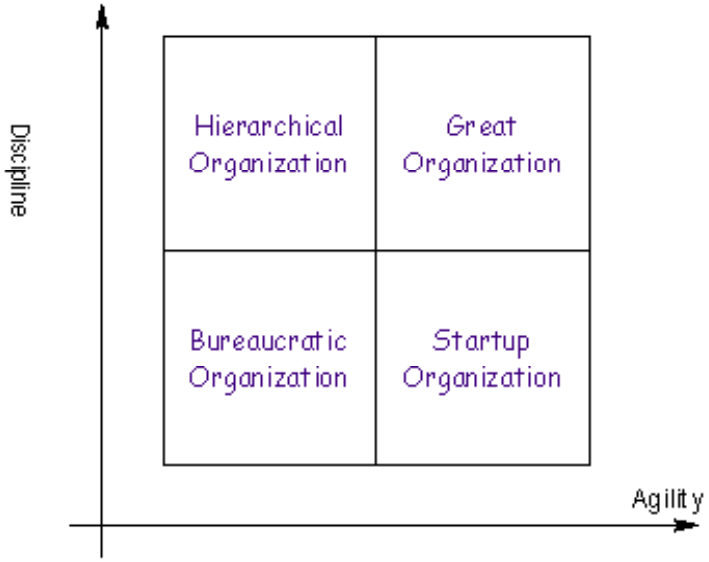
In his book *Good to Great*, Collins (2001) characterizes how self-discipline and entrepreneurial agility contribute to greatness at the corporate level (see Figure 7). Boehm and Turner (2004, p. 156) observe similarity between the patterns of creativity at corporate management and in software development, and apply Collins' method for performing self-assessment to software organizations.

Small Indian firms are mostly startup organizations, and agility improves their chances of survival. Both foreign equity participation companies and major Indian companies aim at the North-East quadrant in Figure 7, and therefore agility will be instrumental in attaining their goals.

The rapid growth of Indian outsourcing suppliers has created a dynamic labor market with a large demand for lead developers for new projects. The average turnover level exceeds 15%. Suffice it to mention that in some cases contracts have placed a liability on the supplier for certain levels of turnover in an attempt to protect the interests of the client. Agile outsourcing could break the impact of high turnover. Several core agile practices (e.g., preference for tacit knowledge, pair programming, and retrospection), along with employee stealing, allow new team members to be brought quickly up to speed.

Agile outsourcing could aid the transfer of domain knowledge to Indian suppliers. With more domain knowledge and the pressing need to communicate directly with customers, agile outsourcing would help organizations move farther away from consultants to directly supply end users.

Figure 7. Good-to-great matrix of creative discipline



The increased revenue per employee, arising from moving up the "value chain," could be partly offset by the increase in the onsite/offsite developers ratio. The small agile team size would allow small and medium enterprises to catch up with foreign equity participation companies and major Indian companies. As a result, small and medium companies could directly participate in complete service and product development.

Executable Domain Models

Below we discuss briefly executable domain models, which are believed to become mainstream in the near future (Booch, 2004; Mellor et al., 2004). Executable models are at the heart of the latest OMG initiative—the Model-Driven Architecture (OMG, 2004).

When working on a model or part of it, a developer must be able to answer the question, "Is this model correct?" Agile methods, through the "prove with code principle" and fast cycle, answer this question almost instantaneously, hence the shorter development times and the better customer satisfaction.

In our view, a problem with agile methods is the level at which the software artifacts are tested. When a system is designed in UML, it has to be tested in UML and not in the underlying source language. While, at times, it is necessary to drill down to the lowest level of detail (e.g., to stress-test the build), the majority of testing ought to be at model level, where the precious business logic and knowledge are captured.

In order for a model to be testable, it has to be executable, which means that the modeling language must have executable semantics. Executable UML (xUML) is a UML profile with precise action semantics (AS, 2001) that allows system developers to build executable system models, and then to map these models using code generators to source code for a target platform (Mellor & Balcer, 2002).

Executable domain models are expected to play an increasing role in software production. Executable domain models fit naturally with

the domain-based model organization of agile outsourcing projects.

CONCLUSION

In the late 1990s and early 2000s, agile methods took the imagination of software developers by storm. This fact clearly indicates that heavyweight methods have not been embraced wholeheartedly by developers and managers alike, and are found either impractical or costly (or both) in many environments. Driven by low labor costs for commensurate quality in India, and also by stagnant revenues in developed countries, firms have been increasingly outsourcing software production to India. In this chapter, we introduced a novel project structure creating agile conditions for large outsourcing software projects. The proposed structure is tailored to the needs of the Indian outsourcing suppliers. Several assumptions make agile outsourcing Indian-unique. First, we assume that there are large pools of qualified developers residing in one geographical area and even in one city. Second, the outsourcing country has grown a sustainable software industry that is supported by advanced information and telecommunication technologies, and led by talented managers, allowing it to secure large software projects.

We showed how to slice a large software project into multiple agile projects. We proposed to separate the development activities to R&D activities, carried out onsite, close to the client, and production activities, carried out offsite in India. The onsite, architecture team starts work on the project. In cooperation with the client, the architecture team develops the high-level use case model of the system and completes an architectural prototype with the strategic decisions for the overall system structure. The agile offsite teams are seeded with members of the architecture team and start functioning toward the end of the architectural elaboration. The "seed developers" transfer domain expertise to the supplier's site and act as customer surrogates to help reproduce agile conditions offsite. Outsourcing the entire set of production activities retains interactivity at the offsite location.

The domain-based package structure of the system model is organized by domains, subsystem use cases, and builds. This model organization scales up well to large-sized projects, because the subsystem packages can be recursively divided into smaller ones. The domain packages facilitate reuse of components and classes from different subject matters. The domain models could become software assets reused across projects.

To balance the teams' workloads and avoid replicating functionality by different teams, we proposed a new adaptive technique called employee stealing. Employee stealing is instrumental in disseminating tacit knowledge about the project and in lifting up the developers' expertise. It also improves the inter-team communication and coordination, and precludes the loss of common goal.

We plan on combining aspects of domain engineering and executable domain models in xUML with agile outsourcing to raise the level of abstraction and reuse, and to automate the repetitive tasks in the software process. We are especially interested in employing techniques preventing architectural degradation in large systems and in building "software factories" that produce families of sustainable software assets using highly automated processes. We would like to support the emerging "supply chains" for software development, which enables mass customization. To fully support domain-driven design, we ought to differentiate clearly between building a product platform and designing an application. Gearing executable domain models to the proposed agile outsourcing methodology will result in making agile outsourcing even more agile.

REFERENCES

Agile Alliance. (2001). Agile Alliance manifesto. Retrieved from *www.aanpo.org*

Akella, R., & Dossani, R. (2004). Private communication.

Arthur, B.W. (1994). *Increasing returns and path dependence in the economy.* Ann Arbor: University of Michigan Press.

AS. (2001). UML actions semantics. Retrieved from *www.omg.org*

Beck, K. (1999). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Bettin, J. (2004). Model-driven software development: An emerging paradigm for industrialized software asset development. Retrieved from *http://www.softmetaware.com*

Blumofe, R.D., & Leiserson, C.E. (1999). Scheduling multithreaded computations by work stealing. *Journal of ACM, 46*(5), 720-748.

Boehm, B., & Turner, T. (2004). *Balancing agility with discipline*. Boston: Addison-Wesley.

Booch, G. (2004). MDA: A motivated manifesto? *Software Development*, (August). Retrieved from *http://www.sdmagazine.com*

Brownsword, L., & Clements, P. (1996). *A case study in successful product line development.* Technical Report CMU/SEI-96-TR-035, Software Engineering Institute.

Collins, J. (2001). *Good to great.* New York: HarperCollins.

D'Costa, A.P. (2003). Uneven and combined development: Understanding India's software exports. *World Development, 31*(1), 211-226.

Dossani, R., & Kenney, M. (2003). Went for cost, stayed for quality?: Moving the back office to India. Asia-Pacific Research Center, Stanford University. Retrieved from *http://APARC.stanford.edu*

Evans, G. (2004). Agile RUP: Taming the Rationa® Unified Process®. In B. Roussev (Ed.), *Management of object-oriented software development*. Hershey, PA: Idea Group Inc.

Henderson-Sellers, B. (1996). *Object-oriented metrics, measures of complexity.* Englewood Cliffs, NJ: Prentice-Hall.

Jacobson, I. (1987). Object-oriented development in an industrial environment. *ACM SIGPLAN Notices, 22*(12), 183-191.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified software development process.* Boston: Addison-Wesley.

Kruchten, P. (2004). Scaling down large projects to meet the Agile "Sweet Spot." *The Rational Edge*, (August).

Mellor, S.J., Kendall, S., Uhl, A., & Weise, D. (2004). *MDA distilled*. Boston: Addison-Wesley.

Mellor, S.J., & Balcer, M.J. (2002). *Executable UML: A foundation for Model-Driven Architecture*. Boston: Addison-Wesley.

Nasscom. (2004). Indian software and services exports. Retrieved from *www.nasscom.org*

OMG. (2004). OMG Model-Driven Architecture. Retrieved from *http://www.omg.org/mda/*

Outsourcing Institute. (2004). Retrieved from *http://www.outsourcing.com*

Paulk, M.C., Curtis, B., Chrissis, M.B., & Weber, C.V. (1993). Capability maturity model, version 1.1. *IEEE Software, 10*(4), 18-27. Software Engineering Institute. (2004). Retrieved from *www.sei.cmu.edu*

Pollice, G. (2001). RUP and XP, part I: Finding common ground, and part II: Valuing differences. *The Rational Edge*.

Pollice, G. (2004). RUP and eXtreme Programming: Complementing processes. In B. Roussev (Ed.), *Management of object-oriented software development*. Hershey, PA: Idea Group Inc.

Roussev, B., & Wu, J. (2001). Task scheduling on NOWs using lottery-based workstealing. *Annual Review of Scalable Computing, 3*, World Scientific, Singapore University Press.

Royce, W. (2002). The case for results-based software management. *The Rational Edge*, (June).

SEI. (2004). Carnegie Mellon Software Engineering Institute, Product Line Practice. Retrieved from *www.sei.cmu.edu/productlines/*

Toth, K., Kruchten, P., & Paine, T. (1993). Modernizing air traffic control through modern software methods. *Proceedings of the 38th Annual Air Traffic Control Association Conference*, Nashville, TN.

ENDNOTES

1. In domain modeling, a software asset is anything from models, components, frameworks, generators, to languages and techniques.
2. Strategic assets are the software assets at the heart of a business—assets that grow into an active human- and machine-usable knowledge base about a business and its process.
3. A software liability is software that is cost burden—that is., software that costs more than it is delivering to the business (Bettin, 2004).
4. Mass customization meets the requirements of heterogeneous markets by producing goods and services to match individual customer's needs with near mass production efficiency.
5. One-off systems have a very short lifespan and are too expensive for most practical purposes.

This work was previously published in Management of the Object-Oriented Development Process, edited by B. Roussev & L. Liu, pp. 109-140, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 2.26
Decision Rule for Investment in Frameworks of Reuse

Roy Gelbard
Bar-Ilan University, Israel

ABSTRACT

Reuse helps to decrease development time, code errors, and code units. Therefore, it serves to improve quality and productivity frameworks in software development. The question is not HOW to make the code reusable, but WHICH amount of software components would be most beneficial, that is, cost-effective in terms of reuse, and WHAT method should be used to decide whether to make a component reusable or not. If we had unlimited time and resources, we could write any code unit in a reusable way. In other words, its reusability would be 100%. However, in real life, resources are limited and there are clear deadlines to be met. Given these constraints, decisions regarding reusability are not always straightforward. The current research focuses on decision-making rules for investing in reuse frameworks. It attempts to determine the parameters, which should be taken into account in decisions relating to degrees of reusability. Two new models are presented for decision-making relating to reusability: (i) a restricted model and (ii) a non-restricted model. Decisions made by using these models are then analyzed and discussed.

INTRODUCTION

Reuse helps decrease development time, code errors, and code units, thereby improving quality and productivity frameworks in software development. Reuse is based on the premise that educing a solution from the statement of a problem involves more effort (labor, computation, etc.) than inducing a solution from a similar problem for which such efforts have already been expended. Therefore, reuse challenges are structural, organizational, and managerial, as well as technical.

Economic considerations and cost-benefit analyses in general, must be at the center of any discussion of software reuse; hence, the cost-benefit issue is not HOW to make the code reusable, but WHICH amount of software components would be most beneficial, that is, cost-effective

for reuse, and WHAT method should be used when deciding whether to make a component reusable or not.

If we had unlimited time and resources, we could write any code unit in a reusable way. In other words, its reusability would be 100% (reusability refers to the degree to which a code unit can be reused). However, in real life, resources are limited and there are clear deadlines to be met. Given these constraints, reusability decisions are not always straightforward.

A review of the relevant literature shows that there are a variety of models used for calculating-evaluating reuse effectiveness, but none apparently focus on the issue of the degree to which a code is reusable. Thus, the real question is how to make reusability pragmatic and efficient, that is, a decision rule for investment in reuse frameworks. The current study focuses on the parameters, which should be taken into account when making reusability degree decisions. Two new models are presented here for reusability decision-making:

- A non-restricted model, which does not take into account time, resources, or investment restrictions.
- A restricted model, which takes the above-mentioned restrictions into account.

The models are compared, using the same data, to test whether they lead to the same conclusions or whether a contingency approach is preferable.

BACKGROUND

Notwithstanding differences between reuse approaches, it is useful to think of software reuse research in terms of attempts to minimize the average cost of a reuse occurrence (Mili, Mili, & Mili, 1995).

[Search* + *(*1-p*) * (ApproxSearch* +q ** Adaptation old **+ *(*1-q*)* Development*** new ***)]***

Where:

- ***Search*** (***ApproxSearch***) is the average cost of formulating a search statement of a library of reusable components and either finding one that matches the requirements exactly (appreciatively), or being convinced that none exists.
- ***Adaptation*** old is the average cost of adapting a component returned by approximate retrieval.
- ***Development*** new is the average cost of developing a component that has no match, exact or approximate, in the library.

For reuse to be cost-effective, the above must be smaller than:

p ****Development* exact **+*(*1-p*)* q * Development*** approx **+*(*1-p*)* (*1-q*)´ Development*** new***)***

Where:

- ***Development*** exact and ***development*** new represent the average cost of developing custom-tailored versions of components in the library that could be used as is, or adapted, respectively. Note that all these averages are time averages, and not averages of individual components, that is, a reusable component is counted as many times as it is used.

Developing reusable software aims at maximizing P (probability of finding an exact match) and Q (probability of finding an approximate match), that is, maximizing the coverage of the application domain and minimizing adaptation for a set of common mismatches, that is, packaging components in such a way that the most common old mismatches are handled easily. Increasing P

and Q does not necessarily mean putting more components in the library; it could also mean adding components that are more frequently needed, because adding components not only has its direct expenses (adaptation costs), but also increases search costs.

There are two main approaches to **code adaptation**: (1) identifying components that are generally useful and (2) covering the same set of needs with fewer components, which involves two paradigms: (i) abstraction and (ii) composition. Composition supports the creation of a virtually unlimited number of aggregates from the same set of components, and reduces the risk of combinatorial explosion that would result from enumerating all the possible configurations. In general, the higher the level of abstraction at which composition takes place, the wider the range of systems (and behaviours) that can be obtained. The combination of abstraction and composition provides a powerful paradigm for constructing systems from reusable components (Mili et al., 1995).

Frakes and Terry describe a wide range of metrics and adaptation models for software reuse. Six types of metrics and models are reviewed: cost-benefit models, maturity assessment models, amount of reuse metrics, failure modes models, reusability assessment models, and reuse library metrics (Frakes & Terry, 1996).

Other studies (Henninger, 1999; Otso, 1995; Virtanen, 2000; Ye, Fischer, & Reeves, 2000; Ye & Fischer, 2002), present additional metrics and methods, evaluate and make comparisons, but as is typical in an emerging discipline such as systematic software reuse, many of these metrics and models still lack formal validation. Despite this, they are used and are found useful in industrial practice (Ferri et al., 1997; Chaki, Clarke, Groce, Jha, & Veith, 2004).

Empirical work (Mens & Tourwé, 2004; Paulson, Succi, & Eberlein, 2004; Tomer, Goldin, Kuflik, Kimchi, & Schach, 2004; Virtanen, 2001; Ye, 2002) has analyzed existing reuse metrics and their industrial applicability. These metrics are then applied to a collection of public domain software products and projects categories to assess the level of correlation between them and other well-known software metrics such as complexity, volume, lines of code, and so forth.

Current research is focused on decision-making rules for investment in reuse frameworks. The well-known "simple model" and "development cost model" deal with these decisions, but do not take into account restrictions and constraints such as time, budget, resources, or other kinds of investment, such as delivery time, that may impact on the decision to reuse.

ANALYZING NEW REUSE MODELS

Assume a software development project contains 3 code components: A, B, and C, and we need to determine two things: Which of these components should be reusable? What criteria should be taken into account?

There are eight combinations—choice alternatives for these 3 components, as shown in Table 1 (+ represents "make reusable," - represents "don't make reusable").

A. The Non-Restricted Model

The model contains the following parameters:

- **Ci—**cost of creating component **i** from scratch (without making it reusable).
- **Ri**—cost of making component **i** reusable (extra costs – not included in **Ci**).
- **ICi**—cost of implementing reusable component **i** into code.
- **NRi**—number of reuses of component i. (C, R, and NR are in man-hours).

Savings resulting from making component **i** reusable are represented as follows:

Table 1. Choice alternatives

Alternative	Component A	Component B	Component C
1	-	-	-
2	+	-	-
3	-	+	-
4	+	+	-
5	-	-	+
6	+	-	+
7	-	+	+
8	+	+	+

SAVi = NRi *(Ci – ICi) – (Ci + Ri)

Therefore: If **SAVi** > 0, it is worthwhile to make component **i** reusable.

Suppose a company that employs two kinds of programmers: **M** and **N**. Programmers of type **M** are permanent employees of the firm. Programmers of type **N** are highly qualified consultants who are employed by the company for specific projects. The company is going to write/create/develop a new project, and has to make a decision regarding which components should be reusable.

The following are additional parameters:

- **Cim**—hours needed for programmer M to create component **i** from scratch.
- **Rim**—hours needed for programmer M to make component **i** reusable.
- **ICim**—hours needed for programmer M to implement reusable component **i** into code.
- **Sm**—costs of programmer M, per 1 hour.

Hence:

Ci= Min(Cim*Sm, Cin*Sn)
Ici= Min(ICim*Sm, ICin*Sn)
Ri= Min(Rim*Sm, Rin*Sn)

Hence:

SAVi = NRi *(Min(Cim*Sm, Cin*Sn) – Min(ICim*Sm, ICin*Sn)) – (Min(Cim*Sm, Cin*Sn) + Min(Rim*Sm, Rin*Sn))

B. The Restricted Model

The non-restricted model has the following limitations:

- It requires absolute values
- It is quite difficult to measure parameters such as: **Ci**, **Ri,** and **Ici**
- It does not take into account the most typical situation where time and budget are restricted as well as in-house investment in reuse, that is, time and resources for reusable code developing.

In order to avoid these limitations, the restricted model is based upon the following parameters:

- **I**—maximal **investment** that can be allocated for writing a reusable code.
- **T**—maximal calendar **time** that can be allocated for writing a reusable code.
- **Ii**—percent of "**I"** needed to make component **i** reusable.
- **Ti**—percent of "**T"** needed to make component **i** reusable.
- **Ci**—relative **complexity** of creating component **i** from scratch.
- **Fi—frequency** (%) of future projects that are likely to reuse component **i.**
- **Pi**—relative **profit** of making component **i** reusable.
- $\mathbf{R_I}$**—r**emainder of "**I"**, after some reusable components have been written.
- $\mathbf{R_T}$**—**remainder of "**T**," after some reusable components have been written.

Assume that: **Pi** = **Ci** * **Fi.**

Hence: component **i** is the next component to be made reusable if:

Pi = Max(P1, P2, ..., Pn-1, Pn)
Ii <= **RI**
Ti <= RT

C. Illustrative Example: Non-Restricted Model

The following example (Example 1) demonstrates the decision made by the non-restricted model. Assume we want to develop 10 projects, each one containing components A, B, and C according to Table 2.

Hence**: NRa** = 10, **NRb** = 1, **NRc** = 4

Table 3 presents illustrative assumptions concerning **Cim** and **Cin** (hours needed for programmer type M and N to create component **i** from scratch).

Moreover, assume programmers' costs to be: **Sm** = 20, **Sn** = 40

Hence:

Table 2. Example 1, number of components for future reuse

Project	1	2	3	4	5	6	7	8	9	10
Component A	+	+	+	+	+	+	+	+	+	+
Component B	+									
Component C	+	+	+	+						

Table 3. Example 1, Ci illustrative assumptions

Programmer type	Component A	Component B	Component C
Type M	*300*	*20*	*150*
Type N	*200*	*10*	*100*

Ca = Min(300*20, 200*40) = 6,000
Cb = Min(20*20, 10*40) = 400
Cc = Min(150*20, 100*40) = 3,000

Table 4 presents illustrative assumptions concerning **Rim** and **Rin** (hours needed for programmers type M and N to make component **i** reusable).

Hence:

Ra = Min(650*20, 300*40) = 12,000
Rb = Min(15*20, 7*40) = 280
Rc = Min(150*20, 80*40) = 3,000

Table 5 presents illustrative assumptions concerning **ICim** and **ICin** (hours needed for programmers type M /N to implement reusable component **i** into code).

Hence:

ICa = Min(60*20, 15*40) = 600
ICb = Min(5*20, 3*40) = 100
ICc = Min(50*20, 10*40) = 400

Hence:

SAVa = 10 *(6,000 – 600) – (6,000 + 12,000) = 36000 > 0
SAVb = 1 *(400 – 100) – (400 + 280) = -380 < 0
SAVc = 4 *(3000– 400) – (3,000+ 3,000) = 4400 > 0

In light of the mentioned, the reuse decision according to the non-restricted model is to make components A and C reusable (i.e., alternative 6).

D. Illustrative Example: Restricted Model

The following example (Example 2) demonstrates the decision made by the restricted model, based on the previous example (Example 1). Assume the following:

1. **I**—10,000.
2. **T**—150. The available remaining time to make the existing code reusable.

Table 4. Example 1, Ri illustrative assumptions

Programmer type	Component A	Component B	Component C
Type M	*650*	*15*	*150*
Type N	*300*	*7*	*80*

Table 5. Example 1, ICi illustrative assumptions

Programmer type	Component A	Component B	Component C
Type M	*60*	*5*	*50*
Type N	*15*	*3*	*10*

Table 6. Parameters used by Example 2

Component	Ci	Fi(%)	Pi	Ii(%)	Ti(100%)
A	*30*	*100*	*30*	*120*	*200*
B	*1*	*10*	*0.1*	*2.8*	*4.7*
C	*15*	*40*	*0.6*	*30*	*100*

3. **Ci**—assume component B is the easiest one to develop, and requires 10 hours. Assume component A requires 300 hours and component C requires 150 hours. Hence, complexities are: C_A=30, C_B=1, C_C =15.
4. **Fi**—component A will be reused by 100% of future projects, B by 10%, and C by 40%.
5. $\mathbf{I_A}$ = 12,000/10,000 = 120%, $\mathbf{I_B}$ = 280/10,000 = 2.8%, $\mathbf{I_C}$ = 3000/10,000 = 30%.
6. $\mathbf{T_A}$ = 300/150 = 200%, $\mathbf{T_B}$ = 7/150 = 4.7%, $\mathbf{T_C}$ = 150/150 = 100%.

Hence Example 2 parameters are shown in Table 6.

Taking time and investment restrictions into account, the reuse decision, according to the restricted model is to make only component C reusable (i.e., alternative 5).

CONCLUSION AND FUTURE TRENDS

The current study presented two new reuse decision making models: a restricted model and a non restricted model, which mainly differ in the way they take into account real-life constraints-restrictions such as time, budget, and resources repetition.

The models produced different results from the same data. The decision made by the restricted model pinpointed fewer software components for reuse. It is worth mentioning that different groups of software components were not the issue, but rather different subgroups of the same group, that is, software components selected by the restricted model were subgroups of components selected by the non-restricted model.

Moreover, the parameters of the restricted model relate to relative value arguments, by contrast to the parameters of non-restricted model, which relate to absolute values. While absolute values are difficult to measure, relative values are simpler to define. There are a variety of formal methods by which relative values may be defined, methods that are used in other areas of software engineering, such as cost estimation, effort estimation, priority decision, and others.

The reusability decision made by the restricted model may be biased by the following parameters: time, resources, component complexity, and number-percent of future projects in which the component would be reused. Further research should be conducted, focusing on decision robustness in light of the mentioned parameters and their possible spectrum.

ACKNOWLEDGMENT

I would like to thank Tami Shapiro for her contribution to this work.

REFERENCES

Chaki, S., Clarke, E. M., Groce, A., Jha, S., & Veith, H. (2004). Modular verification of software components. *IEEE Transactions on Software Engineering, 30*(6), 388-402.

Desouza, K. C., Awazu, Y., & Tiwana, A. (2006). Four dynamics for bringing use back into software reuse. *Communications of the ACM, 49*(1), 96-100.

Ferri, R. N., Pratiwadi, R. N., Rivera, L. M., Shakir, M., Snyder, J. J., Thomas, D. W. et al. (1997). Software reuse: Metrics for an industrial project. In *Proceedings of the 4th International Symposium of Software Metrics* (pp.165-173).

Fischer, G., & Ye, Y. (2001). Personalizing delivered information in a software reuse environment. In *Proceedings of the 8th International Conference on User Modeling* (p. 178).

Frakes, W., & Terry, C. (1996). Software reuse: Metrics and models. *ACM Computing Surveys, 28*(2), 415-435.

Henninger, S. (1999). An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology, 6*(2), 111-140.

Kirk, D., Roper, M., & Wood, M. (2006). Identifying and addressing problems in object-oriented framework reuse. *Empirical Software Engineering, 12*(3), 243-274.

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering, 30*(2), 126-139.

Mili, H., Mili, F., & Mili, A. (1995). Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering, 21*(6), 528–562.

Otso, K. J. (1995). *A systematic process for reusable software component selection* (Tech. Rep.). University of Maryland.

Paulson, J. W., Succi, G., & Eberlein, A. (2004). An Empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering, 30*(4), 246-256.

Reifer, D. J. (1997). *Practical software reuse.* Wiley.

Spinellis, D. (2007). Cracking software reuse. *IEEE Software, 24*(1), 12-13.

Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., & Schach, S. R. (2004). Evaluating software reuse alternatives: A model and its application to an industrial case study. *IEEE Transactions on Software Engineering, 30*(9), 601-612.

Virtanen, P. (2000). Component reuse metrics—Assessing human aspects. In *Proceedings of the ESCOM-SCOPE* (pp. 171-179).

Virtanen, P. (2001). Empirical study evaluating component reuse metrics. In *Proceedings of the ESCOM* (pp. 125-136).

William, B., Frakes, W. B. & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering, 31*(7), 529-536.

Ye, Y. (2002). An empirical user study of an active reuse repository system. In *Proceedings of the 7th International Conference on Software Reuse* (pp. 281-292).

Ye, Y., & Fischer, G. (2002). Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the International Conference on Software Engineering* (pp. 513-523).

Ye, Y., Fischer, G., & Reeves, B. (2000). Integrating active information delivery and reuse repository systems. In *Proceedings of ACM-SIGSOFT*

8th International Symposium on Foundations of Software Engineering (pp. 60-68).

KEY TERMS

Decision Rule: Either a formal or heuristic rule used to determine the final outcome of the decision problem.

Non-Restricted Reuse-Costing Model is a reuse-costing model that does not take into account real-life constraints and restrictions, such as time, budget, resources, or any other kind of investment.

Reuse: Using an item more than once. This includes conventional reuse where the item is used again for the same function and new-life reuse where it is used for a new function (wikipedia).

Reuse-Costing Model: A formal model, which takes into account the expenditure to produce a reusable software product.

Restricted Reuse-Costing Model: A reuse-costing model that takes into account real-life constraints and restrictions such as time, budget, resources, or any other kind of investment.

Software Reuse: Also called code reuse, is the use of existing software components (e.g., routines, functions, classes, objects, up to the entire module) to build new software.

This work was previously published in Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, edited by M. Syed & S. Syed, pp. 140-147, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 2.27
Integrated Product Life Cycle Management for Software: CMMI[1], SPICE, and ISO/IEC 20000

Dirk Malzahn
OrgaTech GmbH, Germany

ABSTRACT

This chapter describes how models for software development and service delivery can be integrated into a common approach to reach an integrated product life cycle for software. The models covered by this chapter are the capability maturity model integration (CMMI), SPICE (software process improvement and capability determination, ISO 15504) and ISO 20000 (service management). Whilst the CMMI constellation approach delivers an integration perspective defined in three models (development, acquisition and services), SPICE and ISO 20000 need additional alignment to be usable in an integrated approach.

INTRODUCTION

The focus of the market for IT solutions has changed. Whilst many companies and organizations followed the latest "hype" several years ago, they now trust in reliable and sustainable solutions.

To ensure this, standardization of quality evaluation becomes more and more important. For supplier selection, make-or-buy decisions and outsourcing strategies, a powerful set of procedures, that can help to assess the capability of internal and external software processes, is required. These procedures have to be based on best practices and must be widely accepted.

On this basis, standards offer the best possibilities: they are usually defined by a wide group of experts, which all contribute their experiences and best practices. Standards are either sponsored by an industry or by national bodies—therefore making these standards de facto mandatory for an industry, nation, or combination of both enforces the acceptance. If a significant group uses a standard, market dynamics have an additional impact. Official certificates, levels, and so forth can be and are used for marketing activities.

In the field of software related standards, lots of different standards have been defined for special topics, but one standard is still missing: a standard that covers a software product from the very beginning—the first idea—up to the very end—the retirement of the software.

On the one hand powerful standards, for example the capability maturity model integration (CMMI) or SPICE (ISO 15504), have been defined for software development. On the other hand, standards for service delivery, for example ITIL or ISO 20000, have been well established; but there is still a wall between the worlds of software development and service delivery. Even though some standards – like SPICE – take a look over the wall, an integrated approach has not been delivered yet.

The need for this integration is obvious. A customer is not interested in having some quality for development and some other quality for service delivery—the customer needs one quality approach that covers the full life cycle of a software product.

BACKGROUND

The Wall Between Software Development and Service Delivery

When IT systems are planned, the focus of the planning is mostly restricted to software development. Topics like operation environment or data management are discussed, but the strategy usually ends with the delivery of the software product.

On the other hand, service-delivering organizations mostly just provide "services" and are not really interested in the software development process.

This behavior leads to multiple difficulties and inefficiencies:

- Software developers and service people do not understand each other. They work in different worlds and have their own "language" and processes.
- The efficiency and effectiveness of service delivery highly depends on the architecture of and assumptions for the software, therefore the service organization has to be integrated early into the software development.
- Service level agreements can be optimized, when both sides reach a common understanding. The development of service level agreements is often based on the "what we need" position of both sides and not on the "what will be best for the customer" position.
- Problem Management is not transparent to the customer. The customer is not interested whether he has a service problem or a software problem—the customer wants a quick and reliable solution. If the software side does not understand the service side, problems often become ping-pong balls.
- Software usually lives longer than the original developer intends. Systems often have to be enhanced just to fulfil the requirements of a new service platform. If this is not taken into account when the software is developed, the effort for updating software may become enormous. Sometimes software has to be retired, just because it is not executable on the new platform!
- New approaches like service oriented architectures (SOA) demand the high integration of software and service elements. Future trends will rather lead to small combined software/service environments than to big software solutions operated by massive computer environments.

Just to ensure that I am not misunderstood: software developing and service delivering orga-

nizations will still deliver and operate solutions with high quality—but the will not do it in the most efficient and effective way. Organizations and companies aiming for the delivery of sustainable and reliable solutions have to ensure, that the solutions are not only developed in the best way but will meet the requirements of the future efficiently, effectively and still with high quality.

Standards for Software Development and Service Delivery

Regarding software development, two standard are widely accepted all over the world: the Capability Maturity Model Integration, published by the Software Engineering Institute (SEI) at the Carnegie Mellon University, and SPICE (Software Process Improvement and Capability Determination), which is published as ISO standard 15504. Both standards define a process framework based on best practices and provide an assessment model to evaluate process capability.

A process reference model (PRM) and a process assessment model (PAM) usually characterize a process framework. The PRM defines processes that have shown evidence to support high quality for the defined domain – in our case software development. The PAM builds the basis for collecting evidence that the PRM is adhered to and to evaluate the capability of the processes defined in the PRM.

In the world of service delivery, ITIL (IT Infrastructure Library) is the most acknowledged standard. To make ITIL assessable, the ISO 20000 standard was developed.

But as it was said before, none of these standards cover the complete software life cycle. This gap is seen by the SEI and discussed by relevant contributors to the ISO standards. In this article two approaches for this integration are discussed—one based on the CMMI and the other based on the connection of SPICE and ISO 20000.

CMMI INTEGRATION PERSPECTIVE

CMMI is one of the best-established process frameworks for software development. Starting with the publication of Watts Humphreys book "Managing the Software Process" in 1989, the CMM and its successor—the CMMI—nearly became a synonym for process improvement in the software world.

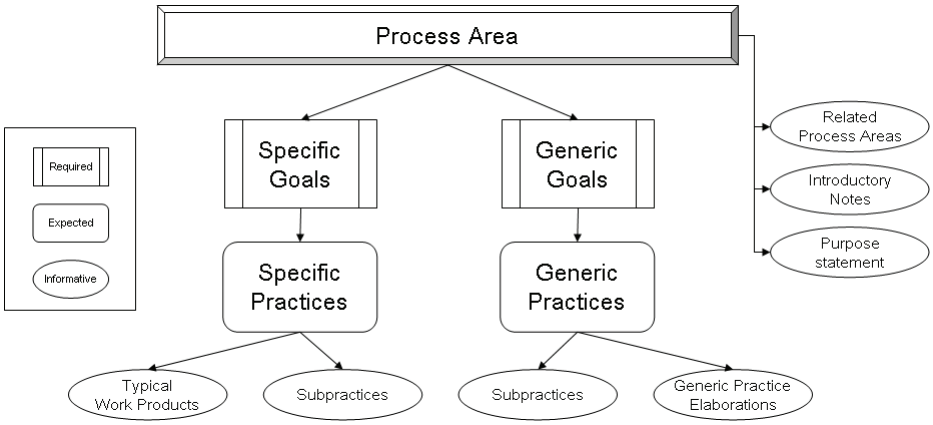
Nevertheless, until 2006, the CMMI was restricted on software and systems development. Since then a new initiative has been started to further develop the CMMI in the direction of acquisition and service processes. If one is talking about "CMMI" mostly the CMMI-DEV (for Development) is meant. This is the classical CMMI stream that covers service development. Other ideas concerning the usage and benefits of the CMMI for organizations that only acquire software, lead to the CMMI-ACQ (for Acquisition), which was published in November 2007. In 2006 and 2007 a CMMI for Services (CMM-SVC) has been developed to close the gap between software development and service delivery. The CMMI-SVC awaits publication in 2008.

CMMI Basics

Independent from the different CMMIs each of these models follows the same structure as shown in Figure 1.

In each CMMI a set of process areas is defined. For each process area the purpose is described, some introductory notes are given and other process areas which have a relation to this process area are listed. To satisfy the process area, goals must be fulfilled. Each process area has specific goals which are unique for this process area. Additional generic goals are defined, which are common for all process areas. For each goal a set of practices is defined, which are considered important for reaching the goal. For each practice

Figure 1. CMMI structure



subpractices, typical work products and elaborations are defined.

From an evaluation perspective, only the goals are required. To reach a so-called "level" an organization has to fulfil the goals of the processes for this level. Nevertheless it is expected, that the defined practices are also fulfilled.

Process Capability vs. Organization Maturity

Talking about the CMMI always means talking about "level". But before the different levels can be explored, the different representations have to be discussed. CMMI knows two different representations, staged and continuous. In the staged representation each process area is assigned to a specified maturity level. To reach a maturity level, all specific goals of the assigned processes and a subset of generic goals have to be fulfilled.

In the continuous representation, each process area is evaluated separately by fulfilling generic goals for this process area. Based on the set of generic goals that are fulfilled, the capability of the process area is measured.

In both representations, the generic goals have high importance. In total 5 generic goals are defined:

- **GG1:** The process supports and enables achievement of the specific goals of the process area by transforming identifiable input work products to produce identifiable output work products[2].
- **GG2:** The process is institutionalized as a managed process.
- **GG3:** The process is institutionalized as a defined process.
- **GG4:** The process is institutionalized as a quantitatively managed process.
- **GG5:** The process is institutionalized as an optimizing process.

In the staged representation 5 levels are defined. The lowest level is level 1, which means that the processes of the organization are still over-

whelmingly chaotic. Goals for process maturity start with the definition of maturity level 2. For this level the specific goals of the assigned process areas and generic goal 2 have to be satisfied. For maturity level 3 the specific goals of the process areas assigned to level 2, the specific goals of the process areas assigned to level 3 and the generic goals 2 and 3 have to be satisfied. For level 4 and 5 specific goals of other process areas are added, but even for these higher levels, only generic goal 2 and 3 have to be satisfied.

In the continuous representation, 6 levels are defined, starting with level 0 and ending with level 5. Remembering, that a capability level is evaluated for each process area, the level is given by the fulfilment of the generic goals. On level 0 no generic goal is satisfied. On level 1, GG1 has to be satisfied, on level 2 GG1 and GG2 have to be satisfied and so on.

As this article is focused on the integration of activities and not mainly on evaluation issues, we will restrict the following chapters to the parts specific to certain process areas. For more information on the generic parts and evaluation procedures the CMMI itself should be used (Software Engineering Institute, 2006a).

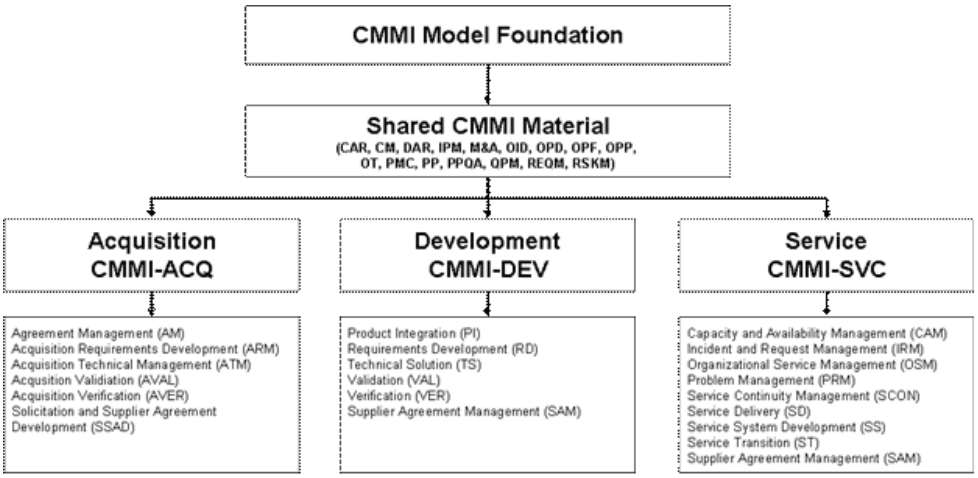
The Constellation Approach

As said before, there is not the "one" CMMI. The idea of the current CMMI version is to have different constellations which share some process areas that are common for all constellations and have additional process areas that are unique for the constellation.

Currently, three constellations are published or planned to be published in 2007:

- **CMMI-DEV:** For development (June 2006) (Software Engineering Institute, 2006-1)
- **CMMI-ACQ:** For acquisition (November 2007) (Software Engineering Institute, 2007)
- **CMMI-SCV:** For services (2008, initial draft September 2006) (Software Engineering Institute, 2006b)

Figure 2. CMMI constellation approach



These constellations only differ in the number and content of process areas. All other contents of the CMMI, for example generic elements, level definitions, typographical conventions, build the model's foundation and are identical for all constellations.

In a first overview the constellation approach can be structured as shown in Figure 2.

The purposes of all process areas are explained in the next chapter.

Shared Process Areas

Causal Analysis and Resolution

The purpose of causal analysis and resolution (CAR) is to identify causes of defects and other problems and take action to prevent them from occurring in the future.

Configuration Management

The purpose of configuration management (CM) is to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits.

Decision Analysis and Resolution

The purpose of decision analysis and resolution (DAR) is to analyze possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria.

The IPPD Addition (Integrated Product and Process Development)

Before describing the next process area, a new term has to be defined: the "IPPD addition". In CMMI, "additions" are used to include material that may be of interest to particular users. The IPPD group of additions covers an IPPD approach that includes practices that help organizations achieve the timely collaboration of relevant stakeholders throughout the life of the product to satisfy customers' needs, expectations, and requirements (Department of Defense, 1996). If you apply the CMMI, you are free to add the IPPD addition or not.

Integrated Project Management + IPPD

The purpose of integrated project management (IPM) is to establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process that is tailored from the organization's set of standard processes.

IPPD Addition: For IPPD, integrated project management +IPPD also cover the establishment of a shared vision for the project and the establishment of integrated teams that will carry out objectives of the project.

Measurement and Analysis

The purpose of measurement and analysis (MA) is to develop and sustain a measurement capability that is used to support management information needs.

Organizational Innovation and Deployment

The purpose of organizational innovation and deployment (OID) is to select and deploy incremental and innovative improvements that measurably improve the organization's processes and technologies. The improvements support the organization's quality and process-performance objectives as derived from the organization's business objectives.

Organizational Process Definition + IPPD

The purpose of organizational process definition (OPD) is to establish and maintain a usable set of organizational process assets and work environment standards.

IPPD Addition: For IPPD, organizational process definition +IPPD also cover the establishment of organizational rules and guidelines that enable conducting work using integrated teams.

Organizational Process Focus

The purpose of organizational process focus (OPF) is to plan, implement, and deploy organizational process improvements based on a thorough understanding of the current strengths and weaknesses of the organization's processes and process assets.

Organizational Process Performance

The purpose of organizational process performance (OPP) is to establish and maintain a quantitative understanding of the performance of the organization's set of standard processes in support of quality and process-performance objectives, and to provide the process-performance data, baselines, and models to quantitatively manage the organization's projects.

Organizational Training

The purpose of organizational training (OT) is to develop the skills and knowledge of people so they can perform their roles effectively and efficiently.

Project Monitoring and Control

The purpose of project monitoring and control (PMC) is to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan.

Project Planning

The purpose of project planning (PP) is to establish and maintain plans that define project activities.

Process and Product Quality Assurance

The purpose of process and product quality assurance (PPQA) is to provide staff and management with objective insight into processes and associated work products.

Quantitative Project Management

The purpose of quantitative project management (QPM) is to quantitatively manage the project's defined process to achieve the project's established quality and process-performance objectives.

Requirements Management

The purpose of requirements management (REQM) is to manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products.

Risk Management

The purpose of risk management (RSKM) is to identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives.

Acquisition Process Areas

Solicitation and Supplier Agreement Development

The purpose of solicitation and supplier agreement development (SSAM) is to prepare a solicitation package and to select one or more suppliers for delivering the product or service and establish and maintain the supplier agreement.

Acquisitions Management

The purpose of agreement management (AM) is to ensure that the supplier and the acquirer perform according to the terms of the supplier agreement

Acquisition Requirements Development

The purpose of the acquisition requirements development (ARD) is to produce and analyze customer and contractual requirements.

Acquisition Technical Management

The purpose of the acquisition technical management (ATM) is to evaluate the supplier's technical solution and to manage selected interfaces of that solution.

Acquisition Validation

The purpose of the acquisition validation (AVAL) is to demonstrate that an acquired product or service fulfils its intended use when placed in its intended environment.

Acquisition Verification

The purpose of acquisition verification (AVER) is to ensure that selected work products meet their specified requirements.

Development Process Areas

Product Integration

The purpose of product integration (PI) is to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product.

Requirements Development

The purpose of requirements development (RD) is to produce and analyze customer, product, and product component requirements.

Supplier Agreement Management

The purpose of supplier agreement management (SAM) is to manage the acquisition of products from suppliers.

Technical Solution

The purpose of technical solution (TS) is to design, develop, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product-related lifecycle processes either singly or in combination as appropriate.

Validation

The purpose of validation (VAL) is to demonstrate that a product or product component fulfils its intended use when placed in its intended environment.

Verification

The purpose of verification (VER) is to ensure that selected work products meet their specified requirements.

Service Process Areas

As mentioned before, the CMMI-SVC has not been published yet. Nevertheless, it is available as initial draft (Software Engineering Institute, 2006b) and was already widely discussed on several conferences (Hollenbach & Buteau, 2006).

Capacity and Availability Management

The purpose of capacity and availability management (CAM) is to plan and monitor the effective provision of resources to support service requirements.

Incident and Request Management

The purpose of the incident and request management (IRM) process area is to ensure the timely resolution of requests for service and incidents that occur during service delivery.

Organizational Service Management*

The purpose of the organizational service management (OSM) process area is to establish and maintain standard services that ensure the satisfaction of the organization's customer base.

Problem Management

The purpose of the problem management (PRM) process area is to prevent incidents from recurring by identifying and addressing underlying causes of incidents.

Service Continuity*

The purpose of the service continuity (SCON) is to establish and maintain contingency plans for continuity of agreed services during and following any significant disruption of normal operations.

Service Delivery

The purpose of the service delivery (SD) process area is to deliver services in accordance with service agreements.

Service System Development*

The purpose of the service system development (SSD) process area is to analyze, design, develop, integrate, and test service systems to satisfy existing or anticipated service agreements.

Service Transition

The purpose of the service transition (ST) process area is to deploy new or significantly changed service systems while managing their effect on ongoing service delivery.

Supplier Agreement Management

The purpose of supplier agreement management (SAM) is to manage the acquisition of products from suppliers.

The process areas marked with an asterisk (*) are additions (like IPPD in the CMMI for Development) and therefore optional.

CMMI Process Categories

In order to develop a better understanding for the dependencies between the process areas, the CMMI-DEV defines 4 categories which are applied to the other constellations below, and collect process areas with a similar focus. Therefore, 3 categories are identical for all constellations. These are

- Process management
- Project management
- Support

Table 1. Shared process areas

	Process Management	**Project Management**	**Support**	**Acquisition / Engineering / Service Establishment and Delivery**
Shared process areas	Organizational Innovation and Deployment, Organizational Process Definition +IPPD, Organizational Process Focus, Organizational Process Performance, Organizational Training	Project Monitoring and Control, Project Planning, Quantitative Project Management, Risk Management	Causal Analysis and Resolution, Configuration Management, Decision Analysis and Resolution, Measurement and Analysis, Process and Product Quality Assurance	Requirements Management

Table 2. CMMI-ACQ process areas

	Process Management	**Project Management**	**Support**	**Acquisition / Engineering / Service Establishment and Delivery**
CMMI-ACQ				Solicitation and Supplier Agreement Development, Agreement Management, Acquisition Requirements Development, Acquisition Technical Management, Acquisition Validation, Acquisition Verification

The fourth category is focussed on the field of application of the constellation and is labelled

- Acquisition (in CMMI-ACQ)
- Engineering (in CMMI-DEV)
- Service Establishment and Delivery (in CMMI-SVC)

Based on this categorization the complete set of CMMI process areas can be categorized as follows.

For shared process areas see Table 1; for CMMI-ACQ see Table 2; for CMMI-DEV see Table 3; for CMMI-SVC see Table 4.

Table 3. CMMI-DEV process areas

	Process Management	**Project Management**	**Support**	**Acquisition / Engineering / Service Establishment and Delivery**
CMMI-DEV		Supplier Agreement Management		Product Integration, Requirements Development, Technical Solution, Validation, Verification

Table 4. CMMI-SVC process areas

	Process Management	**Project Management**	**Support**	**Acquisition / Engineering / Service Establishment and Delivery**
CMMI-SVC	Organizational Service Management	Supplier Agreement Management, Capacity and Availability Management	Problem Management	Incident and Request Management, Service Continuity, Service Delivery, Service System Development, Service Transition

Constellation Based Maturity Levels

As described before, each process area is assigned to a defined maturity level. Whilst on level 4 and 5 only shared process areas are assigned, the assignment on level 2 and 3 is dependent on the constellation (see Table 5).

The shared processes assigned to level 4 are

- Organizational process performance
- Quantitative project management

Table 5. Maturity level 2 and 3 for CMMI constellations

	Maturity Level 2	Maturity Level 3
Shared process areas	Requirements Management, Project Planning, Project Monitoring and Control, Measurement and Analysis, Process and Product Quality Assurance, Configuration Management	Decision Analysis and Resolution, Integrated Project Management +IPPD, Organizational Process Definition +IPPD, Organizational Process Focus, Organizational Training, Project Management
CMMI-ACQ	Solicitation and Supplier Agreement Development, Agreement Management, Acquisition Requirements Development	Acquisition Technical Management Acquisition Validation, Acquisition Verification
CMMI-DEV	Supplier Agreement Management	Product Integration, Requirements Development, Technical Solution, Validation, Verification
CMMI-SVC	Supplier Agreement Management, Incident and Request Management	Capacity and Availability Management, Service Continuity Service Delivery, Service System Development, Service Transition, Organizational Service Management, Problem Management

The shared processes assigned to level 5 are

- Causal Analysis and Resolution
- Organizational Innovation and Deployment

A CMMI Based, Integrated Product Life Cycle

Even though the CMMI provides 3 different constellations, these constellations can be used to define 2 different integrated product life cycles:

- Organizations which provide service delivery for acquired software products should use the process areas of the CMMI-ACQ in combination with the process areas of the CMMI-SVC.
- Organizations which develop software and provide service delivery should use the process areas of the CMMI-DEV in combination with the process areas of the CMMI-SVC.

Even the combination of all three constellations is thinkable, when a service delivering organization partially acquires and partially develops the software.

SPICE / ISO 20000 INTEGRATION PERSPECTIVE

Besides the CMMI world, another possibility for an integrated product life cycle is the combination of two ISO standards: the ISO 15504—better known as SPICE—and the ISO 20000.

SPICE Basics

The ISO 15504 (SPICE) is structured in 5 parts. Part 1 defines the basic concept and the vocabulary. In part 2 rules for performing an assessment are defined, and in part 3 guidance for the assessment is given. Part 4 gives additional guidance on the use for process improvement and capability determination.

The interesting part under the integration perspective is part 5. This part defines an exemplar process assessment model.

Whilst the capability determination is widely similar to the approach of the CMMI continuous representation, the process model is different from the CMMI.

SPICE defines 3 categories. These categories are structured in groups and each group has several processes (ISO/IEC, 2006). The categories with their groups are:

- Primary life cycle processes
  - Acquisition process group (ACQ)
  - Supply process group (SPL)
  - Engineering process group (ENG)
  - Operation process group (OPE)
- Organizational life cycle processes
  - Management process group (MAN)
  - Process improvement process group (PIM)
  - Resource and infrastructure process group (RIN)
  - Reuse process group (REU)
- Supporting life cycle processes
  - Supporting process group (SUP)

Comparing the SPICE categories with the CMMI constellations, strong connections can be identified between the process groups of the primary life cycle processes and the CMMI constellations. The acquisition process group and supply process group have the same focus as the CMMI-ACQ as well as the engineering process group and the CMMI-DEV. Only the CMMI-SVC does not have a counterpart in SPICE. The operation processes group and some processes of the supporting process group address service related topics, but a common approach is not delivered.

To better understand the content of the SPICE process groups each group with its processes should be further described—as defined in SPICE (ISO/IEC, 2006):

Primary Life Cycle Processes

The primary life cycle processes consist of processes that serve primary parties during the life cycle of software. A primary party is one that initiates or performs the development, operation, or maintenance of software products. These primary parties are the acquirer, the supplier, the developer, the operator, and the maintainer of software products.

- The acquisition process group (ACQ) consists of processes performed by the customer, in order to acquire a product and/or a service. The processes of this group are:
  - ACQ.1: Acquisition preparation
  - ACQ.2: Supplier selection
  - ACQ.3: Contract agreement
  - ACQ.4: Supplier monitoring
  - ACQ.5: Customer acceptance
- The supply process group (SPL) consists of processes performed by the supplier in order to propose and deliver a product and/or a service. The processes of this group are:
  - SPL.1: Supplier tendering
  - SPL.2: Product release
  - SPL.3: Product acceptance support
- The engineering process group (ENG) consists of processes that directly elicit and manage the customer's requirements, specify, implement, and/or maintain the software product and it's relation to the system. The processes of this group are:
  - ENG.1: Requirements elicitation
  - ENG.2: System requirements analysis
  - ENG.3: System architectural design
  - ENG.4: Software requirements analysis
  - ENG.5: Software design
  - ENG.6: Software construction
  - ENG.7: Software integration
  - ENG.8: Software testing
  - ENG.9: System integration
  - ENG.10: System testing
  - ENG.11: Software installation
  - ENG.12: Software and system maintenance
- The operation process group (OPE) consists of processes performed in order to provide for the correct operation and use of the software product and/or service. The processes of this group are:
  - OPE.1: Operational use
  - OPE.2: Customer support

Organizational Life Cycle Processes

The organizational life cycle processes consist of processes employed by an organization to establish and implement an underlying structure made up of associated life cycle processes and personnel and continuously improve the structure and processes. They are typically employed outside the realm of specific projects and contracts; however, lessons from such projects and contracts contribute to the improvement of the organization.

- The management process group (MAN) consists of processes that contain practices that may be used by anyone who manages any type of project or process within a software life cycle. The processes of this group are:
  - MAN.1: Organizational alignment
  - MAN.2: Organizational management
  - MAN.3: Project management
  - MAN.4: Quality management
  - MAN.5: Risk management
  - MAN.6: Measurement
- The process improvement process group (PIM) consists of processes performed in order to define, deploy, assess and improve the

processes performed in the organizational unit. The processes of this group are:
  - PIM.1: Process establishment
  - PIM.2: Process assessment
  - PIM.3: Process improvement
- The resource and infrastructure process group (RIN) consists of processes performed in order to provide adequate human resources and necessary infrastructure as required by any other process performed by the organizational unit. The processes of this group are:
  - RIN.1: Human resource management
  - RIN.2: Training
  - RIN.3: Knowledge management
  - RIN.4: Infrastructure
- The reuse process group (REU) consists of processes performed in order to systematically exploit reuse opportunities in the organization's reuse programmes. The processes of this group are:
  - REU.1: Asset management
  - REU.2: Reuse program management
  - REU.3: Domain engineering

Supporting Life Cycle Processes

The supporting life cycle processes consist of processes that support another process as an integral part with a distinct purpose and contribute to the success and quality of the software project. A supporting process is employed and executed, as needed, by another process. The processes of this group are:

- SUP.1: Quality assurance
- SUP.2: Verification
- SUP.3: Validation
- SUP.4: Joint review
- SUP.5: Audit
- SUP.6: Product evaluation
- SUP.7: Documentation
- SUP.8: Configuration management
- SUP.9: Problem resolution management
- SUP.10: Change request management

Each SPICE process has a well-defined structure. After the process ID (e.g., SUP.10) and the process name (e.g., change request management), the purpose of the process is described. Then the

Table 6. CMMI and SPICE process structure

CMMI	SPICE
Acronym	Process ID
Process Name	Process Name
Purpose Statement	Process Purpose
Specific Goals	Process Outcomes
Specific Practices	Base Practices
Generic Goals	Process Attributes
Typical Work Products	Work Products

process outcomes are defined. These process outcomes—plus process attributes—have to be achieved to reach a capability level in SPICE. Afterwards base practices for each process are defined and work products of the process are listed.

Comparing the structure of a CMMI process area and a SPICE process, there are lots of similarities (see Table 6).

All in all, SPICE has 48 processes that mainly cover software development and have only small focus on service delivery. For this part, the ISO 20000 seems more applicable.

ISO 20000 Basics

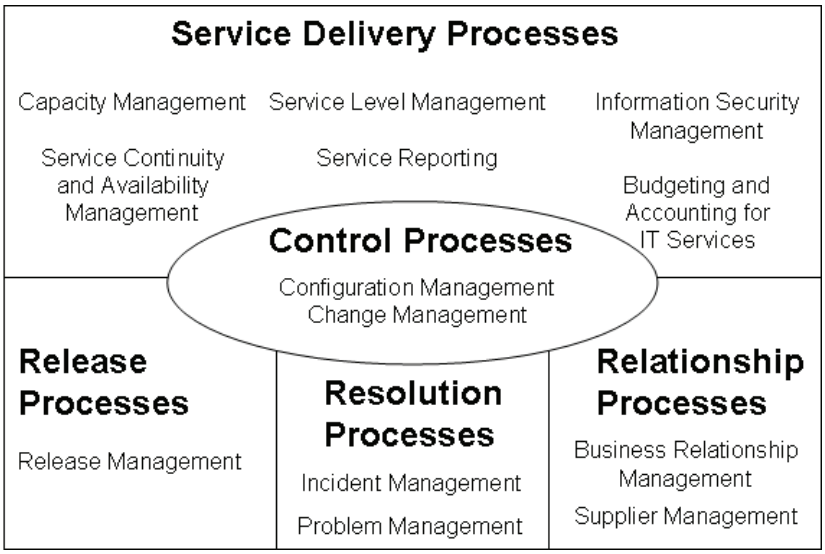
ISO 20000—Service Management covers the classical service delivery processes. Regarding the scope of ISO 20000, this standard represents "an industry consensus on quality standards for IT service management processes. These service management processes deliver the best possible service to meet a customer's business needs within agreed resource levels" (ISO/IEC, 2005b).

ISO 20000 defines 13 processes in 5 process groups.

Regarding each process group,

- The objective of the service delivery process group is to define, agree, record and manage levels of service, and consists of the processes
  - Capacity management
  - Service continuity and availability management
  - Service level management
  - Service reporting
  - Information security management
  - Budgeting and accounting for IT services
- The objective of the relationship process group is to describe the related aspects of

Figure 3. ISO 20000 processes



supplier management and business relationship management, and consists of the processes
    - Business relationship management
    - Supplier management
- The objectives of the resolution process group are to restore agreed service and minimize disruption to the business, and consists of the processes
    - Incident management
    - Problem management
- The objective of the control process group is to define and control the components of service and infrastructure, and consists of the processes
    - Configuration management
    - Change management
- The objective of the release process group is to deliver, distribute and track changes, and consists of the process
    - Release management

Those who know ITIL may have found lots of similarities in the process names and structure of ISO 20000. The ISO 20000 is well aligned with ITIL. Whilst ITIL is a collection of best practices, ISO 20000 defines specifications to support a service provider in delivering high quality services. The other way round, ITIL best practices help to achieve the quality of service management as defined by ISO 20000. It has to be recognized that ISO 20000 and ITIL are developed in strong connection, often impacted by the same persons.

Interfaces Between SPICE and ISO 20000

Trying to integrate SPICE and ISO 20000 it has to be taken into account, that both standards cover similar or identical elements in some processes. If an integrated product life cycle should be defined, these interfaces have to be harmonized.

First of all, the operational process group of SPICE has to be analyzed. This process group consists of two processes, operational use (OPE.1) and customer support (OPE.2).

The operational use process has the purpose to ensure the correct and efficient operation of the product for the duration of its intended usage and in its installed environment. Topics like operational risks, operational testing, criteria for operational use and the monitoring of the operational use are covered by this process.

The customer support process has the purpose of establishing and maintaining an acceptable level of service. Topics of this process are establishing of product support, performance monitoring and customer satisfaction.

Even though these processes address service aspects, they only deliver a high level overview. Nevertheless these topics address similar elements as the ISO 20000 processes service level management and business relationship management.

Other service delivery related processes can be found in the group of the supporting life cycle processes. The processes in focus are:

- SUP.8: Configuration management
- SUP.9: Problem resolution management
- SUP.10: Change request management

With configuration management the integrity of work products is established and maintained and these work products are made available to parties concerned. The problem resolution management process focuses on identification, analysis, management and controlling of discovered problems, while change request management ensures that change requests are managed, tracked and controlled.

Some other useful information for service delivery can be found in the process related to supplier management (acquisition / supply process groups).

Integration of SPICE and ISO 20000

To reach a fully integrated product life cycle, two requirements have to be satisfied:

- A set of processes has to be defined, which covers all stages of the life cycle—a so called process reference model (PRM)
- A model to evaluate the process capability has to be defined and must be applicable to all processes—a so called process assessment model (PAM)

Both requirements are satisfied by the CMMI constellations: the PAM is defined in the model foundation which is mandatory for all constellations and the PRM is given by the defined process areas.

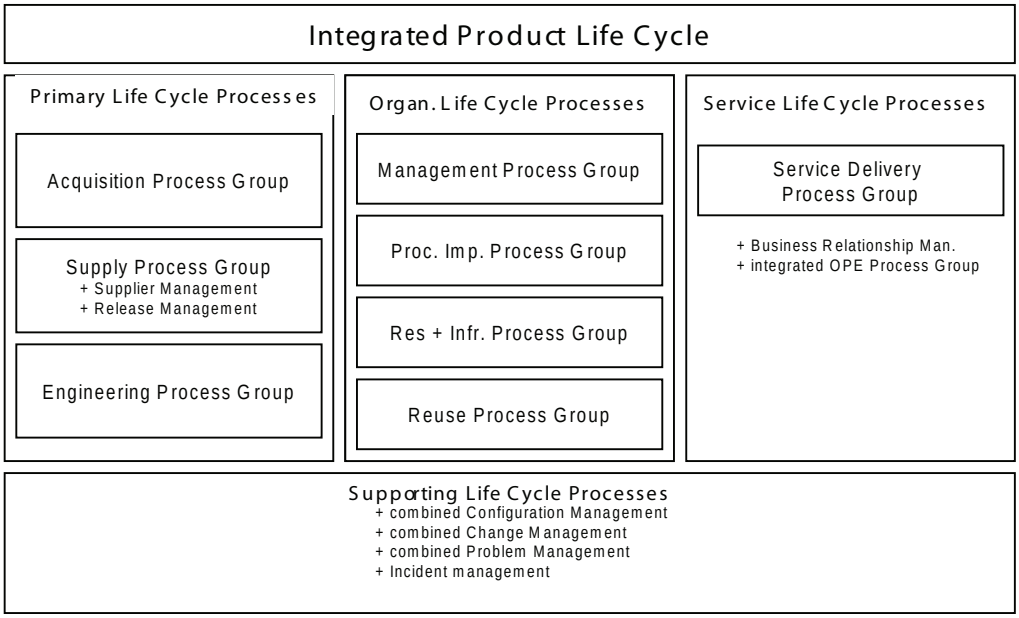
For the SPICE and ISO 20000 integration, this is not that easy. On the one hand SPICE and ISO 20000 address identical topics in different processes, on the other hand, only SPICE has an assessment model. A way to solve this problem was defined in early 2007 (Malzahn, 2007) and should be described in the following chapters.

A Combined PRM for SPICE and ISO 20000

As described before, double definitions in SPICE and ISO 20000 have to be eliminated and all processes need to have an identical structure. Therefore in a first step, the ISO 20000 processes have to be restructured into the SPICE process structure. In a second step, processes with a similar focus must be aligned.

For the first step, the ISO 20000 process groups and processes become process groups and processes of the combined model. For each ISO 20000 process the defined objective becomes the

Figure 4. Integrated Product Life Cycle based on SPICE and ISO 20000



process purpose. For the rest of the text it has to be decided, which text passage becomes an outcome, a base practice, or a work product.

In the second step, all processes with a similar focus have to be aligned. Concerning SPICE and ISO 20000 it is proposed to

- Integrate the SPICE OPE process group into the service level management, service reporting and business relationship management processes of ISO 20000,
- Combine configuration management
- Combine problem resolution management and problem management,
- Combine change request management and change management.

Other possibilities for further alignment are e.g., given by the combination of the ISO 20000 release management process and the SPICE product release process, and by integrating the relationship processes into the acquisition / supply process group.

After this integration and combination, the process groups may be restructured as follows:

- The ISO 20000 control processes become part of the support process group,
- The ISO 20000 release management process becomes part of the Supply process group,
- ISO 20000 problem management and incident management become part of the support process group,
- Business relationship management may become part of the service delivery process group.

In the very end, a combined PRM for SPICE and ISO 20000 may consist of the SPICE process categories and a new service life cycle category with the following amendments:

A Combined PAM for ISO 20000 and SPICE

The definition of a process assessment model for the integration of SPICE and ISO 20000 is easy to define—it is the approach defined for SPICE. Regarding ISO 20000, no measurement framework is defined. The requirements concerning measurement are given in part one of the ISO 20000 as follows:

"The service provider shall apply suitable methods for monitoring and, where applicable, measurement of the service management processes" (ISO/IEC, 2005a). Therefore no inconsistencies in the measurement and rating can occur. The SPICE definitions are applicable because each ISO 20000 process was restructured in the PRM and therefore all contextual and structural requirements are satisfied.

COMPARISON OF BOTH APPROACHES

Regarding both approaches—the CMMI constellations and the integration of SPICE and ISO 20000—there are pros and cons for each approach:

- The CMMI constellations are highly integrated by using the same model foundation, but CMMI-SVC will not see the light of day before 2008
- SPICE and ISO 20000 require additional effort to be integrated, but both are ISO standards and therefore widely accepted—especially if legal matters have to be taken into account.

Other impacts may be the region or industry of the organization. The CMMI is very strong in the United States; SPICE is heavily used in Europe.

Most defence industry companies are interested in CMMI, whilst major parts of the automotive industry prefer SPICE.

FUTURE TRENDS

This article covers two possible integration approaches. Nowadays more and more approaches see the light of day. Especially the integration of ITIL and CMMI or SPICE is widely discussed (Barafort, Di Renzo, Lejeune, Prime & Simon, 2005; Foegen & Graumann, 2007). Nevertheless ITIL is a best practice collection and not a measurement framework and therefore we still see some problems in the ITIL integration.

Hopefully the CMMI-SVC proves that it is a powerful tool for this intended integration and maybe at some point in time the ISO will find a way to publish an assessment model that covers the complete product life cycle for software.

Another promising approach will be the Enterprise SPICE initiative. The goal of this initiative is to "integrate and harmonize existing standards [...] to provide a single process reference model and process assessment model that addresses broad enterprise processes. Enterprise SPICE will provide an efficient and effective mechanism for assessing and improving processes deployed across an enterprise" (SPICE User Group, 2007).

Using Integrated Models

Defining an integration approach is only a short part on the way to software development and service delivery integration. Even though it builds the indispensable basis, integration only works, if it is accepted by organizations, teams, and people. To reach this, some simple rules of the thumb should be followed:

- Integrate the working level of development and service delivery at least in the review of an integrated model – preferably in the development. If the working level understands the need for integration and is part of the integration process, the integrated model is better accepted.
- Provide translation between the development, service delivery and integration approach. Only if software development and service delivery people reach a common understanding, an integrated approach can be established.
- Provide training on the approach. Training must not be focused on "we combine standard A with standard B" but on "we define an approach for the complete life cycle". There is no longer "their work" and "our work" but a common responsibility from first idea to retirement.

If and only if the need for integration is understood and accepted by people on working level, integration can be established – otherwise there will still be two worlds with all their differences and borders.

CONCLUSION

The decision, which approach may deliver the best benefit, must be taken by the organization itself. But one thing is inevitable: if the capability of the processes of an organization is not evaluated against accepted standards and for the complete life cycle, the organization keeps the back door open for chaotic elements in their process suite and therefore has an open door for abusing strategies, processes, and procedures.

REFERENCES

Barafort, B., Di Renzo, B., Lejeune, V., Prime, S., & Simon, J. M. (2005). ITIL based service management measurement and ISO/IEC 15504 process assessment: A win-win opportunity.

In *Proceedings of the SPICE 2005 Conference*, Klagenfurt, Austria.

Department of Defence (1996). *Guide to integrated product and process development (Version 1.0).* Washington, DC: Office of the Under Secretary of Defense (Acquisition and Technology).

Foegen, M., & Graumann, S. (2007, June). CITIL: ITIL integrated into CMMI. In *Proceedings of the 12th annual European SEPG Conference*, Amsterdam, Netherlands.

Hollenbach, R., & Buteau, B. (2006, November). CMMI for services, introducing the CMMI for service constellation. In *Proceedings of the CMMI Technology Conference*, Denver CO.

Humphrey, W. (1989). *Managing the software process.* Boston, MA: Addison-Wesley.

ISO/IEC (2005a). *ISO/IEC 20000-1Information technology—Service management—Part 1: specification*, ISO/IEC.

ISO/IEC (2005b). *ISO/IEC 20000-2- Information technology – Service management – Part 2: Code of practice*, ISO/IEC.

ISO/IEC (2006). *ISO/IEC 15504-5 -Information technology—Process assessment—Part 5: An exemplar process assessment Model*, ISO/IEC.

Malzahn, D. (2007, May). *A service extension for SPICE?* In Proceedings of the SPICE 2007 Conference. Seoul, South Korea.

Software Engineering Institute (2007). *CMMI for acquisition, Version 1.2.* Pittsburgh, PA.

Software Engineering Institute (2006a). *CMMI for development, Version 1.2*. Pittsburgh, PA.

Software Engineering Institute (2006b): *CMMI for services* (Initial Draft). Pittsburgh, PA.

SPICE User Group (2007). *Enterprise SPICE introduction*. Retrieved May 15, 2008, from http://www.enterprisespice.com/web/Introduction.html

ENDNOTES

1 CMMI is a registered trademark of the Software Engineering Institute, Carnegie Melon University.

2 This explains why the continuous representation only requires to satisfy generic goals. GG1 specifies, that the specific goals have to be achieved.

This work was previously published in Information Technology Governance and Service Management: Frameworks and Adaptations, edited by A. Cater-Steel, pp. 423-442, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 2.28
BROOD:
Business Rules–Driven Object Oriented Design

Pericles Loucopoulos
Loughborough University, UK

Wan M. N. Wan Kadir
Universiti Teknologi Malaysia, Malaysia

ABSTRACT

A critical success factor for information systems is their ability to evolve as their environment changes. There is compelling evidence that the management of change in business policy can have a profound effect on an information system's ability to evolve effectively and efficiently. For this to be successful, there is a need to represent business rules from the early requirements stage, expressed in user-understandable terms, to down-stream system design components and maintain these throughout the lifecycle of the system. Any user-oriented changes could then be traced and if necessary propagated from requirements to design specifications and evaluated by both end-users and developers about their impact on the system. The BROOD approach, discussed in this article, aims to provide seamless traceability between requirements and system designs through the modelling of business rules and the successive transformations, using UML as the modelling framework.

INTRODUCTION

The ubiquitous nature of information systems and the increasing dependency of organizations, government and society on such systems highlight the importance of ensuring robustness in their operation. At the same time rapid changes in the environment of information systems places an increasing emphasis on the ability of these systems to evolve according to emerging requirements. A large proportion of a total systems' lifecycle cost is devoted to introducing new requirements, and removing or changing existing system functional-

ity (Grubb & Takang, 2003). Software evolution therefore is considered as a key challenge in the development and maintenance of information systems (Erlikh, 2000).

In recent years there has been an increasing interest of the IS community in business rules, which has resulted in dedicated rule-centric modeling frameworks and methodologies (Ross & Lam, 1999; Zaniolo et al., 1997), international initiatives for the investigation of business rules' role in the context of knowledge management (Hay & Healy, 1997), conferences, workshops and tutorials (Mens, Wuyts, Bontridder, & Grijseels, 1998), and rule-centric rule management tools and application development support environments (e.g., Blaze Advisor Builder, BRS RuleTrack, Business Rule Studio, Haley Technologies, ILOG Rules, Platinum Aion, Usoft Developer and Visual Rule Studio). Whilst these efforts make significant contributions in their own right, a key challenge remains unanswered namely the *linking of business rules specifications to software designs*.

The aim of the BROOD (business rules-driven object oriented design) approach is to address the issue of software evolution from both *requirements* and *design* perspectives. This confluence should provide a seamless and traceable facility that arguably should bring about a more effective way of dealing with software evolution, by aligning changes of the information system to changes in its environment. BROOD adopts as its methodological paradigm that of object orientation with UML as its underlying graphical language. It augments UML by explicitly considering business rules as an integral part of an object-oriented development effort. To this end BROOD aims:

i. To explicitly model business rules in a manner understandable to end-user stakeholders.
ii. To map these to formal descriptions amenable to automation and analysis.
iii. To provide guidelines on the deployment of business rules in the development process.
iv. To provide guidelines on the evolution of requirements and related design specifications.

The article is organized as follows. Section 2 discusses the background to business rules modeling. Section 3 introduces the motivation for BROOD. Section 4 introduces the BROOD metamodel as the foundation for modeling business rules. Section 5 discusses the manner in which business rules are linked to design components via the concept of 'rule phrase.' The BROOD process is detailed in section 6. The BROOD approach is supported by an automated tool and this is briefly discussed in Section 7. The article concludes with an overview of BROOD, observations on its use on a large application and comparisons with traditional approaches.

The language details for business rules definition are given in appendix A. The BROOD approach is demonstrated through an industrial application which is described in appendix B. This application had originally been developed using a traditional approach. Therefore, it proved useful not only as a means of providing a practical grounding on BROOD but also on comparing and contrasting the use of BROOD with a traditional development effort.

BUSINESS RULES MODELLING

The motivation of BROOD is to provide a development environment whereby the business analysis and system design domains are supported by business rules modeling with the specific aim to facilitating more effective software evolution.

The term "business rule" has been used by different authors in different ways. For example, in (Rosca, Greenspan, Feblowitz, & Wild, 1997), business rules are:

statements of goals, policies, or constraints on an enterprise's way of doing business.

In (Herbst, 1996a), they are defined as:

statements about how the business is done, i.e. about guidelines and restrictions with respect to states and processes in an organization.

Krammer considers them as "programmatic implementations of the policies and practices of a business organization" (Krammer, 1997) whilst Halle states that:

depending on whom you ask, business rules may encompass some or all relationship verbs, mathematical calculations, inference rules, step-by-step instructions, database constraints, business goals and policies, and business definitions. (Halle, 1994).

In general, business rules in the information systems field may be viewed in terms of two perspectives: (a) business rules as applied to conceptual modeling and (b) business rules as applied to evolvable software systems development.

Business Rules in Conceptual Modeling

1. Business rules as part of requirements gathering and systems analysis have not been ignored by structured analysis, information engineering or object-oriented analysis approaches (Moriarty, 1993) which, to varying degrees, subsume or represent business rules as part of notation schemes used to specify application requirements (Gottesdiener, 1997) Ross (1997) comments that traditional IS methodologies have addressed rules poorly, and only relatively late in the system development lifecycle. (Hay & Healy, 1997) mention that rules dealing with information structure may be represented by any of several flavors of entity—relationship or object class diagrams, and responses to events may be shown via essential data flow diagrams (McMenamin & Palmer, 1984) or as entity life history diagrams (Robinson & Berrisford, 1994).

From a conceptual perspective there are approaches that consider business rules as an integral part of the modeling and analysis of systems' requirements. An early effort in this direction was the *RUBRIC* project (Loucopoulos & Layzell, 1986; van Assche, Layzell, Loucopoulos, & Speltinex, 1988) parts of which were integrated into the information engineering (Martin, 1989) method.

In *BROCOM* (Herbst, 1996b, 1997), the rule language is a type of structured English, and therefore it is highly expressive. Moreover, rules are organized according to a rich meta-model, and can be retrieved based on a number of different criteria. As far as methodological guidance is concerned, Herbst proposes the development of various models which are helpful during the analysis phase, but the process of creating and using them is not clearly defined. The transition from analysis to design and implementation has not been addressed by this approach.

The *DSS* approach (Rosca, Greenspan, & Wild, 2002; Rosca et al., 1995) focuses on the analysis phase of IS development by supporting the rationale behind the establishment of rules. DSS adopts the ECA (event-condition-action) paradigm for structuring rule expressions and also links these expressions to the entities of an underlying enterprise model. The absence of a formal rule language confines the use of DSS on modeling tasks.

The Business Rules Group (*BRG*), formerly known as the GUIDE Business Rule Project (Hay & Healy, 1997), investigated an appropriate formalization for the analysis and expression of business rules (Hay & Healy, 2000). This approach identifies terms and facts in natural language rule statements, and consequently, it offers a high level

of expressiveness. The meta-model it provides for describing the relations between these terms and facts is very detailed. Therefore, rule models are (a) highly manageable and (b) formal and fully consistent with the information models of a specific organization.

The *IDEA* method (Zaniolo et al., 1997) focuses on the maintenance of formality and consistency with underlying business models. The method offers guidance for every activity being involved in the development of a rule-centric information system. The IDEA method is directed towards the use of specific active and deductive databases, and of the corresponding rule languages. As a result of this, (a) IDEA rules are rather difficult to be expressed or even understood by business people;

Table 1. Comparative evaluation of business rule in conceptual modeling

Criteria \ BR Approach	BRG	BRO-COM	BRS
Concepts			
Business Rule Definition	IS	IS	Business
Business Rule Taxonomy			
- Structural Rules	High (10)	Low (0)	Medium (1)
- Behavioural Rules	Medium (8)	High (>30)	Medium (8)
- Derivation	Medium (2)	Low (0)	Medium (2)
Bus. Rule Management Elements	Medium (5)	Medium (9)	High (>30)
Modelling Language			
Understandability	Medium	Medium	High
Expressiveness (business rules)	Medium	High	High
Unambiguity	Medium	High	Medium
Formality	Medium	Medium	High
Evolvability	Medium	Medium	High
Process			
Lifecycle coverage	A	A	A + D
Process description	N/A	High	High
Coherence	N/A	High	High
Support for evolution	No	Yes	Yes
Pragmatics			
Communicability	Medium	High	High
Usability	Medium	High	High
Resources availability	Low	Medium	High
Openness	High	Medium	High

Lifecycle coverage: A-Analysis, D-Design, I-Implementation, M-Maintenance

and (b) the choice of technologies to be employed for the development of an information system is rather limited.

The *BRS* approach (Ross, 1997) is formal, in accordance with the underlying data models of an organization, offers sufficient methodological guidance, and allows management of rule expressions based on a very detailed meta-model. It is also one of the few methods that adopts a graphical notation for expressing rules. Regarding the development process, BRS introduces a business rule methodology called BRS Proteus™ methodology that defines a number of steps for both business and system modeling (Ross, & Lam, 2003). BRS also provides the BRS RuleTrack™, an automated tool for recording and organizing business rules.

The object constraint language (*OCL*) of UML (Eriksson & Penker, 2000) is tightly bound with the widely accepted UML but lacks methodological guidance for the collection of rules. Rule structures are implied by the allocation of rules to classes, attributes, associations and operations.

A comparative evaluation of the treatment of business rules for conceptual modeling by three widely used approaches is shown in Table 1.

Business Rules in Evolvable Software Evolution

The majority of approaches in this category aim to improve the understanding and evolution of a software system by logically and physically separating business rule components from other software components.

The adaptive object model (*AOM*), which is also known as the dynamic object model (Riehle, Tilman, & Johnson, 2000), is *"a system that represents classes, attributes, and relationships as metadata"* (Yoder, Balaguer, & Johnson, 2001). Unlike traditional object-oriented design, AOM is based on objects rather than classes. It provides descriptions (metadata) of objects that exist in the system. In other words, AOM provides a meta-architecture that allows users to manipulate the concrete architectural components of the model such as business objects and business rules. These components are stored as an object model in a database instead of in code. The code is only used to interpret the stored objects. Thus, a user only needs to change the metadata instead of changing the code to reflect domain changes.

The *coordination contract* method aims to separate coordination from computation aspects (or core components) of a software system (Andrade, Fiadeiro, Gouveia, & Koutsoukos, 2002). It is motivated by the fact that there should be two different kinds of entities in a rapidly changing business environment—core business entities which are relatively stable and volatile business products which keep changing for the business to remain competitive (Andrade & Fiadeiro, 2000). Volatile business products are implemented as contracts. A contract aims to externalize the interactions between objects (core entities) by explicitly define them in the conceptual model. It extends the concept of association class by adding a coordination role similar to other components in architecture-based software evolution such as architectural connectors (Oreizy, Medvidovic, & Taylor, 1998), glue (Schneider, 1999), actor (Astley & Agha, 1998) or change absorbers (Evans & Dickman, 1999).

Business Rule Beans (*BRBeans*), formerly known as accessible business rules (Rouvellou, Degenaro, Rasmus et al., 1999; Rouvellou, Degenaro, Rasmus et al., 2000), is a framework that provides guidelines and infrastructures for the externalization of business rules in a distributed business application (IBM, 2003). Business rules are externally developed, implemented and managed to minimize the impact of their changes on other components such as core business, application, and user interface objects. They are implemented as server objects, which are fired by embedded trigger points in application objects. A rule management facility is provided to help us-

Table 2. Comparative evaluation of business rules in evolvable software systems

Criteria \ BR Approach	Adaptive Object Model (AOM)	Coordination Contract	Business Rule Beans (BRBeans)
Concepts			
Business Rule Definition	Implicit	Implicit	Explicit
Business Rule Taxonomy	primitive, composite, workflow	ECA	derivation, constraint, invariant, script, classifier
Business Rule Management Elements	Nil	Nil	Yes
Modelling Language			
Understandability	High	Medium	Medium
Expressiveness (business rules)	Low	Medium	Medium
Formality	Low	High	Medium
Evolvability	High	High	High
Process			
Lifecycle coverage	(Evolutionary)	D + I + T + M	A + D + I + T + M
Process description	Low	Medium	High
Coherence	Medium	Medium	Medium
Support for evolution	Low	Medium	High
Pragmatics			
Communicability	High	Medium	Medium
Usability	Low	Medium	Medium
Resources availability	Medium	Medium	High
Openness	Medium	Medium	Low

ers to understand the existing rules and to locate the rules when changes are required. BRBeans is implemented as a part of WebSphere Application Server by IBM *"to support business applications that externalize their business rules"* (Kovari, Diaz, Fernandes et al., 2003).

A comparative evaluation of the treatment of business rules evolvable software systems development by the three approaches is shown in Table 2.

MOTIVATION FOR THE BROOD APPROACH

According to Lehman's laws (Lehman & Belady, 1985), a software system that is used in a real-world environment inevitably must change or become progressively less useful in that environment. Lehman's laws also state that the software structure tends to become more complex due to the implemented changes and its size must continue to grow to accommodate new user requirements. Therefore, there is a need to introduce a method

that facilitates the management of the increasingly complex and larger size software system due to its evolution.

The position put forward in this article is that developers need to identify the sources of changes for software evolution in the system's environment and that some of the most volatile of these components tend to be business rules. In section 0 many contemporary approaches were reviewed all of which aim to externalize business rules from software components.

At the conceptual modeling level, there are approaches that separate syntax and semantics for modeling business rules. This effort localizes the changes to business rule components, and also increases the understanding and maintainability of business rules specification. This category of approaches provides a great deal of help in dealing with the concepts related to business rules, but they provide relatively little description on the design and implementation aspect of business rules.

At the implementation level, approaches create separate software components that implement business rules. As a result, the business rule changes will only localize to such components, and reduce the impact of changes to the overall software structure. This group of approaches provides very good facilities for developing evolvable software components but is less helpful in representing business rules at the conceptual business level.

The BROOD approach addresses both business modeling and the linking of business model components to software architecture components. By focusing on the conceptual level, BROOD attempts to externalizing changes from software components. This user-oriented view enhances understandability and maintainability since it encourages the direct involvement of business stakeholders in the maintenance of their business rules.

By introducing a linking component between the conceptual model of business rules and software design, BROOD attempts to increase business rule traceability. Traceability is highly desirable since one can keep 'forward' and 'backward' tracks of changes between business and software.

BROOD considers both *product* and *process* perspectives of the development and evolution of a software system. The *product* is defined using the BROOD metamodel, which specifies the structure for business rule specification, software design, and their linking elements. The *process* refers to a set of systematic and well-defined steps that should be followed during software development and evolution. The BROOD process emphasizes several important activities in a software lifecycle that contribute to a more resilient software system.

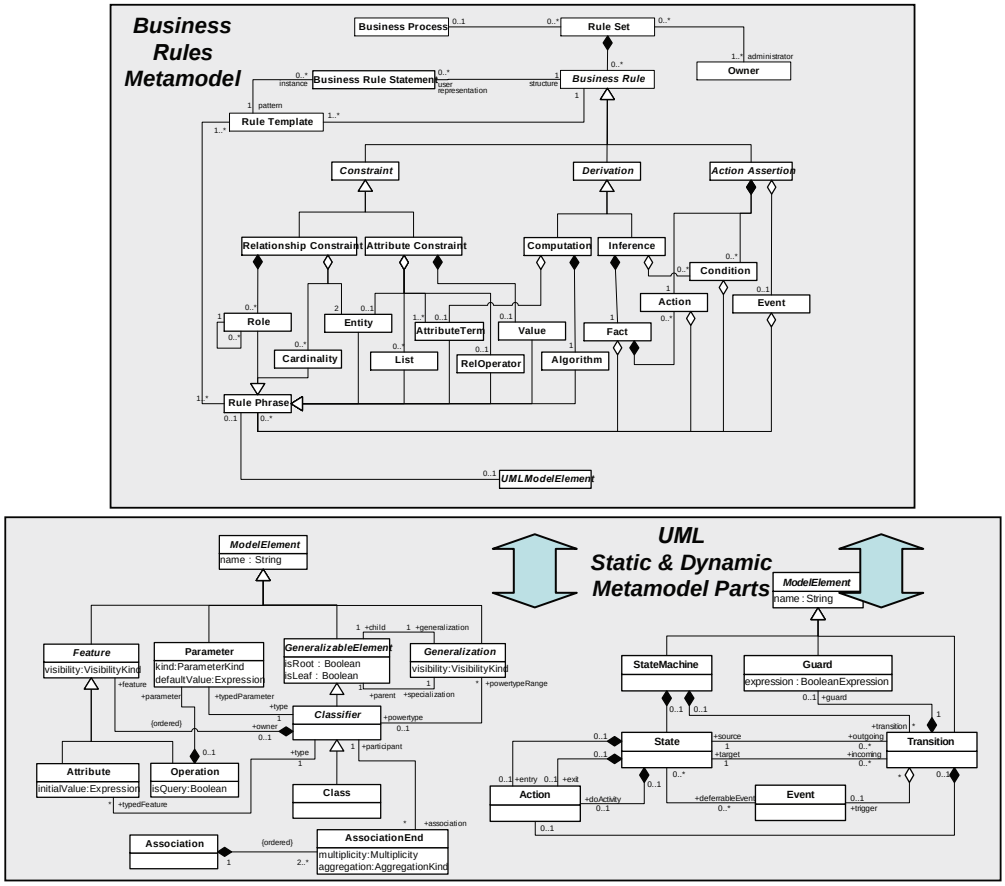
THE BROOD METAMODEL

The initial concept of the metamodel was introduced in (Wan Kadir & Loucopoulos, 2003; Wan Kadir & Loucopoulos, 2004). The metamodel is complemented by a language definition based on the context-free grammar EBNF, which is included in appendix A. The language definition defines the allowable sentence patterns for business rule statements and describes the linking elements between business rules and the related software design elements.

At the outset, three main desirable characteristics were set for developing an appropriate business rule metamodel, which would be consistent with the aims of BROOD:

- It should have an exhaustive and mutually exclusive typology to capture different types of business rules.
- It should have the structured forms of expressions for linking the business rules to software design.
- It should include rule management elements to improve business rule traceability in a business domain.

Figure 1. The BROOD business rule metamodel



These three characteristics form the basis for the development of the business rule metamodel, which is shown in Figure 1. This figure shows the business rules metamodel together with parts of the UML metamodel that deal with static (classes) and dynamic (actions and events) aspects. The key requirement of BROOD for tracing changes from business to software through the use of business rules is achieved by integrating these three metamodels.

Business Rules Typology

The metamodel classifies business rules into three main types, which are constraint, action assertion, and derivation.

Constraints

Constraint rules specify the static characteristics of business entities, their attributes, and their relationships. They can be further divided into

attribute and relationship constraints. The former specifies the uniqueness, optionality (null), and value check of an entity attribute. The latter asserts the relationship types as well as the cardinality and roles of each entity participating in a particular relationship.

Examples of attribute constraints from the MediNet application expressed according to the BROOD syntax (see `attribute constraint` definition in appendix A) are the following:

Patient must have a unique patient registration number.
Patient may have a passport number.
Bill must have a unique bill number.
The amount of Bill must be less than the maximum bill amount set by the paymaster.
An employee level of a Panel Patient must be in {employer, executive, production operator}.

Examples of relationship constraints for MediNET (see `relationship constraint` definition in appendix A) are:

Clinic item is a/an item type of bill item.
Bill must have zero or more bill item.
HCP Service Invoice is a/an Invoice.

Actions

Action assertion concerns a behavioral aspect of the business. Action assertion specifies the *action* that should be activated on the occurrence of a certain *event* and possibly on the satisfaction of certain *conditions*. An event can be either a simple or a complex event where the latter is constructed by one or more simple events using the logical connectives AND/OR. A condition may be a simple or complex condition. A simple condition is a Boolean expression which compares a value of an entity attribute with any literal value or the value of another entity attribute using a relational operator. It can also be an inspection of the existence of a value of an entity attribute in a list of values.

An action is performed by a system in response to the occurrence of an event and the satisfaction of the relevant condition. The execution of action may change the state of the system. An action may be a simple action or a sequence of simple actions. Simple actions can be further categorized into three different types, trigger actions, object manipulation actions, and user actions. Trigger action invokes an operation, a process, a procedure, or another rule under certain circumstances. Object manipulation action sets the value of the attribute or create/delete an instance of an entity. User action is a manual task that is done by system users. During implementation, user action is often implemented as a message displayed to the user.

Examples of action assertion for MediNET (see `action assertion` definition in appendix A) are:

When new invoice created then calculate invoice end date.
When patient consultation completed then removed the patient from consultation queue and create bill for the patient.
When invoice entry updated if stock of drug smaller than re-order threshold then reorder the drug.

Derivation

A derivation rule derives a new fact based on existing facts. It can be of one of two types, *computation*, which uses a mathematical calculation or algorithm to derive a new arithmetic value, or *inference*, which uses logical deduction or induction to derive a new fact. Typically, an inference rule may be used to represent permission such as user policy for data security. An example of a computation derivation rule such as "The amount HCP MediNET usage invoice is computed as the amount of transaction fees, which are calculated as the transaction

fee multiply by the total number of transactions, plus the monthly fee" would be expressed as:

```
let a = transaction_fee;
let b = number_of_treated_patient;
transaction_fees = a * b;
invoice_amount = transaction_fees + monthly_fee;
```

Examples of inference rules are given below:

Table 3. Business rule templates

Types	Templates
Attribute Constraint	<entity> must have | may have [a unique] <attributeTerm>. <attributeTerm1> must be | may be <relationalOperator> <value> | <attributeTerm2>. <attributeTerm> must be in <list>.
Relationship Constraint	[<cardinality>] <entity1> is a/an <role> of [<cardinality>]<entity2>. [<cardinality>] <entity1> is associated with [<cardinality>]<entity2>. <entity1> must have | may have [<cardinality>] <entity2>. <entity1> is a/an <entity2>.
Action Assertion	When <event> [if <condition>] then <action>. *The templates of <event> :* <attributeTerm> is updated <entity> is deleted | is created <operation>|<rule> is triggered the current date/time is <dateTime> <number> <timeUnit> time interval from <dateTime> is reached <number> <timeUnit> after <dateTime> <userEvent> *The templates of <condition> :* <attributeTerm1> <relationalOperator> <value | attributeTerm2> <attributeTerm> [not] in <list> *The templates of <action> :* trigger <process> | <operation> | <rule> set <attributeTerm> to <value> create | delete <entity> <userAction>
Computation	<attributeTerm> is computed as <algorithm>
Derivation	if <condition> then <fact>. *The templates of <fact> :* <entity> | <attributeTerm> is [not] a <value> <entity> may [not] <action>

If the paymaster's last quarter transaction is more than RM12,000.00 and the paymaster has no past due invoices then the paymaster is a preferred customer.

If the user type is equal to HR Officer and the user company is equal to patient paymaster then the user may view the patient's medical certificate.

The Rule Template

Rule templates are the formal sentence patterns by which business rules can be expressed. They are provided as a guideline to capture and specify business rules as well as a way to structure the business rule statements. Each rule template consists of one or more well-defined rule phrases, which are discussed in section 0.

By using the available templates, an inexperienced user may easily produce a consistent business rule statement. Rule templates help users to avoid tedious and repeated editing when creating many similar rules; and ensure uniformity by restricting the type of rules that can be written by business users. The use of templates also allows the precise linking of business rules to software design elements. The templates can be directly derived from the rules definition in Appendix A. Business rules templates are shown in Table 3.

The Rule Management Elements

Management elements are also included in the BROOD metamodel for facilitating the organization and management of business rules. These elements include the *rule set*, *business process*, and *owner*.

Rule set is used to group business rules into a set of closely interrelated rules. Each business rule model must have a single rule set, which is considered as the root rule set. This rule set must have at least one rule statement or another rule set.

One of the popular ways to identify a rule set is through its related business process. For example, the rules 'The bill amount is calculated as the sum of amounts of all bill items' and 'If a patient is a panel patient and his paymaster pays the bill in full, the balance is set to 0 and the bill status is set to paid' can be grouped in a rule set which is related to 'bill preparation' process. By properly organizing rules, the complexity of managing a large set of rules can be reduced.

Each business rule model must have an owner. An owner may also be defined for a rule set. The owner of a parent rule set is assumed to be the owner of its child rule set if the child does not define its owner. It is important to define the owner information in a business rule model to determine the access rights and responsibility to a business rules repository, especially for software systems with multiple user groups that possess different business rules. An owner may be an organizational unit, an individual user, a user group or role that is responsible for the management of the respective business rules. During business rule implementation, each rule set, business process, and owner is given a unique identifier.

THE RULE PHRASE

A rule phrase in BROOD links a user-oriented business rule definition to a software design component. There are alternative ways in which this may be achieved. For example, using a rule object or rule engine, or making use of OCL. The use of rule object or rule engine increases the semantic distance between analysis and design and imposes implementation considerations. The use of constraints expressed using OCL may provide a link between business rule specifications and software design but OCL is still hard to understand by business users although OMG claims that no mathematical background is required in using OCL.

Rule phrases are considered as the building blocks for rule statements. They can be maintained independently during implementation, in other words, they are not deleted when a business rule is deleted. However, the modification and deleting of a rule phrase is not recommended since a careful effort is needed in reviewing its aggregated business rules. In addition to playing a role as the building blocks for business rule statements, rule phrases are also important in linking business rules to software design elements.

The mappings between rule phrase types and UML model elements are summarized in Table 4. Most of the rule phrases are directly linked to class diagram model elements. Entity and attribute term are directly connected to the respective class and attribute in the class diagram. Cardinality and role are correspondingly linked to multiplicity and role of an association end of a relationship. Algorithm is linked to operation specification.

Rule phrases for event, condition, and action, which are the building blocks for action assertion rules, are naturally linked to statechart diagram. Event, condition, and action are respectively linked to event, guard, and action of a state transition in a statechart diagram. Consequently, event and action may be linked to a class operation, and guard may be linked to an operation specification, in a class diagram. List and relational operator contain enumerated values whilst value contains a literal value. However, value and list can be linked to an operation that return a single and multiple values respectively.

THE BROOD PROCESS

The BROOD process is described using the process model based on the syntax and semantics of the OMG software process engineering

Table 4. Association between rule phrases and design elements

Rule Phrase Type	Software Design Elements
Entity	Class
Attribute Term	Attribute
Operation Term	Operation
Attribute Constraints	Attribute.isUnique, Attribute. notNull
Cardinality	AssociationEnd.multiplicity
Role	AssociationEnd.role
Event	Transition.event □ Class.operation
Condition	Transition.guard, Operation.specification
Action	Transition.action □ Class.operation
Algorithm	Operation.specification
Value	- (literal value), Operation.
List	- (enumeration), Operation
Relational Operator	- (enumeration)

Figure 2. An excerpt from OMG software process engineering metamodel (OMG, 2002)

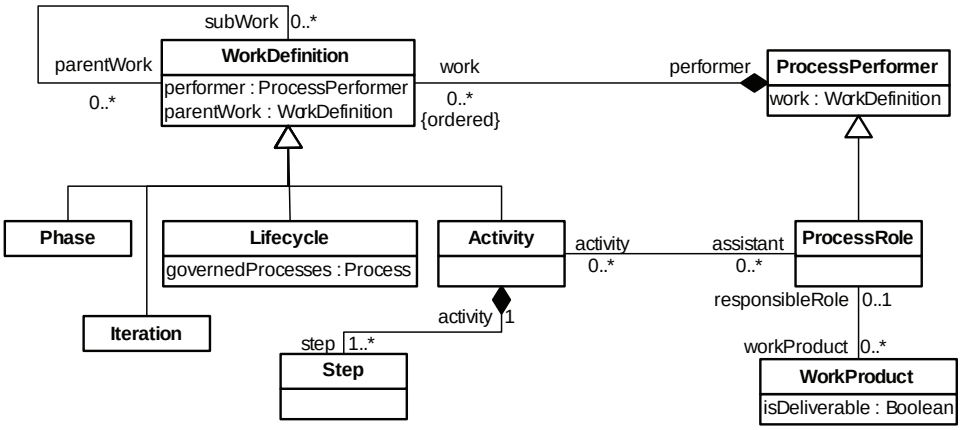
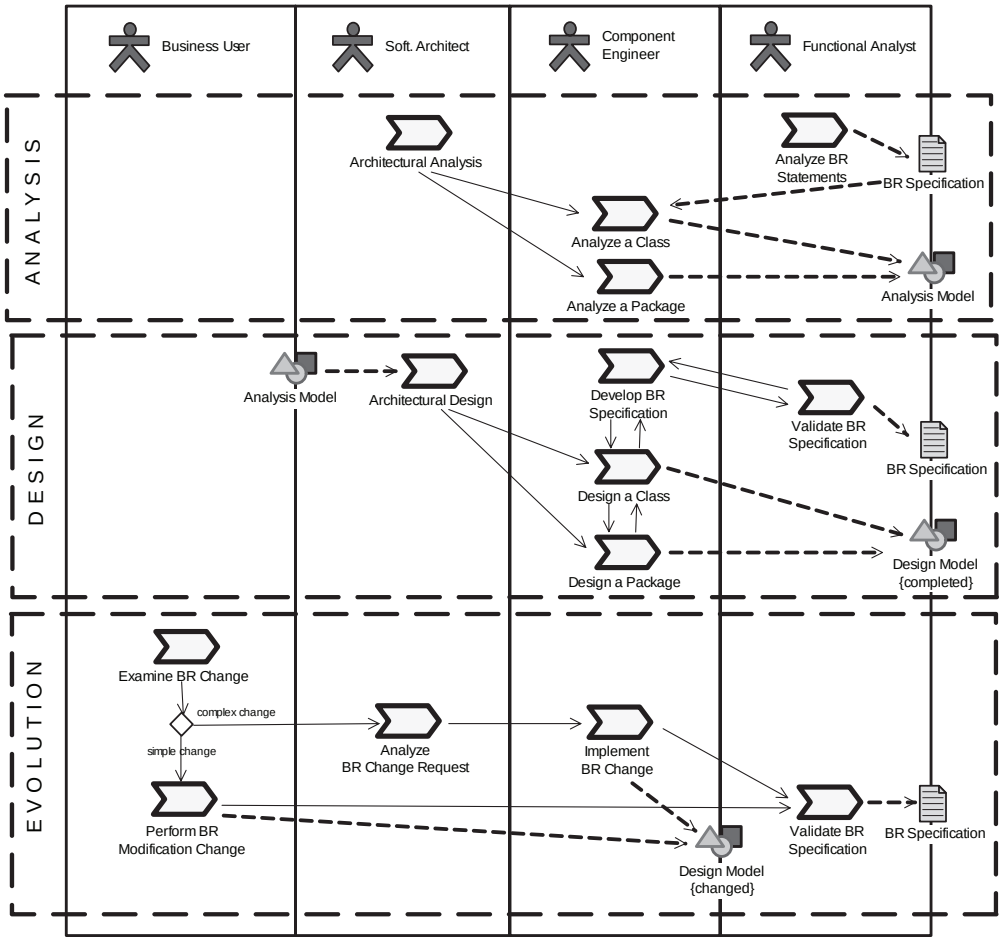


Figure 3. The flow of activities in the BROOD process



metamodel (SPEM). SPEM was developed by the Object Management Group to provide a metamodel and notations for specifying software processes and their components (OMG, 2002). SPEM extends the unified modeling language (UML) (OMG, 2001) metamodel with process specific stereotypes. A part of SPEM that shows most of the important components of a process structure is shown in Figure 2.

In SPEM, a *work product* is an artifact produced, consumed, or modified by a process. It may be a piece of information, a document, model, or source code. It is either used as an input by workers to perform an activity, or a result or an output of such activities. A work product is called a deliverable if it is needed to be formally delivered by a process. The examples of work products in BROOD are class diagram, statechart diagram, and business rule specification. Each work product is associated with a process role that is formally responsible for its production.

A *process role* defines the responsibilities of an individual, or a group of individuals working together as a team. Each process role performs or assists with specific activities.

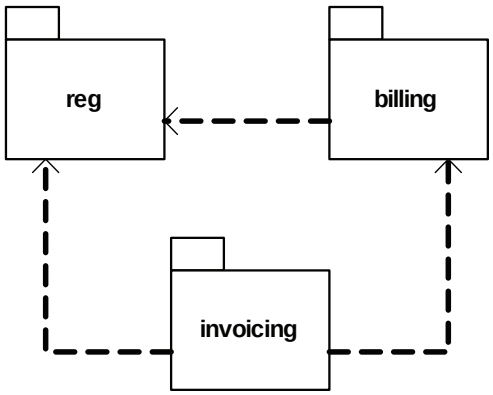
The core activities of the BROOD process are situated in the analysis, design, and evolution phases. Analysis phase produces analysis model that contains two main work products: the initial business rule specification and preliminary software design models. Both work products are refined and linked during the design phase to produce a more traceable and consequently evolvable software system. The flow of activities in each BROOD phase is shown in Figure 3.

The Analysis Phase

As shown in Figure 4, the analysis phase starts with an architectural analysis activity that considers the work products from requirements phase such as use-case model, business model, initial architecture descriptions, and supplementary requirements. A software architect performs architectural analysis by identifying the analysis packages based on the functional requirements and knowledge of the application domain. Each package realizes a set of closely related use cases and business processes to minimize the coupling between packages, which in turn localizes business changes. This activity identifies analysis classes and outlines their name, responsibilities, attributes, and relationships. In order to extract more information about the behavior of the classes, collaboration or interaction diagrams can be developed based on the process flows (scenario) in the use case models. The main work products produced by this activity are analysis class diagrams and packages in their outline version.

Considering the MediNet application, architectural analysis resulted in three packages *business processes* i.e. *registration*, *billing*, and *invoicing*. The registration package groups all classes related to patient registration such as `Patient`, `Paymaster`, `HCProvider`, `Clinic`, `User`, and `RegLocation`. Billing package contains classes related to billing and drugs inventory such as `Bill`, `BillPayment`, `Bill _ Item`, `TransType`, `TransItem`, and `ExpenseItem`. Invoicing package includes classes related to invoicing and invoice payment

Figure 4. Packages for the MediNet application



for example `Invoice`, `InvoiceItem`, `Payment`, and `PaymentAllocation`.

The outline of analysis class diagrams and packages are further refined by class analysis and package analysis activities, respectively. A component engineer identifies more detailed information about responsibilities and attributes of each class. Different types of relationships between classes such as association, aggregation, and inheritance are also identified. The possible states and their transitions can be identified to understand the behavior of objects from certain classes. These steps are repeated until a complete analysis class diagram, statechart diagram and package are achieved.

The activity of business rule modeling considers the informal statements captured during initial requirements and identifies the types for each business rule statement based on the BROOD typology. Business rule statements are transformed into more structured business rule specifications according to the templates' definition.

Table 5. Business rule statements for the MediNET application

Business Process	Business Rule Example	Rule Type
Registration	A patient must have a unique registration number.	Att. Constraint
	A patient may have more than one paymaster.	Rel. Constraint
	If a patient has an outstanding balance, then the patient should be banned from consultation registration	Action Assertion
	When consultation registration is successfully completed, then put the patient into the consultation queue.	Action Assertion
	If a patient's condition is critical then the patient is an emergency patient.	Inference
Billing	The amount of a panel patient's bill must not exceed the maximum bill amount set by the paymaster.	Att. Constraint
	Each bill item is associated with an item from the clinic transaction items	Rel. Constraint
	When consultation is completed then create bill.	Action Assertion
	If the bill is a panel patient's bill then create panel transaction item.	Action Assertion
	The amount of a bill is computed as the sum of all amounts of bill items.	Computation
	The amount of bill item is computed as the unit amount multiply by the quantity.	Computation
	A bill can be modified only if the user role is Chief Clinic Assistant.	Inference
Invoicing	One invoice must have zero or more payments.	Rel. Constraint
	When a payment is not received within 30 days from the invoice date, then the first reminder will be sent.	Action Assertion
	The amount of HCP MediNET usage invoice is computed as the sum of monthly subscription fee plus transaction fees.	Computation
	A paymaster (panel company) is under probation if the paymaster has an invoice with category 1 past due and the current balance is more than RM 5,000.00.	Inference

Table 5 shows a set of structured rules for the MediNet application. This template provides the means of managing rules as they get discovered and analyzed and acts as a 'repository' of rules for their entire lifecycle.

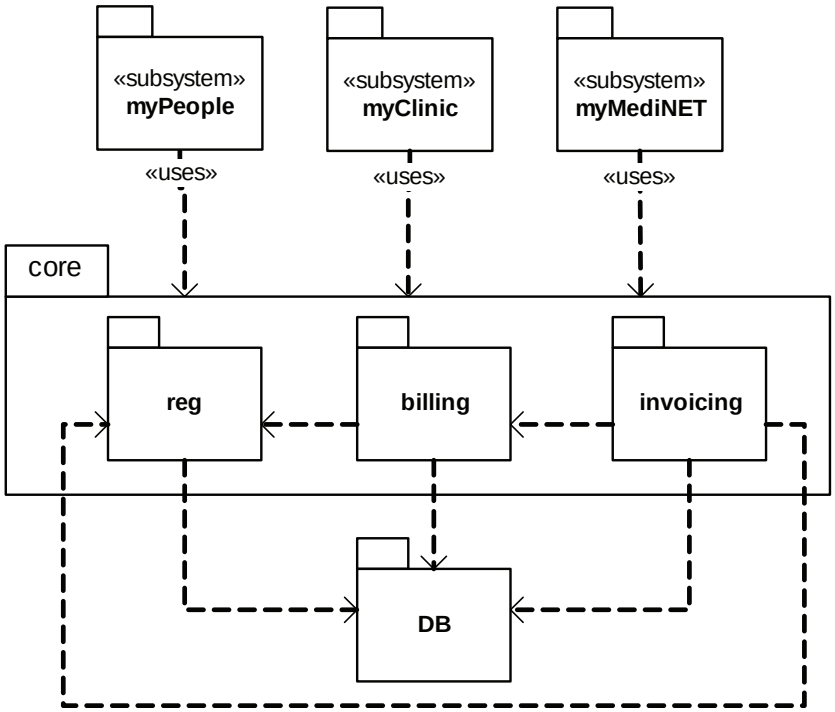
The Design Phase

The design phase involves the identification of application-specific and application-general subsystems. The application-specific subsystems are related to packages that group a set of closely related services in an application domain. The application-general subsystems are related to implementation technology decisions such as the introduction of user interface and database connectivity layers. The MediNet subsystems definition is shown in Figure 5.

The class design activity elaborates further the static and dynamic information of classes that were defined during the analysis phase. Additional information on the operations, attributes, and relationships can be added to each class. The specification of operations and attributes is made using the syntax of the chosen programming language. If necessary, the methods that specify the algorithm for the implementation of operations are specified.

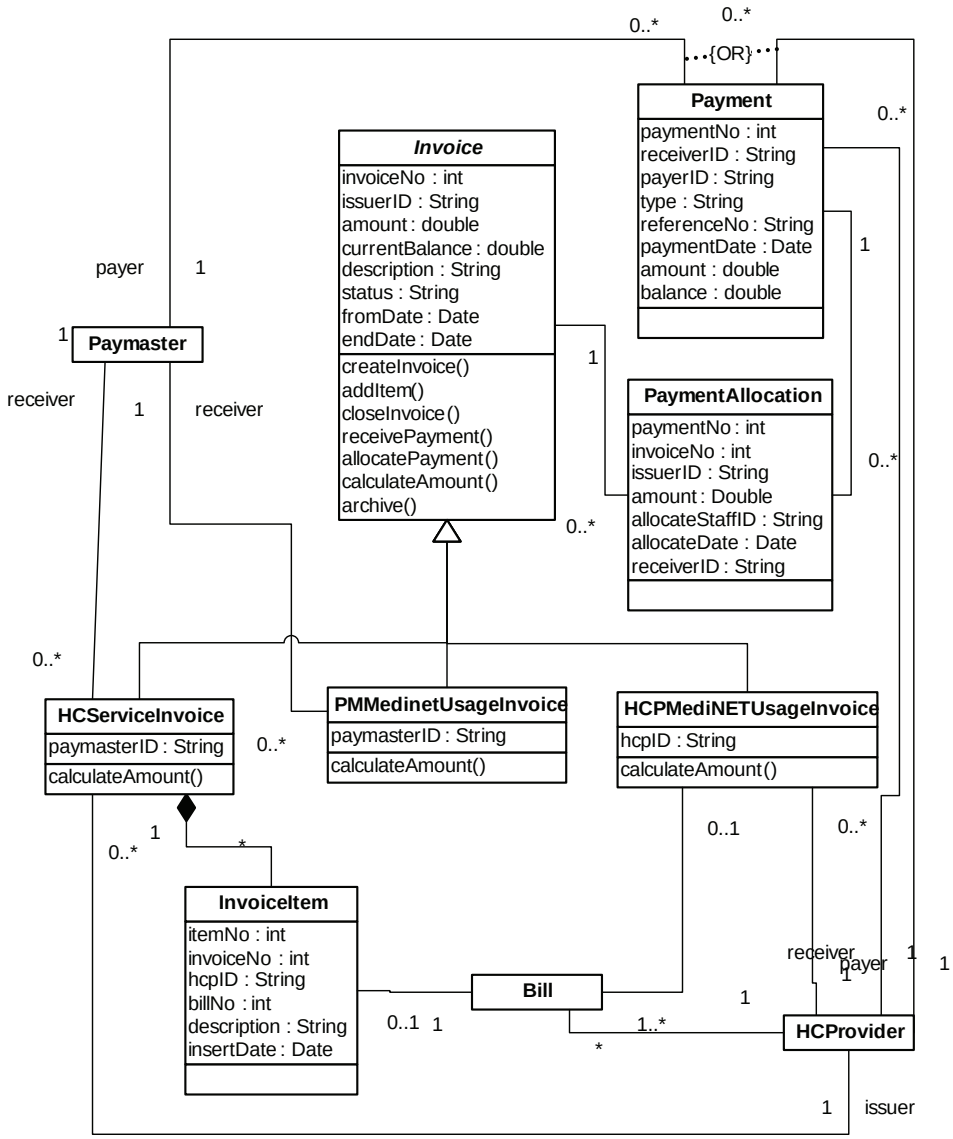
The class design activity for the MediNet application resulted in detailed specification of for the three packages of registration, billing and invoicing. The class association diagram of Figure 6 shows the class details for invoicing. In order to reduce diagrammatic complexity all parameters and return values are hidden in the class operations.

Figure 5. Software architecture for the MediNet application



The calculation of invoice amount is different for different types of invoice. The amount for healthcare service invoice is calculated as the total of its item amounts after applying additional computation rules such as bill limit, invoice limit and discount. MediNET uses the open item invoicing method that allows an invoice issuer to track each unpaid invoice as an individual item for aging purposes. Panel patient bills are considered as the items for HCP MediNET usage and HCP service usage invoices. For HCP MediNET usage invoice, the number of bills issued by a particular HCP is counted as the number of transactions, which is later used in the invoice amount calculation.

Figure 6. Class association diagram for invoicing for the MediNet application



In terms of payment, MediNET allows balance forward invoicing method in addition to open item method.

Within the design process classes are further elaborated in terms of the events and conditions that trigger their transition from one state to another. These are shown as statechart diagrams. For example, a statechart diagram for the `HCServiceInvoice` object is shown in Figure 7.

Within the BROOD design phase, rule phrase specifications are developed. Each rule phrase definition is stored in the repository called rule phrase entries. The possible values for rule phrase may be a set of enumerated values or the values of the linked software design element. A component engineer may define certain attributes for each business rule specification such as rule priority, owner, and business process. Each business rule

Figure 7. The STD `HCServiceInvoice` *object for the MediNet application*

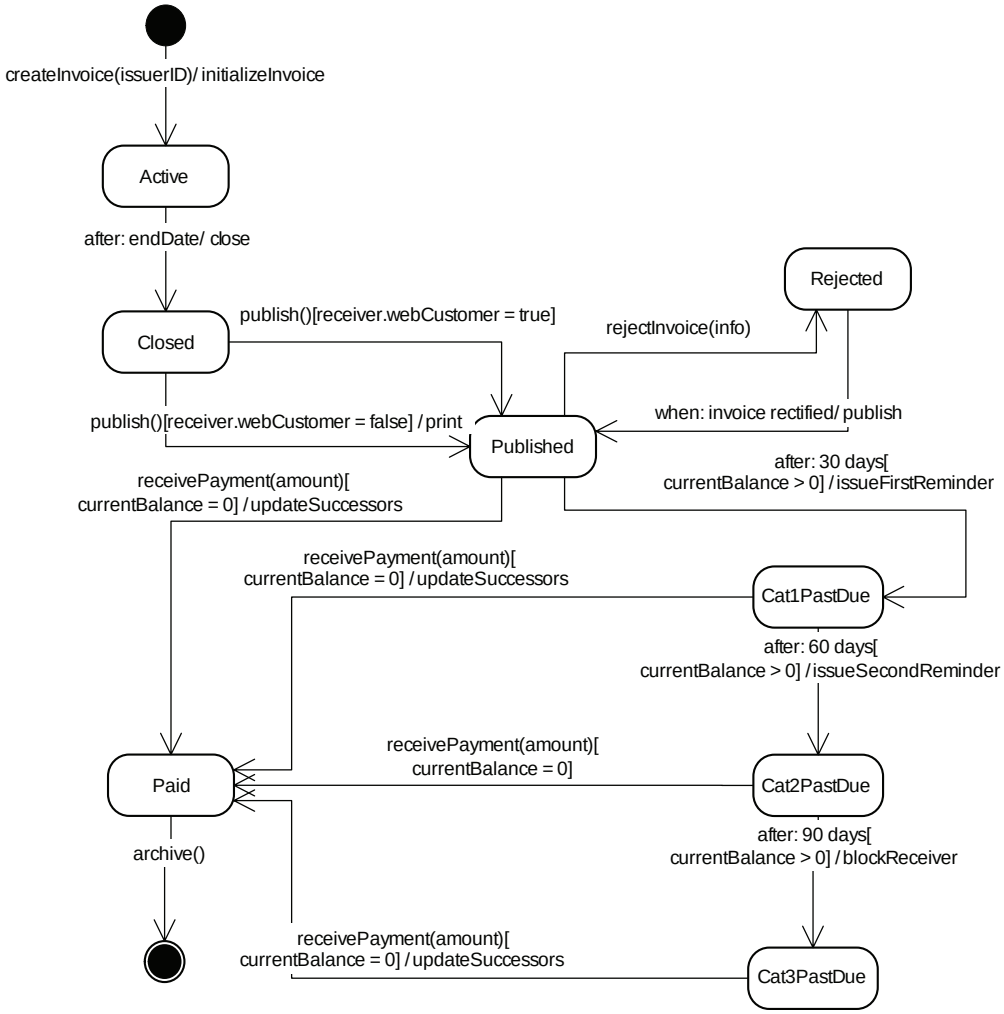


Table 6. Rule phrases and linked software design elements for the MediNet application

B Rule Category	Business Rule Phrases	Software Design Elements
Attribute Constraint	<entity> = 'a patient'	Patient (class)
	'must have'	- (patRegNo.optionality)
	'a unique'	- (patRegNo.uniqueness)
	<attributeTerm> = 'registration number'	Patient.patRegNo (attribute)
Relationship Constraint	<cardinality> = 'one and only one'	- (AssociationEnd.multiplicity)
	<entity> = 'transaction item'	TransItem (class)
	<role> = 'item type'	- (AssociationEnd.name)
	<entity> = 'bill item'	Bill_Item (class)
Action Assertion	<event> = '30 day after the creation date of the invoice'	- (Trans1.event.spec)
	<condition> = 'current balance of the invoice is greater than 0'	- (Trans1.guard.body)
	<action> = 'trigger issue the first reminder'	- (Trans1.action.initialiseInvoice(). spec)
Computation	<attributeTerm> = 'the amount of HCP MediNET Usage invoice'	HCPMediNETUsageInvoice.amount
	<algorithm> = 'the sum of monthly subscription fee plus transaction fee'	HCPMediNETUsageInvoice. calculateAmount().specification
Inference	<attributeTerm> = 'a paymaster status'	Paymaster.status
	<value> = 'under probation'	- (literal value)
	<condition> = 'the paymaster has an invoice with category 1 past due' AND 'the current balance is greater than RM 5,000.00'	Paymaster.getStatus().specification

statement can also be arranged in an appropriate rule set to assist the future management of the business rules.

For the MediNet application, the rules shown in Table 5 are specified according to rule phrases syntax as shown in Table 6.

The first rule in Table 6 shows the rule phrase derived from the attribute constraint rule, informally defined in the analysis phase as "A patient must have a unique registration number." The rule phrases '`a patient`' and '`registration number`' are respectively linked to `Patient` class and `patRegNo` attribute. The keywords '`must have`' and '`a unique`' are not statically linked to any design element. Instead, they are used to dynamically toggle the optionality and uniqueness values of `patRegNo` attribute during the creation or modification of the business rule statement. In other words, they are used to enable the automated change propagation to software design.

The second rule in Table 6 shows a relationship constraint., The rule phrases '`clinic item`'

and '`bill item`' are respectively linked to `TransItem` class and `Bill_Item` class. The rule phrases '`one and only one`' and '`clinic item`' play a similar role to keywords as in the attribute constraint rule, that is their purpose is to propagate business changes to design elements. The former specifies the multiplicity of an association end whilst the latter specifies the role of an association end.

In the *action assertion* rule "When a payment is not received within 30 days from the invoice date, then the first reminder will be sent," the rule phrases that represent the event, condition, and action are not directly linked to any design element but they are respectively used to generate the specifications of the transition's event, guard, and action in the HCP service usage invoice STD. Since event, condition, and action rule phrases are themselves composed by other rule phrases, they may be indirectly linked to the related design components via these rule phrases.

The *computation* and *inference* rules are linked to the operation specification —the computation rule is linked to the specification of `calculateAmount()` operation in `HCPMediNETUsageInvoice` class and the inference rule is linked to `getStatus()` operation from `Paymaster` class. During the development of an inference rule, a new operation is often needed to be added in its associated class to perform the derivation and return the inferred value.

The Evolution Phase

In general, business rule changes may be classified into simple and complex changes. A simple change is concerned with the modification, addition, or deletion of business rules that do not need to in-

Table 7. Simple change scenarios for the MediNet application

Change Scenarios	Changed Business Rules
1. HCP allows patients to make 'more than one payment for their bills' instead of the previously set 'single payment for each bill'.	One patient bill is associated with zero or more payments.
2. HCP makes small changes on the conditions to issue the reminder and block paymaster.	WHEN *15 days* from the invoice date IF a payment is not received THEN issue the first reminder. WHEN *30 days* from the invoice date IF the payment is not received THEN issue the second reminder. WHEN *45 days* from the invoice date IF the payment is not received THEN block the paymaster.
3. The MediNET supplier offers a more attractive usage charge to HCPs. They are charged based on the number of treated patients regardless the number of patient visits.	The amount of HCP usage invoice IS CALCULATED AS if (opt new package) then the transaction fee multiply by the number of registered patients, else, the transaction fee multiply by the number of treated patients, plus the monthly fee.
4. HCP introduces 5% discount to its internet customer.	If the paymaster is an internet customer, then give 5% discount to their invoices.
5. The HCP decides that each expense item must belong to one of the pre-defined types.	Zero or more expense item is associated with one and only one transaction item.

troduce new rule phrases or design elements. A complex change involves the addition or deletion of rule phrases or design elements.

Ordinarily, *simple business rules changes* could be performed by business users. The examples of five change scenarios that require simple business changes in MediNET system are shown in Table 7.

The implementation of a *complex* business rule change requires more effort than that of simple change. It involves the introduction of new rule phrases or design elements, which is needed to be performed by an individual with the knowledge of software design. In addition to technical skills, it often requires creative skills in making a design decision. Three examples of complex rules changes are shown in Table 8.

The first scenario initiates the modification of two existing business rule statements, the calculation of bill and the calculation of invoice amount. These business rule changes consequently lead to a minor change in software design, that is the introduction of `hasMaxBill` attribute in the `Paymaster` class.

Table 8. Complex change scenarios for the MediNet application

Change Scenarios	Changed Business Rules

1. HCP introduces new package for paymaster. In this package, the paymaster may limit the maximum amount of each patient bill to RM 20.00, and the excessive cost is absorbed by HCP. However, the paymaster must pay a monthly fee of RM5.00 for each patient.	The amount of a bill is computed as
	let amount = the sum of all amounts of bill items
	if (patient is a panel patient) AND (paymaster has maximum bill amount) AND (amount > RM 20.00)
	amount = 20
	The amount of HCP service invoice is computed as
	let amount = the total of the invoice items
	if (paymaster has maximum bill amount)
	amount = amount + 5 * the number of paymaster's patients

2. Paymaster wishes to provide different healthcare benefit coverage for different groups of its payees.	If (the patient is a panel patient) AND (the patient is an executive staff) then the patient is entitled to any type of treatments and medical procedures. If (the patient is a panel patient) AND (the patient is a production staff) then the patient is entitled for an outpatient treatment.

3. HCP would like to introduce a 5% discount on the invoices to preferred paymasters as a way to express gratitude to the loyal, potential, and good paying paymasters.	If (a paymaster has been a paymaster panel for more than 5 years) then (the customer is a 'loyal' customer).
	If (a paymaster has an average of at least RM24000.00 for the invoices over the last five years) then (the paymaster is considered as a 'potential' customer).
	If (a paymaster never has a past due invoice for the last two years) then (the paymaster is considered as a good paying paymaster).
	When (the invoice in created) if (the paymaster is a loyal, potential and good paying customer) then (set the discount of the invoice to 5%)

In the second scenario, the paymaster decided to introduce different healthcare benefit coverage to different levels of their payees. For example, executive staff is entitled to any medical treatment and medical procedures whilst production staff is only paid for outpatient treatments. It is obvious that simply implementing this new requirement into the existing `Paymaster` or `PanelPatient` class may increase the complexity of these classes. Therefore, additional classes that are responsible to manage the healthcare benefit coverage are required to be added to the existing software design. The possible candidates for these classes include `BenefitCoverage`, `SelectedClinic`, `MedicalProcedure`, and `Entitlement`.

The third scenario requires the intervention of a software developer. This scenario requires a number of new inference rules to be added to define a loyal, potential, and good paying customer. In addition to these business rules, an action assertion rule that initializes the value of the invoice discount during invoice creation should also be added. The introduction of the new inference rules consequently requires `isLoyal()`, `isPotential()`, and `isGoodPaying()` operations to be added to the `Paymaster` class. Similarly, the newly introduced action assertion rule requires component engineers to modify the action component of the transition from the initial state to '`Active`' state in the STD for `HCServiceInvoice` object.

THE BROOD SUPPORT TOOL

The BROOD process introduces several additional activities to the traditional object-oriented software design process. These additional activities include the documentation of business rules and their linking to software design components. To assist a developer with these BROOD-specific activities, a tool has been developed that supports the activities of business rule specification and management, software design editing, and business rule change propagation.

The BROOD tool was developed on top of the generic modeling environment (GME) (Ledeczi et al., 2001; VU, 2003), which is a configurable modeling environment.

The metamodel and templates, which are discussed in section 0, were used to implement the BROOD tool environment.

GME was used to visually edit the software design models, business rule specification, and rule phrase entries. Three main modules (known as interpreters in GME) were developed to simplify the rule phrase management, business rule composition, and business rule modification. These modules also perform the *automated propagation* of business rule changes to the respective software design elements, since a manual undertaking of such propagation would be impractical for most applications.

The BROOD tool has been designed to be used by both software developers and business users. A user-friendly interface is provided to ease the management and traceability of business rules by non-IT users. An overview of the BROOD support tool is shown in Figure 8.

The metamodel, the graphical model editor, the rule phrase management, the business rules composition and the business rules modification functions are part of the core component and user application layer in the BROOD tool architecture. The rule phrase entries, business rule specification, and software design models are stored in the storage layer.

The BROOD tool maintains the consistencies between business rule and the linked software design each time a business rule is created or modified. It provides full automated support in performing simple changes and partial support for complex changes since these require creative skills of software engineers in making a design decision.

There are four main types of model that can be managed using the BROOD tool: rule phrase entries, business rule, class diagram, and statechart diagram. Users may select the type of model to

Figure 8. Overview of the BROOD tool

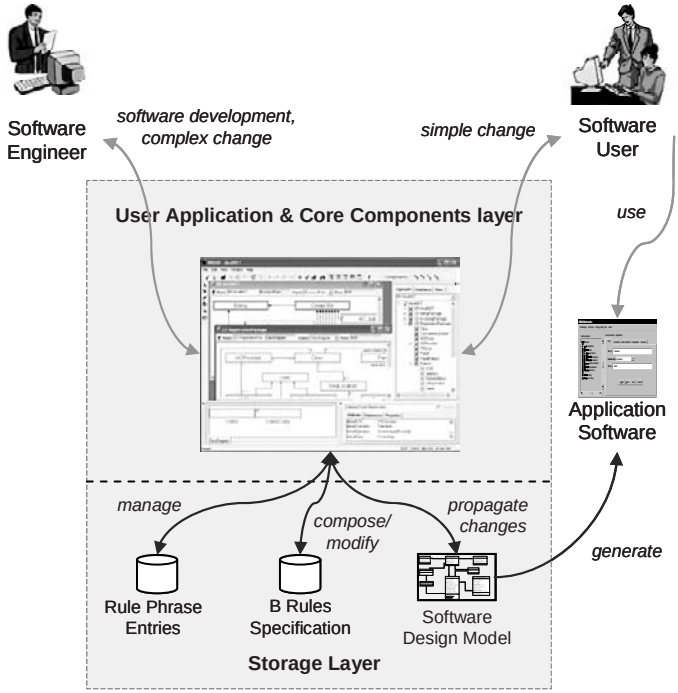
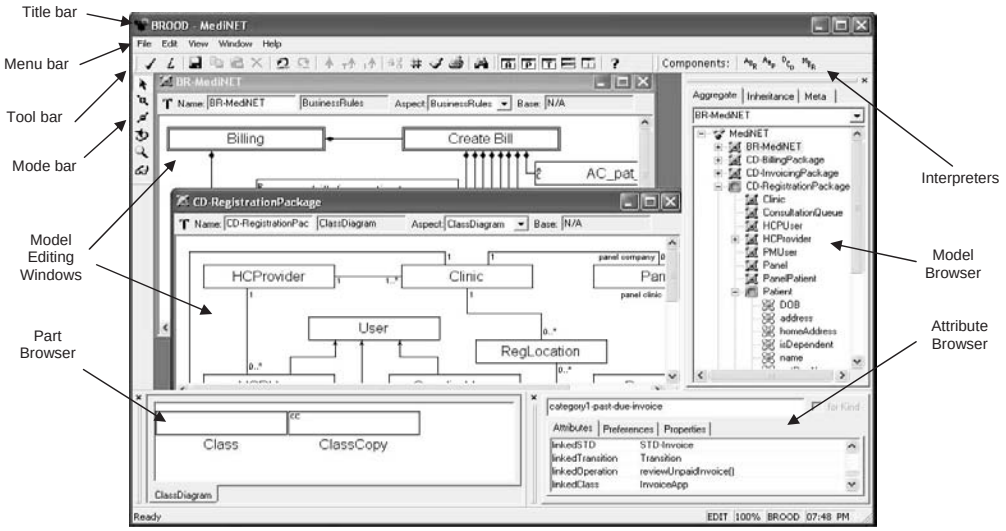


Figure 9. Example of the BROOD model editor



be created from a set of choices. An example of the BROOD model editor is shown in Figure 9. The model editor provides a convenient way to create a model and also to connect it or parts of it to other models.

While graphical model editing is convenient for visual models such as those of class and statechart diagrams, it is less helpful for business rules specification.

The graphical model editor can be used for some simple business rules definition such as cardinality, relational operator, list, and optionality but for more complex rules the BROOD tool offers a dedicated rule editor, the add business rule (ABR) module. This module performs two main tasks: business rule composition and software design updating. In business rule composition mode, rule phrases are used to construct a business rule statement. In software design updating mode the module updates the software design model that corresponds to the composed rule.

The BROOD tool also helps with the implementation of business rule changes. The modify business rule (MBR) module was developed to assist tool users in performing this task, an example of which is shown in Figure 9.

A full description of the tool is beyond the scope of this article. It should be stressed however, that the tool plays an important part in the effective application of the BROOD approach by simplifying a sometimes tedious, error-prone, and time-consuming task of linking and propagating business rule changes to software design components.

DISCUSSION

The main aim of BROOD has been to facilitate the process of software evolution through: (a) externalization of business rules and their explicit modeling and (b) the linking of each modeled business rule with a corresponding software component. This approach provides full traceability between end-user concepts and software designs. By combining BROOD to design traceability in source code (Alves-Foss, Conte de Leon, & Oman, 2002), it is possible to achieve effective traceability in a software system.

The BROOD metamodel offers a complete foundation and infrastructure for the development of a software system that is resilient to business rule changes.

With regard to business rule typology, BROOD introduces three main business rule types: constraints, action assertion, and derivations. These types are further divided into an adequate number of sub-types and templates. In contrast to BRG, BROCOM, and BRS approaches, BROOD attempts to remove the redundancy by reducing the unnecessary business rule types. At the same time, it improves the incompleteness of business rule types in AOM, coordination contract, and BRBeans approaches. In terms of business rule management elements, BROOD provides the concept of *ruleset* to organize the groups and hierarchy of the closely related business rules.

In terms of its modeling language, BROOD offers a high level of expressiveness. The keywords in the language definition and a sufficient number of sentence templates should provide adequate representation constructs. In general, achieving total expressiveness of the modeling language business rules is relatively hard to achieve due to the large number of ways of expressing business rules in a natural language. The usability of BROOD in this context will be proved in due course once the approach has been applied on different domains and applications. BROOD was found to have a high level of un-ambiguity by the introduction of the appropriate typology and templates. BROOD provides a mutually exclusive set of business rule types and removes the superfluous templates in order to avoid conflict and redundancy in representing the meaning of business rules.

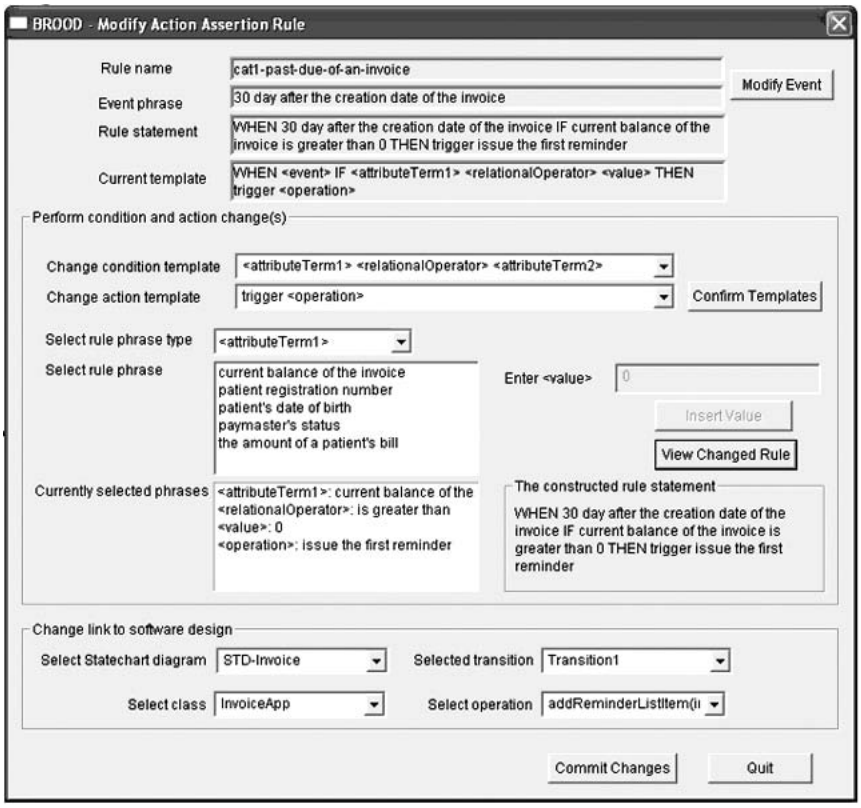
In practical terms, BROOD can be applied using the UML-based SPEM metamodel, which provides a set of concepts and notations to de-

scribe various software process components such as lifecycle phases, activities, process roles, and work products. The use of business rule templates and UML improves the usability of the BROOD approach. The templates allow users to create a business rule statement by simply composing the existing rule phrases whilst UML provides abstractions for users to naturally design a software system. Moreover, the detailed process description is provided to guide users especially in performing complex tasks such linking business rules to software design and handling different types of changes.

The utility of BROOD was demonstrated in this paper through the use of the MediNet industrial application. This application had originally been developed using a standard object-oriented approach. It was therefore possible (and indeed desirable) to use the case study not only as a way of demonstrating BROOD but also for comparing and contrasting BROOD to a traditional development approach.

By considering UML for software design, BROOD maintains the well-known object-oriented design quality attributes such as modularity, high cohesion, low coupling, efficiency, and portability. BROOD however provides additional quality attributes such as *requirements traceability*, *software evolvability*, and *approach usability*.

Figure 10. Example of the BROOD business rules modifier



The traditional approach deployed for MediNet did not provide explicit traceability of business policy defined during the requirements specification phase. Instead, it provides a so-called 'seamless transition' from the use case models that document the user requirements to the analysis and design models. This resulted in business rules being embedded in both requirements specification and software design models. In contrast, with BROOD there was a natural transformation of the MediNET requirements into the structured business rules specification and in turn this specification was directly related to software design components.

Concerning software evolution, the implementation of changes using the traditional approach required the use of expertise with specific knowledge of the MediNET software design. Since software engineers do not normally initiate business changes, they had to repeat all phases in MediNET development lifecycle especially requirements and analysis phases. Locating the related software design components was hard since there was no explicit link between the MediNET design models and its user requirements.

In relation to approach usability, the traditional approach was easier to apply during development since it did not have to deal with additional steps that were added to explicitly specify, document, and link business rules specification to software design. These steps were found to increase the complexity and duration of software development process. However, the availability of the business rule typology and templates, which provide the guidelines for the analysis of business rule statements and the identification of rule phrases, were found useful in minimizing these problems. The business rule templates have improved the MediNET system understandability and increased the involvement of business users in the MediNET development. During evolution, BROOD was found easier to be used than the traditional approach. Using BROOD, business users could perform the simple business rule changes as demonstrated in the MediNET application. Rapid change implementation is important especially in business critical applications with intolerable downtime. The detailed process description facilitated the implementation of complex changes in MediNET.

In summary, BROOD contributes to three critical areas namely business rules specification, object-oriented design, and software evolution process. The proposed business rule specification extends the state-of-the-art approaches to business rule representation by reducing redundancy and avoiding conflict among business rule types in its typology. The structures of rule templates have been defined so as to make them suitable for linking to software designs in support of future software evolution. A specification is aligned to changing user requirements via the linking of business rules to software designs through a detailed transformation of business rule into the specification of related software design components. Thus, the externalization of frequently changing aspects of a system into detailed business rules and the maintenance of associations between these and corresponding software components should provide a strong framework for effective software evolution.

ACKNOWLEDGMENT

The authors would like to thank the human resource department of Universiti Teknologi Malaysia (UTM) for partially sponsoring this research, and Penawar Medical Group, Malaysia for the permission to use its MediNET healthcare information system requirements specification as the case study. The authors wish to also express their gratitude to the three anonymous reviewers and to the editor of the special issue, Professor Dinesh Batra, whose insightful and detailed comments have contributed to the production of a much improved version of this article.

REFERENCES

Alves-Foss, J., Conte de Leon, D., & Oman, P. (2002). *Experiments in the use of xml to enhance traceability between object-oriented design specifications and source code.* Paper presented at the 35th Annual Hawaii International Conference on System Sciences.

Andrade, L., & Fiadeiro, J. (2000, October 15-19). *Evolution by contract.* Paper presented at the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications 2000, Workshop on Best-practice in Business Rules Design and Implementation, Minneapolis, Minnesota USA.

Andrade, L., Fiadeiro, J., Gouveia, J., & Koutsoukos, G. (2002). Separating computation, coordination and configuration. *Journal of Software Maintenance and Evolution: Research and Practice, 14*(5), 353-359.

Astley, M., & Agha, G. A. (1998, 20-21 April). *Modular construction and composition of distributed software architectures.* Paper presented at the Int. Symposium on Software Engineering, for Parallel and Distributed Systems, Kyoto, Japan.

Eriksson, H.-E., & Penker, M. (2000). *Business modelling with uml*: OMG Group, Wiley Computer Publishing, John Wiley & Sons, Inc.

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IEEE IT Professional, 2*(3), 17 - 23.

Evans, H., & Dickman, P. (1999, October). *Zones, contracts and absorbing change: An approach to software evolution.* Paper presented at the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '99), Denver, Colorado, USA.

Gottesdiener, E. (1997). Business rules show power, promise. *Application Development Trends, 4*(3, March 1997).

Grubb, P., & Takang, A. A. (2003). *Software maintenance: Concepts and practice*. Singapore: World Scientific Publishing.

Halle, B. V. (1994). Back to business rule basics. *Database Programming and Design*(October 1994), 15-18.

Hay, D., & Healy, K. A. (1997). *Business rules: What are they really?* GUIDE (The IBM User Group). Retrieved from http://www.Business-RulesGroup.org/):.

Hay, D., & Healy, K. A. (2000). *Defining business rules ~ what are they really?* (No. Rev 1.3): the Business Rules Group.

Herbst, H. (1996a). *Business rule oriented conceptual modelling.* Verlag: Physica .

Herbst, H. (1996b). Business rules in system analysis: A meta-model and repository system. *Information Systems, 21*(2), 147-166.

Herbst, H. (1997). *Business rule-oriented conceptual modeling.* Germany: Physica-Verlag.

IBM (Cartographer). (2003). *Ibm websphere application server enterprise*

Kovari, P., Diaz, D. C., Fernandes, F. C. H., Hassan, D., Kawamura, K., Leigh, D., et al. (2003). *Websphere application server enterprise v5 and programming model extensions: Websphere handbook series* (First Edition ed.): International Business Machines Corporation.

Krammer, M. I. (1997). Business rules: Automating business policies and practicies. *Distributed Computing Monitor*(May 1997).

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., et al. (2001, 17 May). *The generic modeling environment.* Paper presented at the Workshop on Intelligent Signal Processing, Budapest, Hungary.

Lehman, M. M., & Belady, L. A. (1985). *Program evolution: Processes of software change.* London: Academic Press, Inc.

Loucopoulos, P., & Layzell, P. J. (1986, 1987). *Rubric: A rule based approach for the development of information systems.* Paper presented at the 1st European workshop on fault diagnosis, reliability and related knowledge based approaches, Rhodes.

Martin, J. (1989). *Information engineering*: Prentice-Hall.

McMenamin, S. M., & Palmer, J. F. (1984). *Essential systems analysis.* Englewood Cliffs, NJ: Yourdon Press.

Mens, K., Wuyts, R., Bontridder, D., & Grijseels, A. (1998). *Tools and environments for business rules.* Paper presented at the ECOOP'98, Brussels, Belgium.

Moriaty, T. (1993). The next paradigm. *Database Programming and Design.*

OMG (Cartographer). (2001). *Omg unified modeling language specification*

OMG (Cartographer). (2002). *Software process engineering metamodel specification*

Oreizy, P., Medvidovic, N., & Taylor, R. N. (1998, April 19-25). *Architecture-based runtime software evolution.* Paper presented at the International Conference on Software Engineering 1998 (ICSE'98), Kyoto, Japan.

Riehle, D., Tilman, M., & Johnson, R. (2000). *Dynamic object model* (No. WUCS-00-29): Dept. of Computer Science, Washington University.

Robinson, K., & Berrisford, G. (1994). *Object-oriented ssadm*. Englewood Cliffs, NJ: Prentice Hall.

Rosca, D., Greenspan, S., Feblowitz, M., & Wild, C. (1997, January 1997). *A decision support methodology in support of the business rules lifecycle.* Paper presented at the International Symposium on Requirements Engineering (ISRE'97), Annapolis, MD.

Rosca, D., Greenspan, S., & Wild, C. (2002). Enterprise modeling and decision-support for automating the business rules lifecycle. *Automated Software Engineering, 9*(4), 361 - 404.

Rosca, D., Greenspan, S., Wild, C., Reubenstein, H., Maly, K., & Feblowitz, M. (1995, November 1995). *Application of a decision support mechanism to the business rules lifecycle.* Paper presented at the 10th Knowledge-Based Software Engineering Conference (KBSE95), Boston, MA.

Ross, R. G. (1997). *The business rule book: Classifying, defining and modelling rules*: Data Base Newsletter.

Ross, R. G., & Lam, G. S. W. (1999). *Ruletrack: The brs meta-model for rule management*: Business Rule Solutions, Inc.

Ross, R. G., & Lam, G. S. W. (2003). *The brs proteus^{tm} methodology* (Fourth ed.): Business Rule Solutions.

Rouvellou, I., Degenaro, I., Rasmus, K., Ehnebuske, D., & McKee, B. (1999, November 1-5). *Externalizing business rules from enterprise applications: An experience report.* Paper presented at the Conference on Object-Oriented Programming, Systems, Languages, and Applications, Denver, Colorado.

Rouvellou, I., Degenaro, L., Rasmus, K., Ehnebuske, D., & McKee, B. (2000, June). *Extending business objects with business rules.* Paper presented at the 33rd International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Europe 2000), Mont Saint-Michel/ St-Malo, France.

Schneider, J. (1999). *Components, scripts, and glue : A conceptual framework for software composition.* Bern:University of Bern.

van Assche, F., Layzell, P. J., Loucopoulos, P., & Speltinex, G. (1988). *Rubric: A rule-based representation of information system constructs.* Paper presented at the ESPRIT Conference, Brussels, Belgium.

VU (Cartographer). (2003). *Gme 3 user's manual*

Wan Kadir, W. M. N., & Loucopoulos, P. (2003, 23-26 June). *Relating evolving business rules to software design.* Paper presented at the International Conference on Software Engineering Research and Practice (SERP), Las Vegas, Nevada, USA.

Wan Kadir, W. M. N., & Loucopoulos, P. (2004). Relating evolving business rules to software design. *Journal of Systems Architecture, 50*(7), 367-382.

Yoder, J. W., Balaguer, F., & Johnson, R. (2001, October 14-18). *Adaptive object models for implementing business rules.* Paper presented at the Third Workshop on Best-Practices for Business Rules Design and Implementation, Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001), Tampa Bay, Florida, USA.

Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R., Subrahmanian, V. S., & Zicari, R. (1997). *Advanced database systems*: Morgan Kaufmann.

APPENDIX A. SPECIFICATION OF BROOD

Part I: The BROOD Metamodel

Business Rule Organisation and Typology

business_rule_model = rule_set, owner;
rule_set = (rule_set | rule_statement), {rule_set | rule_statement}, [owner], [business_process];
business_rule = (constraint | action_assertion | derivation), name, [is_mandatory], [priority], [is_propagatable];
Contraint
constraint = att_constraint | rel_constraint;
Attribute Constraint
att_constraint = entity, ('must have' | 'may have'), ['a unique'], att_term | att_term, ('must be' | 'may be'), relational_op, (value | att_term) | att_term, 'must be in', list;
att_term = attribute, 'of', entity;
Relationship Constraint
rel_constraint = ([cardinality], entity, 'is a/an', role, 'of', [cardinality], entity | [cardinality], entity, 'is associated with', [cardinality], entity | entity, ('must have' | 'may have'), [cardinality], entity | entity, 'is a/an' entity), *{Association}*;

Action Assertion

action_assertion = 'WHEN', event, ['IF', condition], 'THEN', action, {StatechartDiagram, Transition};
Event
event = simple_event | complex_event;
simple_event = (change_event | time_event | user_event), {Class, Operation};
change_event = att_term ('is updated' |) | entity ('is deleted' | 'is created') (operation | business_rule), 'is triggered';
time_event = date_time | n, time_unit, 'time interval from', date_time , 'is reached' | number, time_unit, 'after', date_time;
user_event = string;
complex_event = simple_event, {('Or' | 'And'), simple_event {('Or' | 'And'), simple_event}
Condition
condition = simple_condition | complex_condition;

simple_condition = [‘Not’], attribute_term, relational_op, (value | attribute_term) | attribute_term, (‘in’ | ‘not in’), list;
complex_condition = simple_condition, (‘Or’ | ‘And’), simple_condition, {(‘Or’ | ‘And’), simple_condition};
Action
action = simple_action | action_sequence;
simple_action = trigger_action | object_ manipulation_action | user_action, {Class, Operation};
trigger_action = ‘trigger’, (process | operation | business_rule);
object_manipulation_action = ‘set’, att_term, ‘to’, value | (‘create’ | ‘delete’), object;
action_sequence = simple_action, {simple_action};

Derivation

derivation = computation | inference;
Computation
computation = attribute_term, ‘is computed as’, algorithm, {Class, Operation};
algorithm = string;
(* i.e. any specification language for specifying the algorithm e.g., OCL, pseudo-code, etc. *)
Inference
inference = ‘If’, condition, ‘then’, fact;
fact = (attribute_term | entity), relational_op, [‘a’], value) | entity, (‘may’ | ‘may not’), action, {Class, Operation};

Rule Phrases / Linking Elements / Low-Level Definitions
(* Some low level non-terminal symbol such as <real>, <integer> and <string> are not defined. **** *)
entity = phrase, Class;
attribute = phrase, Class, Attribute;
operation = phrase, {Class, Operation};
cardinality = phrase, maxCard, minCard;
eventPhrase = phrase, event, {Class, Operation};
actionPhrase = phrase, action, {Class, Operation};
role = string;
list = string,{string};
phrase = string;
value= string | integer | real | date | time;
number = real | integer;
time_unit = ‘second’ | ‘minute’ | ‘hour’ | ‘day’ | ‘month’ | ‘year’;
relational_op = ‘equal’ | ‘not equal’ | ‘less than’ | ‘less than or equal’ | ‘greater than’ | ‘greater than or equal’;

name = string;
priority = ‘high’ | ‘medium’ | ‘low’;
is_mandatory = boolean;
is_propagatable = boolean;
boolean = ‘true’ or ‘false’;

Part II: The BROOD Process

The following specification is based on OMG Software Process Engineering Metamodel.

Process: Business Rule-based Object-Oriented Design (BROOD)

Phase: **Analysis**
Activity: **Analyze Business Rule Statements**
ProcessRole: **Functional Analyst**
ActivityParameters {kind: input}
WorkProduct: **Use-Case Model** {state: **revised**}
WorkProduct: **Business Rule Statements** {state: **revised**}
ActivityParameters {kind: output}
WorkProduct: **Business Rule Specification** {state: **initial draft**}
Steps
Step: **Identify business rule type**
Step: **Rewrite business rules according to sentence templates**
Step: **Resolve rule conflicts and redundancy**

Activity: **Architectural Analysis**
ProcessRole: **Software Architect**
ActivityParameters {kind: input}
WorkProduct: **Use-Case Model** {state: **revised**}
WorkProduct: **Business Model** {state: **completed**}
WorkProduct: **Architecture Description** {state: **initial draft**}
WorkProduct: **Supplementary Requirements** {state: **revised**}
ActivityParameters {kind: output}
WorkProduct: **Analysis Class Diagram** {state: **outline**}
WorkProduct: **Analysis Package** {state: **outline**}
WorkProduct: **Architecture Description** {state: **revised draft**}
Steps
Step: **Identify analysis packages**
Step: **Identify analysis classes**
Step: **Describe analysis object interactions**

Activity: **Analyze a Class**
ProcessRole: **Component Engineer**

ActivityParameters {kind: input}
WorkProduct: **Analysis Class Diagram** {state: **outlined**}
ActivityParameters {kind: output}
WorkProduct: **Analysis Class Diagram** {state: **completed**}
Steps
Step: **Identify class responsibilities**
Step: **Identify class attributes**
Step: **Identify class relationships**

Activity: **Analyze a Package**
ProcessRole: **Component Engineer**
ActivityParameters {kind: input}
WorkProduct: **Analysis Package** {state: **outlined**}
WorkProduct: **Architecture Description** {state: **revised draft**}
ActivityParameters {kind: output}
WorkProduct: **Analysis Package** {state: **completed**}
Steps
Step: **Analyze the cohesiveness of each package**
Step: **Analyze the dependencies between packages**
Phase: **Design**
Activity: **Architectural Design**
ProcessRole: **Software Architect**
ActivityParameters {kind: input}
WorkProduct: **Use-Case Model** {state: **revised**}
WorkProduct: **Analysis Model** {state: **completed**}
WorkProduct: **Architecture Description** {state: **revised draft**}
WorkProduct: **Supplementary Requirements** {state: **revised**}
ActivityParameters {kind: output}
WorkProduct: **Design Class Diagram** {state: **outline**}
WorkProduct: **Design Package** {state: **outline**}
WorkProduct: **Architecture Description** {state: **revised**}
Steps
Step: **Identify subsystems and their interfaces**
Step: **Identify architectural significant classes**
Step: **Identify generic design mechanisms**

Activity: **Design a Class**
ProcessRole: **Component Engineer**
ActivityParameters {kind: input}
WorkProduct: **Design Class Diagram** {state: **outlined**}
ActivityParameters {kind: output}
WorkProduct: **Design Class Diagram** {state: **completed**}

WorkProduct: **Design Statechart Diagram** {state: **completed**}
Steps
Step: **Identify operations**
Step: **Identify attributes**
Step: **Identify relationships**
Step: **Describe method**
Step: **Describe state**
Step: **Link Statechart diagram element to class diagram**

Activity: **Design a Sub-System**
ProcessRole: **Component Engineer**
ActivityParameters {kind: input}
WorkProduct: **Sub-System** {state: **outlined**}
WorkProduct: **Architecture Description** {state: **revised**}
ActivityParameters {kind: output}
WorkProduct: **Sub-System** {state: **completed**}
Steps
Step: **Design sub-system dependencies**
Step: **Design sub-system interfaces**

Activity: **Develop Business Rule Specifications**
ProcessRole: **Component Engineer**
ActivityParameters {kind: input}
WorkProduct: **Business Rule Specifications** {state: **initial draft**}
ActivityParameters {kind: output}
WorkProduct: **Business Rule Specifications** {state: **revised draft**}
Steps
Step: **Define rule phrases**
Step: **Link rule phrase to design elements**
Step: **Form structured rule statements**
Step: **Populate rule attributes**
Step: **Organize rule set**

Activity: **Validate Business Rule Specifications**
ProcessRole: **Functional Analyst / Business User**
ActivityParameters {kind: input}
WorkProduct: **Business Rule Specifications** {state: **revised draft**}
ActivityParameters {kind: output}
WorkProduct: **Business Rule Specifications** {state: **completed**}
Steps
Step: **Ensure correctness of business rule specifications**
Step: **Ensure understandability of business rule specifications**

Phase: **Evolution**

Activity: **Examine Business Rule Change Request**
ProcessRole: **Business User / Functional Analyst**
ActivityParameters {kind: input}
WorkProduct: **Business Rule Change Request** {state: **initial**}
WorkProduct: **Business Rule Specifications** {state: **completed}**
ActivityParameters {kind: output}
WorkProduct: **Business Rule Change Request** {state: **revised**}
Steps
Step: **Determine the type of business rule change**
Step: **Revise business rule change request (for complex change)**

Activity: **Perform Business Rule Modification Change**
ProcessRole: **Business User / Functional Analyst**
ActivityParameters {kind: input}
WorkProduct: **Design Model** {state: **completed}**
ActivityParameters {kind: output}
WorkProduct: **Design Model** {state: **changed}**
Steps
Step: **Locate the relevant business rule specification**
Step: **Perform change on business rule specification**
Step: **Propagate change to software design**

Activity: **Analyze Business Rule Change Request**
ProcessRole: **Software Architect**
ActivityParameters {kind: input}
WorkProduct: **Business Rule Change Request** {state: **revised**}
WorkProduct: **Design Model** {state: **completed}**
ActivityParameters {kind: output}
WorkProduct: **Business Rule Change Plan** {}
Steps
Step: **Identify the effect of changes**
Step: **Produce the detailed change plan**

Activity: **Implement Business Rule Change**
ProcessRole: **Component Engineer**
ActivityParameters {kind: input}
WorkProduct: **Business Rule Change Plan** {}
ActivityParameters {kind: output}
WorkProduct: **Design Model** {state: **changed**}
Steps
Step: **Review the change plan**
Step: **Perform the changes**

APENDIX B. THE MEDINET APPLICATION

MediNET is a suite of internet applications that addresses the administrative and back-end processing requirements of the healthcare business community. It acts as a secondary layer to the existing administrative and information systems. MediNET allows various components of the healthcare industry to exchange business data instantaneously and automate their routine administrative tasks. Therefore, facilitated businesses are able to reduce their administrative burdens, become more efficient and make better informed business decisions. In contrast to the traditional applications, MediNET does not require its users to maintain separately installed software. It allows its users to leverage the power of technology without having to bear massive development, acquisition, infrastructure or maintenance costs.

MediNET users only need to pay as and when they use the application. In general, MediNET users can be divided into three categories: paymasters, healthcare providers (HCPs), and supplier. Paymasters are those who pay for medical or healthcare services, for examples employers, insurers and managed care organizations. They use MediNET to maintain the basic parts of the patient records such as performing their payee registration and defining the healthcare benefit coverage of their payees. HCPs are the professionals who dispense medical treatment, for examples general practitioners (GPs), hospitals and dentists. HCPs use MediNET to manage patient records, patient billing and paymaster invoicing.

The current implementation of MediNET is only limited to employers as the paymasters and GPs as the HCPs. The supplier is the company who owns, provides and maintains the MediNET application. It rents MediNET to HCPs and paymasters as and when the applications are needed and charges them based on the number of performed transactions.

MediNET was chosen as a case study due to the various frequently changing business rules introduced by its different users. For example, HCPs provide different packages to the paymasters that constrain the way they perform the patient billing and paymasters invoicing. Paymasters may also want to introduce different healthcare benefit coverage to different staff levels that control the eligibility of the staff's treatments. The business rules related to the packages and benefit coverage are frequently changed by the HCPs and paymasters. Other common changes to business rules include the introduction of invoice discounts, the rule to block non-paying paymasters, and the conditions to issue reminder for past due invoices. These frequent changes indicate the need for an approach to simplify the implementation of changes in MediNET software system.

This work was previously published in the Journal of Database Management, edited by K. Siau, Volume 19, Issue 1, pp. 41-73, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 2.29
Bug Fixing Practices within Free/Libre Open Source Software Development Teams[1]

Kevin Crowston
Syracuse University, USA

Barbara Scozzi
Politecnico di Bari, Italy

ABSTRACT

Free/Libre open source software (FLOSS, e.g., Linux or Apache) is primarily developed by distributed teams. Developers contribute from around the world and coordinate their activity almost exclusively by means of email and bulletin boards, yet some how profit from the advantages and evade the challenges of distributed software development. In this article we investigate the structure and the coordination practices adopted by development teams during the bug-fixing process, which is considered one of main areas of FLOSS project success. In particular, based on a codification of the messages recorded in the bug tracking system of four projects, we identify the accomplished tasks, the adopted coordination mechanisms, and the role undertaken by both the FLOSS development team and the FLOSS community. We conclude with suggestions for further research.

INTRODUCTION

In this article, we investigate the coordination practices for software bug fixing in Free/Libre open source software (FLOSS) development teams. Key to our interest is that most FLOSS software is developed by distributed teams, that is, geographically dispersed groups of individuals working together over time towards a common goal (Ahuja et al., 1997, p. 165; Weisband, 2002). FLOSS developers contribute from around the world, meet face to face infrequently, if at all, and coordinate their activity primarily by means of computer mediated communications (Raymond, 1998; Wayner, 2000). As a result, distributed teams

employ processes that span traditional boundaries of place and ownership. Since such teams are increasingly commonly used in a diversity of settings, it is important to understand how team members can effectively coordinate their work.

The research literature on distributed work and on software development specifically emphasizes the difficulties of distributed software development, but the case of FLOSS development presents an intriguing counter-example, at least in part: a number of projects have been outstandingly successful. What is perhaps most surprising is that FLOSS development teams seem not to use many traditional coordination mechanisms such as formal planning, system level design, schedules and defined development processes (Mockus et al., 2002, p. 310). As well, many (though by no means all) programmers contribute to projects as volunteers, without working for a common organization and/or being paid.

The contribution of this article is to document the process of coordination in effective FLOSS teams for a particularly important process, namely bug fixing. These practices are analyzed by adopting a process theory, that is, we investigate which tasks are accomplished, how and by whom they are assigned, coordinated, and performed. In particular, we selected four FLOSS projects, inductively coded the steps involved in fixing various bugs as recorded in the projects' bug tracking systems and applied coordination theory to identify tasks and coordination mechanisms carried out within the bug-fixing process.

Studying coordination of FLOSS processes is important for several reasons. First, FLOSS development is an important phenomenon deserving of study for itself. FLOSS is an increasingly important commercial issue involving all kind of software firms. Million of users depend on systems such as Linux and the Internet (heavily dependent on FLOSS software tools) but as Scacchi notes "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success" (Scacchi, 2002, p. 1; Scacchi, 2005). Understanding the reasons that some projects are effective while others are not is a further motivation for studying the FLOSS development processes. Second, studying how distributed software developers coordinate their efforts to ensure, at least in some cases, high-performance outcomes has both theoretical and managerial implications. It can help understanding coordination practices adopted in social collectives that are not governed, at least apparently, by a formal organizational structure and are characterized by many other discontinuities that is, lack of coherence in some aspects of the work setting: organization, function, membership, language, culture, etc. (Watson-Manheim et al., 2002). As to the managerial implications, distributed teams of all sorts are increasingly used in many organizations. The study could be useful to managers that are considering the adoption of this organizational form not only in the field of software development.

The remainder of the article is organized as follows. In Section 2 we discuss the theoretical background of the study. In Section 3 we stress the relevance of process theory so explaining why we adopted such a theoretical approach. We then describe coordination theory and use it to describe the bug-fixing process as carried out in traditional organizations. The research methodology adopted to study the bug-fixing process is described in Section 4. In Section 5 and 6 we describe and discuss the study's results. Finally, in Section 7 we draw some conclusions and propose future research directions.

BACKGROUND

In this section we provide an overview of the literature on software development in distributed environment and the FLOSS phenomenon.

Distributed Software Development

Distributed teams offer numerous potential benefits, such as the possibility to perform different projects all over the world without paying the costs associated with travel or relocation, or ease of reconfiguring teams to quickly respond to changing business needs (DeSanctis & Jackson, 1994; Drucker, 1988) or to exploit available competences and distributed expertise (Grinter et al., 1999; Orlikowski, 2002). Distributed teams seem particularly attractive for software development, because software, as an information product, can be easily transferred via the same systems used to support the teams (Nejmeh, 1994; Scacchi, 1991). Furthermore, while many developed countries face a shortage of talented software developers, some developing countries have a pool of skilled professionals available, at lower cost (Metiu & Kogut, 2001, p. 4; Taylor, 1998). As well, the need to have local developers in each country for marketing and localization have made distributed teams a business need for many global software corporations (Herbsleb & Grinter, 1999b, p. 85).

While distributed teams have many potential benefits, distributed workers face many real challenges. The specific challenges vary from team to team, as there is a great diversity in their composition and in the setting of distributed work. As mentioned, distributed work is characterized by numerous discontinuities that generate difficulties for members in making sense of the task and of communications from others, or produce unintended information filtering (de Souza, 1993). These interpretative difficulties make it hard for team members to develop a shared mental model of the developing project (Curtis et al., 1990, p. 52). A lack of common knowledge about the status, authority and competencies of participants brought together for the first time can be an obstacle to the creation of a social structure and the development of team norms (Bandow, 1997, p. 88) and conventions (Weisband, 2002), thus frustrating the potential benefits of increased flexibility.

Numerous studies have investigated social aspects of software development teams (e.g., Curtis et al., 1988; Humphrey, 2000; Sawyer & Guinan, 1998; Walz et al., 1993). These studies conclude that large system development requires knowledge from many domains, which is thinly spread among different developers (Curtis et al., 1988). As a result, large projects require a high degree of knowledge integration and the coordinated efforts of multiple developers (Brooks, 1975). However, coordination is difficult to achieve as software projects are non-routine, hard to decompose perfectly and face requirements that are often changing and conflicting, making development activities uncertain.

Unfortunately, the problems of software development seem to be exacerbated when development teams work in a distributed environment with a reduced possibility for informal communication (Bélanger, 1998; Carmel & Agarwal, 2001; Herbsleb & Grinter, 1999a). In response to the problems created by discontinuities, studies of distributed teams stress the need for a significant amount of time spent in "community building" (Butler et al., 2002). In particular, members of distributed teams need to learn how to communicate, interact and socialize using CMC. Successful distributed cross-functional teams share knowledge and information and create new practices to meet the task-oriented and social needs of the members (Robey et al., 2000). Research has shown the importance of formal and informal adopted coordination mechanisms, information sharing for coordination and communications, and conflict management for project's performance and quality (Walz et al., 1993). However, the processes of coordination suitable for distributed teams are still open topics for research (e.g., Orlikowski, 2002).

The FLOSS Phenomenon: A Literature Overview

The growing literature on FLOSS has addressed a variety of questions. Some researchers have examined the implications of free software from economic and policy perspectives (e.g., Di Bona et al., 1999; Kogut & Metiu, 2001; Lerner & Tirole, 2001) as well as social perspective (e.g., Bessen, 2002; Franck & Jungwirth, 2002; Hann et al., 2002; Hertel et al., 2003; Markus et al., 2000). Other studies examine factors for the success of FLOSS projects (Hallen et al., 1999; Leibovitch, 1999; Pfaff, 1998; Prasad, n.d.; Valloppillil, 1998; Valloppillil & Cohen, 1998, Crowston and Scozzi, 2003). Among them, an open research question deals with the analysis of how the contributions of multiple developers can be brought into a single working product (Herbsleb & Grinter, 1999b). To answer such a question, a few authors have investigated the processes of FLOSS development (e.g., Jensen & Scacchi, 2005; Stewart & Ammeter, 2002). The most well-known model developed to describe FLOSS organization structure is the bazaar metaphor proposed by Raymond (1998). As in a bazaar, FLOSS developers autonomously decide the schedule and contribution modes for software development, making a central coordination action superfluous. While still popular, the bazaar metaphor has been broadly criticized (e.g., Cubranic, 1999). According to its detractors, the bazaar metaphor disregards some aspects of the FLOSS development process, such as the importance of the project leader control, the existence of de-facto hierarchies, the danger of information overloads and burnout, the possibility of conflicts that cause a loss of interest in a project or forking, and the only apparent openness of these communities (Bezroukov, 1999a, 1999b).

Nevertheless, many features of the bazaar model do seem to apply. First, many teams are largely self-organizing, often without formally appointed leaders or formal indications of rank or role. Individual developers may play different roles in different projects or move from role to role as their involvement with a project changes. For example, a common route is for an active user to become a co-developer by contributing a bug fix or code for a new feature, and for active and able co-developers to be invited to become members of the core. Second, coordination of project development happens largely (though not exclusively) in a distributed mode. Members of a few of the largest and most well-established projects do have the opportunity to meet face-to-face at conferences (e.g., Apache developers at *ApacheCon*), but such an opportunity is rare for most project members. Third, non-member involvement plays an important role in the success of the teams. Non-core developers contribute bug fixes, new features or documentation, provide support for new users and fill a variety of other roles in the teams. Furthermore, even though the core group provides a form of leadership for a project, they do not exercise hierarchical control. A recent study documented that self-assignment is a typical coordination mechanism in FLOSS projects and direct assignment are nearly non-existent (Crowston et al., 2005). In comparison to traditional organizations then, more people can share power and be involved in FLOSS project activities. However, how these diverse contributions can be harnessed to create a coherent product is still an important question for research. Our article addresses this question by examining in detail a particular case, namely, coordination of bug-fixing processes.

CONCEPTUAL DEVELOPMENT

In this section, we describe the theoretical perspectives we adopted to examine the coordination of bug fixing, namely, a process-oriented perspective and the coordination theory. We also introduce the topic of coordination and discuss

the literature on coordination in software development and the (small) literature on coordination in FLOSS teams.

Processes as Theories

Most theories in organizational and information system research are variance theories, comprising constructs or variables and propositions or hypotheses linking them. By adopting a statistical approach, such theories predict the levels of dependent or outcome variables from the levels of independent or predictor variables, where the predictors are seen as necessary and sufficient for the outcomes. In other words, the logical structure of such theories is that if concept *a* implies concept *b*, then more of *a* means more (or less) of *b*. For example, the hypothesis that the adoption of ICT makes organization more centralized, examined as a variance theory, is that the level of organization centralization increases with the number of new ICTs adopted.

An alternative to a variance theory is a process theory (Markus & Robey, 1988). Rather than relating levels of variables, process theories explain how outcomes of interest develop through a sequence of events. In that case, antecedents are considered as necessary but not sufficient for the outcomes (Mohr, 1982). For example, a process model of ICT and centralization might posit several steps each of which must occur for the organization to become centralized, such as development and implementation of an ICT system and use of the system to control decision premises and program jobs, resulting in centralization of decision making as an outcome (Pfeffer, 1978). However, if any of the intervening steps does not happen, a different outcome may occur. For example, if the system is used to provide information directly to lower-level workers, decision making may become decentralized rather centralized (Zuboff, 1988). Of course, theories may contain some aspects of both variance and process theories (e.g., a variance theory with a set of contingencies), but for this discussion, we describe the pure case. Typically, process theories are of some transient process leading to exceptional outcomes, for example, events leading up to an organizational change or to acceptance of a system. However, we will focus instead on what might be called "everyday" processes: those performed regularly to create an organization's products or services. For example, Sabherwal and Robey (1995) described and compared the processes of information systems development for 50 projects to develop five clusters of similar processes.

Kaplan (1991, p. 593) states that process theories can be "valuable aids in understanding issues pertaining to designing and implementing information systems, assessing their impacts, and anticipating and managing the processes of change associated with them". The main advantage of process theories is that they can deal with more complex causal relationships than variance theories. Also they embody a fuller description of the steps by which inputs and outputs are related, rather than noting the relationship between the levels of input and output variables. Specifically, representing a process as a sequence of activities provides insight into the linkage between individual work and processes, since individuals perform the various activities that comprise the process. As individuals change what they do, they change how they perform these activities and thus their participation in the process. Conversely, process changes demand different performances from individuals. ICT use might simply make individuals more efficient or effective at the activities they have always performed. However, an interesting class of impacts involves changing which individuals perform which activities and how activities are coordinated. Such an analysis is the aim of this article.

Coordination of Processes

In this subsection, we introduce the topic of coordination and present the fundamentals of coordination theory. Studying coordination means analyzing how dependences that emerge among the components of a system are managed. That stands for any kind of system, for example, social, economics, organic, or information system. Hence, the coordination of the components of a system is a phenomenon with a universal relevance (Boulding, 1956). The above definition of coordination is consistent with the large body of literature developed in the field of organization theory (e.g., Galbraith, 1973; Lawrence & Lorsch, 1967; Mintzberg, 1979; Pfeffer & Salancik, 1978; Thompson, 1967) that emphasizes the importance of interdependence.

For example, according to Thompson (1967), organizational action consists of the coordination of the interdependences and the reduction of the costs associated to their management. Two components/systems are said to be interdependent if the action carried out by one of them affect the other one's output or performance (McCann & Ferry, 1979; Mohr, 1971; Victor & Blackburn, 1987). For space reason, it is not possible to present all the contributions on coordination in the literature, but because of its relevance, we here briefly report on Thompson's seminal work. Thompson (1967) identified three main kinds of interdependence, namely *pooled*, *sequential* and *reciprocal interdependence*. Pooled interdependence occurs among organization units that have the same goal but do not directly collaborate to achieve it. Sequential dependence emerges among serial systems. A reciprocal dependence occurs when the output of a system is the input for a second system and vice versa. The three kinds of interdependence require coordination mechanisms whose cost increases going from the first to the last one. The coordination by standardization, that is, routine and rules, is sufficient to manage pooled-dependant systems. Coordination by plan implies the definition of operational schemes and plans. It can be used to manage pooled and sequential dependences. Finally, coordination by mutual adjustment is suitable for the management of reciprocal dependences.

The interest devoted by scholars and practitioners to the study of coordination problems has recently increased due to the augmented complexity of products, production processes and to the rapid advancement in science and technology. To address these issues scholars have developed coordination theory, a systemic approach to the study of coordination (Malone & Crowston, 1994). Coordination theory synthesizes the contributions proposed in different disciplines to develop a systemic approach to the study of coordination. Studies on coordination have been developed based on two level of analysis, a micro and a macro level. In particular, most organization studies adopt a macro perspective, so considering dependencies emerging among organizational units. Other studies adopt a micro perspective, so considering dependencies emerging among single activities/actors. Coordination theory adopts the latter perspective and, in particular, focuses on the analysis of dependencies among activities (rather that actors). Hence, it is particularly useful to the description and analysis of organizational processes, which can be defined as a set of interdependent activities aimed to the achievement of a goal (Crowston, 1997; Crowston & Osborn, 2003). In particular, this approach has the advantage of making it easier to model the effects of reassignments of activities to different actors, which is common in process redesign efforts. We adopted this perspective because the study focuses on analyzing coordination mechanisms within processes.

Consistent with the definition proposed above, Malone and Crowston (1994) analyzed group action in terms of *actors* performing *interdependent tasks.* These tasks might require or create

resources of various types. For example, in the case of software development, actors include the customers and various employees of the software company. Tasks include translating aspects of a customer's problem into system requirements and code, or bug reports into bug fixes. Finally, resources include information about the customer's problem and analysts' time and effort. In this view, actors in organizations face *coordination problems* arising from dependencies that constrain how tasks can be performed.

It should be noted that in developing this framework, Malone and Crowston (1994) describe coordination mechanisms as relying on other necessary group functions, such as decision making, communications, and development of shared understandings and collective sensemaking (Britton et al., 2000; Crowston & Kammerer, 1998). To develop a complete model of a process would involve modeling all of these aspects: coordination, decision making, and communications. In this article though, we will focus on the coordination aspects, bracketing the other phenomenon.

Coordination theory classifies dependencies as occurring between a task and a resource, among multiple tasks and a resource, and among a task and multiple resources. Dependencies between a task and a resource are due to the fact that a task uses or creates a resource. Shared use of resources can in turn lead to dependencies between the tasks that use or create the resource. These dependencies come in three kinds. First, the flow dependence resembles the Thompson's sequential dependency. Second, the fit dependence occurs when two activities collaborate in the creation of an output (though in the case where the output is identical, this might better be called synergy, since the benefit is that duplicate work can be avoided). Finally, the share dependency emerges among activities that share the use of a resource. Dependencies between a task and multiple resources are due to the fact that a task uses, creates or produces multiple resources or a task uses a resource and create another resource. For example, in the case of software development, a design document might be created by a design task and used by programming tasks, creating a fit dependency, while two development tasks might both require a programmer (a share dependency) and create outputs that must work together (a fit dependency).

The key point in this analysis is that dependencies can create problems that require additional work to manage (or provide the opportunity to avoid duplicate work). To overcome the coordination problems created by dependences, actors must perform additional work, which Malone and Crowston (1994) called *coordination mechanisms.* For example, if particular expertise is necessary to perform a particular task (a task-actor dependency), then an actor with that expertise must be identified and the task assigned to him or her. There are often several coordination mechanisms that can be used to manage a dependency. For example, mechanisms to manage the dependency between an activity and an actor include (among others): (1) having a manager pick a subordinate to perform the task; (2) assigning the task to the first available actor; and (3) having a labour market in which actors bid on jobs. To manage a usability subdependency, the resource might be tailored to the needs of the consumer (meaning that the consumer has to provide that information to the producer) or a producer might follow a standard so the consumer knows what to expect. Mechanisms may be useful in a wide variety of organizational settings. Conversely, organizations with similar goals achieved using more or less the same set of activities will have to manage the same dependencies, but may choose different coordination mechanisms, thus resulting in different processes. Of course, the mechanisms are themselves activities that must be performed by some actors, and so adding coordination mechanisms to a process may create additional dependences that must themselves be managed.

Coordination in Software Development

Coordination has long been a key issue in software development (e.g., Brooks, 1975; Conway, 1968; Curtis et al., 1988; Faraj & Sproull, 2000; Kraut & Streeter, 1995; Parnas, 1972). For example, Conway (1968) observed that the structure of a software system mirrors the structure of the organization that develops it. Both Conway (1968) and Parnas (1972) studied coordination as a crucial part of software development. Curtis et al. (1988) found that in large-scale software project, coordination and communication are among the most crucial and hard-to-manage problems. To address such problems, software development researchers have proposed different coordination mechanisms such a planning, defining and following a process, managing requirements and design specifications, measuring process characteristics, organizing regular meetings to track progress, implementing workflow systems, among the others.

Herbsleb and Grinter (1999b), in a study of geographically-distributed software development within a large firm, showed that some of the previously mentioned coordination mechanisms—namely integration plans, component-interface specifications, software processes and documentation—failed to support coordination if not properly managed. The mechanisms needed to be modified or augmented (allowing for the filling in of details, handling exceptions, coping with unforeseen events and recovering from errors) to allow the work to proceed. They also showed that the primary barriers to coordination breakdowns were the lack of unplanned contact, knowing whom to contact about what, cost of initiating a contact, ability to communicate effectively and lack of trust or willingness to communicate openly.

Kraut and Streeter (1995), in studying the coordination practices that influence the sharing of information and success of software development, identified the following coordination techniques: formal-impersonal procedures (projects documents and memos, project milestones and delivery schedules, modification request and error-tracking procedures, data dictionaries), formal-interpersonal procedures (status-review meetings, design-review meetings, code inspections), informal-interpersonal (group meetings and co-location of requirements and development staff, electronic communication such as e-mail and electronics bulletin boards, and interpersonal network). Their results showed the value of both informal and formal interpersonal communication for sharing information and achieving coordination in software development. Note though that this analysis focuses more the media for exchanging information rather than particular dependencies or coordination mechanisms that might be executed via these media. That is, once you have called a group meeting, what should you talk about?

Coordination in FLOSS Development

A few studies have examined the work practices and coordination modes adopted by FLOSS teams in more detail, which is the focus of this article (Iannacci, 2005; Scacchi, 2002; Weber, 2004). Cubranic (1999) observed that the main media used for coordination in FLOSS development teams were mailing lists. Such a low-tech approach is adopted to facilitate the participation of would-be contributors, who may not have access to or experience with more sophisticated technology. The geographical distribution of contributors and the variability in time of contributors precluded the use of other systems (e.g., systems that support synchronous communication or prescriptive coordination technology, such as workflow systems). Mailing lists supported low-level coordination needs. Also, Cubranic (1999) found no evidence of the use of higher-level coordination, such as group decision making, knowledge management, task scheduling and progress tracking. As they are the main coordination mechanisms, the volume

of information within mailing lists can be huge. Mailing lists are therefore often unique repositories of source information on design choices and evolution of the system. However, dealing with this volume of information in large open source software projects can require a large amount of manual and mental effort from developers, who have to rely on their memory to compensate for the lack of adequate tools and automation.

In a well-known case study of two important FLOSS projects, namely Apache and Mozilla, Mockus et al. (2002) distinguished explicit (e.g., interface specification processes, plans, etc.) and implicit coordination mechanisms adopted for software development. They argued that, because of its software structure, the Apache development team had primarily adopted implicit coordination mechanisms. The basic server was kept small. Core developers worked on what interested them and their opinion was fundamental when adding new functionality. The functionality beyond the basic server was added by means of various ancillary projects, developed by a larger community that interacted with Apache only through defined interfaces. Such interfaces coordinate the effort of the Apache developers: as they had to be designed based on what Apache provided, the effort of the Apache core group was limited. As a result, coordination relied on the knowledge of who had expertise in a given area and general communication on who is doing what and when. On the other hand, in the Mozilla project, because of the interdependence among modules, considerable effort is spent in coordination. In this case, more formal and explicit coordination mechanisms were adopted (e.g., module owners were appointed who had to approve all changes in their module).

Jensen & Scacchi (2005) modelled the software-release process in three projects, namely Mozilla, Apache and NetBeans. They identified tasks, their dependencies and the actors performing them. However, they did not analyze the coordination issues in depth and did not focus specifically on the bug-fixing process, which is the aim of this article. Rather, their final goal was to study the relationships among the three communities that form a Web Information Infrastructure.

Iannacci (2005) adopted an organizational perspective to study coordination processes within a single large-scale and well-known FLOSS development project, Linux. He identified three main (traditional) coordination mechanisms, namely standardization, loose coupling and partisan mutual adjustment. Standardization is a coordination mechanism to manage pooled dependencies emerging among different contributors. It implies the definition of well-defined procedures, such as in the case of patch submission or bug-fixing procedures. Loose coupling is used to manage sequential dependencies among the different subgroups of contributors. It is the coordination mechanisms used to, for example, incorporating new patches. Finally, partisan mutual adjustment is a mechanism used to manage what Iannacci (2005) called networked interdependencies, an extension of the reciprocal dependencies as proposed by Thompson (1967). Networked interdependencies are those emerging among contributors to specific part of the software. Partisan mutual adjustment produces a sort of structuring process so creating an informal (sub-)organization. However, these findings are based on a single exceptional case, the Linux project, making it unclear how much can be generalized to smaller projects. Indeed, most of the existing studies are of large and well-known projects and focused on the development process. To our knowledge, no studies have analyzed the bug-fixing process in depth within small FLOSS development teams.

A Coordination Theory Application: The Bug-Fixing Process

To ground our discussion of coordination theory, we will briefly introduce the bug-fixing process, which consists of the tasks needed to correct

software bugs. We decided to focus on the bug-fixing process for three reasons. First, bug fixing provides "a microcosm of coordination problems" (Crowston, 1997). Second, a quick response to bugs has been mentioned as a particular strength of the FLOSS process: as Raymond (1998) puts it, "given enough eyeballs, all bugs are shallow". Finally, it is a process that involves the entire developer community and thus poses particular coordination problems. While there have been several studies of FLOSS bug fixing, few have analyzed coordination issues within bug-fixing process by adopting a process view. For example, Sandusky et al. (2004) analyzed the bug-fixing process. They focus their attention on the identification of the relationships existing among bug reports, but they do not examine in details the process itself. In contrast to the prior work, our article provides empirical evidence about coordination practices within FLOSS teams. Specifically, we describe the way the work of bug fixing is coordinated in these teams, how these practices differ from those of conventional software development and thus suggest what might be learned from FLOSS and applied in other settings.

We base our description on the work of Crowston (1997), who described the bug-fixing process observed at a commercial software company. Such a process is below defined as traditional because 1) it is carried out within a traditional kind of organization (i.e., the boundary are well defined, the environment is not distributed, the organization structure is defined) and 2) refers to the production of commercial rather than FLOSS software. The process is started by a customer who finds a problem when using a software system. The problem is reported (sometimes automatically or by the customer) to the company's response center. In the attempt to solve the problem, personnel in the center look in a database of known bugs. If a match is found, the fix is returned to the customer; otherwise, after identifying the affected product, the bug report is forwarded to an engineer in the marketing center. The assigned engineer tries to reproduce the problem and identify the cause (possibly requesting additional information from the reporter to do so). If the bug is real, the bug report is forwarded to the manager responsible for the module affected by the bug. The manager then assigns the bug to the software engineer responsible for that module. The software engineering diagnoses the problem (if she finds that the problem is in a different module, the report is forwarded to the right engineer) and designs a fix. The proposed fix is shared with other engineers responsible for modules that might be affected. When the feedback from those engineers is positive, the proposed design is transformed into lines of code. If changes in other module are needed, the software engineer also asks the responsible engineers for changes. The proposed fix is then tested, the eventual changed modules are sent to the integration manager. After approving, the integration manager recompiles the system, tests the entire system and releases the new software in the form of a patch. To summarize then, in the traditional bug-fixing process, the following tasks have been identified (Crowston, 1997):

Report, Try to solve the problem, Search database for solution, Forward to the marketing manager, Try to solve the problem/Diagnose the problem, Forward to the Software Engineering Group, Assign the bug, Diagnose the problem, Design the fix, Verify affected modules and ask for approval, Write the code for the fix, Test it, Integrate changes, Recompile the module and link it to the system.

After describing the above process, Crowston (1997) went on to analyze the coordination mechanisms employed. A number of the tasks listed can be seen as coordination mechanisms. For example, the search for duplicate bugs as well as the numerous forward and verify tasks manage some dependency. Searching for duplicate outputs is the coordination mechanism to manage a dependency between two tasks that might have the same output. In this case, the tasks are to respond

to bug reports from customers. These tasks can be performed by diagnosing and repairing the bug, but if the solution to the bug report can be found in the database, then the effort taken to solve it a second time can be avoided. Thus, searching the database for a solution is a way to manage a potential dependency between the two bug-fixing tasks. Forwarding and verifying tasks are coordination mechanisms used to manage dependency between a task and the actor appropriate to perform that task. These steps are needed because many actors are involved in the process and each of them carry out a very specialized task, requiring additional work to find an appropriate person to perform each task.

RESEARCH METHODOLOGY

To address our research question, how are bug fixes coordinated in FLOSS projects, we carried out a multiple case study of different FLOSS projects, using the theoretical approach developed in the previous section. In this section, we discuss sample selection and data sources, data collection and data analysis, deferring a discussion of our findings to the following section.

Sample Section

In this sub-section we describe the basis for selecting projects for analysis. Projects to be studied were selected from those hosted on SourceForge, (http://sourceforge.net/), a Web-based system that currently supports the development of more than 100,000 FLOSS projects (although only a small proportion of these are actually active). We chose to examine projects from a single source to control for differences in available tools and project visibility. Because the process of manually reading, rereading, coding and recoding messages is extremely labor-intensive, we had to focus our attention on a small number of projects. We selected projects to study in-depth by employing a theoretical sampling strategy based on several practical and theoretical dimensions.

First, we chose projects for which data we need for our analysis are publicly available, meaning a large number of bug reports. (Not all projects use or allow public access to the bug-tracking system.) Second, we chose teams with more than 8 developers (i.e., those with write access to the source code control system), since smaller projects seemed less likely to experience significant coordination problems. The threshold of eight members was chosen based on our expectation that coordinating tasks within a team would become more complicated as the number of members increases. We assumed that each member of the team could manage 4 or 5 relationship, but with eight members, we expected some difficulty in coordination to arise. Only 140 projects of SourceForge met the first two requirements in 2002 when we drew our sample. Third, projects were chosen so as to provide some comparison in the target audience and addressed topic, as discussed below. Finally, because we wanted to link coordination practices to project effectiveness, we tried to select more and less effective development teams. To this aim we used the definitions of effectiveness proposed by Crowston et al. (2006a), who suggest that a project is effective if it is active, the resulting software is downloaded and used and the team continues in operation. We selected 4 FLOSS projects to satisfy the mentioned criteria. Specifically, from the 140 large active projects, we selected two desktop chat clients that are aimed at end users (KICQ and Gaim) and two projects aimed primarily at developers (DynAPI, an HTML library and phpMyAdmin, a web-based database administration tool). A brief description of the projects is reported in Table 1, including the project goal, age at the time of the study, volume of communication and team membership. A consequence of the requirement of a significant number of bug reports is that all four projects are relatively advanced, making them representative of mature FLOSS projects. Based on the definition

proposed by Crowston et al. (2006a), Kicq, Gaim and phpMyAdmin were chosen as examples of effective projects because they were active, the resulting software was being downloaded and the group had been active for a while. DynAPI was chosen as an example of a less effective project because the number of downloads and programming activity had rapidly decreased in the months leading up to the study.

Table 1. Four examined projects

	KICQ	**DynAPI**	**Gaim**	**phpMyAdmin**
Goal	ICQ client for the KDE project (a chat client)	Dynamic HTML library	Multi-platform AIM client (a chat client)	Web-based database administration
Registration date	1999-11-19	2000-05-15	1999-11-13	2001-03-18
Development Status	4 Beta, 5 Production Stable	5 Production Stable	5 Production Stable	5 Production Stable
License	GPL	LGPL, GPL	GPL	GPL
Intended Audience	Developers, End Users/Desktop	Developers	Advanced End Users, Developers, End Users/Desktop	Developers, End Users/Desktop, System Administrators
Topic	ICQ, K Desktop Environment (KDE)	Dynamic Content	AOL Instant Messenger, ICQ, Internet Relay Chat, MSN Messenger	Front-Ends, Dynamic Content, Systems Administration
Open bugs/ Total # of bugs	26 /88	45/220	269 /1499	29 /639
Open Support Requests/Total # of requests	12/18		20/107	3/125
Open Patches/ Total # of Patches	1/8	14/144	75/556	7/131
Open Features requests/Total # of requests	9/9	5/12	214/447	214/447
Mailing lists	813 messages in 3 mailing lists	9595 in 5 mailing lists	304 in 1 mailing list (developers)	5456 in 5 mailing lists
# of team members	9	11	9	9
Team member roles (# in role)	Admin/project manager (2); packager (1); developers (3); advisor/ mentor/ consultant(1); not specified (2)	Admin/project manager (1); developers (4); admin (3); not specified (3)	Project manager (1); admin/ developer (1); support manager (1); web designer (1); developers (3) not specified (2)	Project manager/ admin (1); admin/ developer (2); developers (6)

Data Collection

In this sub-section we describe how data were selected and collected. As mentioned above, all of these projects are hosted on SourceForge, making certain kinds of data about them easily accessible for analysis. However, analysis of these data poses some ethical concerns that we had to address in gaining human subjects approval for our study. On the one hand, the interactions recorded are all public and developers have no expectations of privacy for their statements (indeed, the expectation is the opposite, that their comments will be widely broadcast). Consent is generally not required for studies of public behaviour. On the other hand, the data were not made available for research purposes but rather to support the work of the teams. We have gone ahead with our research after concluding that our analysis does not pose any likelihood of additional harm to the poster above the availability of the post to the group and in the archive available on the Internet.

We collected several kinds of data about each of the cases. First, we obtained data indicative of the effectiveness of each project, such as its level of activity, number of downloads and development status. Unfortunately, no documentation on the organization structure, task assignment procedures and coordination practices adopted was available on the projects' web sites (further supporting the position that these teams do not employ formal coordination methods). To get at the bug-fixing process, we considered alternative sources of data. Interviewing the developers might have provided information about their perceptions of the process, but would have required finding their identities, which was considered problematic given privacy concerns. Furthermore, reliance on self-reported data raises concerns about reliability of the data, the response rate and the likelihood that different developers would have different perceptions. While these issues are quite interesting to study (e.g., to understand how a team develops shared mental models of a project, for example, Crowston & Kammerer, 1998), they seemed like distractions from our main research question. Because of these concerns, we elected to use objective data about the bug-fixing process. Hence, the main source of data about the bug-fixing process was obtained from the archives of the bug tracking system, which is the tool used to support the bug-fixing process (Herbsleb et al., 2001, p. 13). These data are particularly useful because they are unobtrusive measures of the team's behaviors (Webb & Weick, 1979) and thus provide an objective description of the work that is actually undertaken, rather than perceptions of the work.

In the bug tracking system, each bug has a request ID, a summary (what the bug is about),

Figure 1. Example bug report and followup messages

[206585] crash with icq chat

Email: Monitor (?)
Date: 2000-05-28 12:56
Priority: 5
Submitted By: hub (khub)
Assigned To: Bill Soudan (bills)
Category: system
Status: Closed
Summary: crash with icq chat
each time I try an icq chat session the whole program closes itself immediately

Followups:

Message

Date: 2000-07-29 08:56
Sender: denis
Ok, since khub reported it works for him, I am closing this bug.

To robvnl I repeat:
"please try latest sources from CVS"...

Date: 2000-06-29 13:02
Sender: robvnl
Module Name: kicq
Latest release is 19991212.
is written here, or did I probably install a Beta version 19991212?
It would be great to can chat again!

Date: 2000-06-18 01:10
Sender: khub
I've try the latest version, it seems to work perfectly
That's marvellous...

Date: 2000-06-17 06:50
Sender: denis
Ok, lets try it one more time - WHAT VERSION OF KICQ do you use?
There was dramatic improvements in chat code since 19991212 beta,
so please try latest sources from CVS and report your comments
back.

Date: 2000-06-08 12:32
Sender: robvnl
Hi, I have the exact same problem. It doesn't make difference
wether I initiate or the other party initiates the chat. I use
RedHat 6.2 and compiled kicq with the export QTDIR=/usr/lib/qt-1.45
(the previous libs) because it needed it. Also it doesn't make
difference to run with the old or new QTDIR set. Sometimes the
chat request results in an user ABORTed at the other side and
I get USER HAS LEFT the chat. In case kicq realy crashes I have
a core dump.

Date: 2000-05-29 05:03
Sender: denis
What version do you try?

a category (the kind of bug, e.g., system, interface), the name of the team member (or user) who submitted it, and the name of the team member it was assigned to. An example bug report in shown in Figure 1 (the example is fictitious). As well, individuals can post messages regarding the bug, such as further symptoms, requests for more information, etc. From this system, we extracted data about who submitted the bugs, who fixed them and the sequence of messages involved in the fix. By examining the name of the message senders, we can identify the project and community members who are involved in the bug-fixing process. Demographic information for the projects and developers and data from the bug tracking system were collected in the period 17–24 November 2002. We examined 31 closed bugs for Kicq, 95 closed bugs for DynAPI, 51 bugs for Gaim and 51 for PhPMyAdmin. The detailed text of the bug reports is not reported because of space restriction but is available on request.

Data Analysis

In this section we present our data analysis approach. For each of the bug reports, we carefully examined the text of the exchanged messages to identify the task carried out by each sender. We first applied the framework developed by Checkland & Scholes (1990), who suggested identifying the owners, customers and environment of the process, the actors who perform it, the transformation of inputs into outputs, the environment and the worldview that makes the process meaningful. We then followed the method described by Crowston & Osborn (2003), who suggested expanding the analysis of the transformation by identifying in more detail the activities carried out in the transformation. We identified the activities by inductively coding the text of the messages in the bug tracking systems of the four projects. We started by developing a coding scheme based on prior work on bug fixing (Crowston, 1997), which provided a template of expected activities needed for task assignment (those listed above). The coding system was then evolved through examination of the applicability of codes to particular examples. For example the message:

I've been getting this same error every FIRST time I load the dynapi in NS (win32). After reloading, it will work… loading/init problem?

represents a report submitted by another user (someone other than the person who initially identified and submitted the bug). This message was coded as "report similar problems". Table 2 shows the list of task types that were developed for the coding. The lowest level elementary task types were successively grouped into 6 main types of tasks, namely *Submit, Assign, Analyze, Fix, Test & Post,* and *Close*. A complete example of the coded version of a bug report (the one from Figure 1) is shown in Figure 2.

Once we had identified the process tasks, we studied in depth the bug-fixing process as carried out in the four cases. Specifically, we compared the sequence of tasks across different bugs to assess which sequences were most common and the role of coordination mechanisms in these sequences. We also examined which actors performed which tasks as well as looked for ways to more succinctly present the pattern of tasks, for example, by presenting them as Markov processes. Because of the shortness and relative simplicity of our task sequences, we could exactly match task sequences, rather than having to statistically assess the closeness of matches to be able to form clusters (Sabherwal & Robey, 1995). Therefore, we were able to analyze the sequences by simple tabulation and counting, though more sophisticated techniques would be useful for larger scale data analysis. In the next Section we present the results of our analysis.

Table 2. Coded tasks in the bug-fixing process

1.0.0 **Submit (S)**
1.1.0 Submit bug (code errors)
1.1.1 Submit symptoms
1.1.2 Provide code back trace (BT)
1.2.0 Submit problems
1.2.1 Submit incompatibility problems (NC)
2.0.0. **Assign (As)**
2.1.0 Bug self-assignment (A*)
2.2.0 Bug assignment (A)
3.0.0 **Analyze (An)**
3.1.0 Contribute to bug identification
3.1.1Report similar problems (R)
3.1.2 Share opinions about the bug (T)
3.2.0 Verify impossibility to fix the bug
3.2.1 Verify bug already fixed (AF)
3.2.2.Verify bug irreproducibility (NR)
3.2.3 Verify need for a not yet supported function (NS)
3.2.4 Verify identified bug as intentionally introduced (NCP)
3.3.0 Ask for more details
3.3.1 Ask for Code version/command line (V)
3.3.2 Ask for code back trace/examples (RBT/E)
3.4.0 Identify bug causes (G)
3.4.1 Identify and explain error (EE)
3.4.2 Identify and explain bug causes different from code (PNC)
4.0.0 Fix (F)
4.1.0 Propose temporary solutions (AC)
4.2.0 Provide problem solution (SP)
4.3.0 Provide debugging code (F)
5.0.0 **Test & Post (TP)**
5.1.0 Test/approve bug solution
5.1.1 Verify application correctness (W)
5.2.0 Post patches (PP)
5.3.0 Identify further problems with proposed patch (FNW)
6.0.0 **Close**
6.1.0 Close fixed bug/problem
6.2.0 Closed not fixed bug/problems
6.2.1 Close irreproducible bug (CNR) and close it
6.2.2 Close bug that asks for not yet supported function (CNS)
6.2.3 Close bug identified as intentionally introduced (CNCP)

Figure 2. Coded version of bug report in Figure 1

Bug ID	**Summary**	**Assigned to**	**Submitter**
0000000	crash with *alfa* chat	gills	kkhub

Task	**Person**	**Comments**
(S)	kkhub	
(V)	cenis	asks what version kkhub is running
(R)	cobvnl	reports the same problem as kkhub. submits information about the operating systems and the libraries
(V)	cenis	asks again what version both users are running
(W)	kkhub	reports the most recent version of cicq works
(TP&C)	cobvnl	reports version information and close the bug
(C)		bug closed

FINDINGS

In this section we present the findings from our analysis of the bug-fixing process in the four projects and the coordination mechanisms employed. Data about the percentage of submitted, assigned and fixed bugs both by team members and individuals external to the team for each project are reported in Table 3. Table 4 summarizes our findings regarding the nature of the bugs fixing process in the four projects.

We now present our overall analysis of the bug-fixing process. Each instance of a bug-fixing process starts (by definition) with a bug submission (S) and finishes with bug closing (C). Submitters may submit problems/symptoms associated with bugs (Ss), incompatibility problems (NC) or/and also provide information about code back trace (BT). After submission, the team's project managers or administrators may assign the bug to someone to be fixed ((A); (A*) if they self-assign the bug). Other members of the community may report similar problems they encountered (R), discuss bug causes (T), identify bug causes (G) and/or verify the impossibility of fixing the bug. Participants often ask for more information to better understand the bug's causes (An). In most cases, but not always, after some discussion, a team member spontaneously decides to fix (F) the bug. Bug fixing may be followed by a test and the submission of a patch (TP). Testing is a coordination mechanism that manages usability between producing and using a patch, by ensuring that the patch is usable. However, as later explained, in the examined projects this type of activity is not often found. The bug is then closed (C). Bugs may also be closed because they cannot be fixed, for example, if they are not reproducible (CNR), involve functions not supported yet (CNS) and/or are intentionally introduced to add new functionality in the future (CNCP). Notice that the closing activity is usually attributed to a particular user.

For our analysis, we consider *Submission*, *Analysis*, *Fix* and *Close* to be operative activities, while *Assignment*, *Test* and *Posting* are coordination mechanisms. As already discussed, *Assignment* is the coordination mechanisms used

to manage the dependency between a task and the actor appropriate to perform it. *Posting* is the mechanisms used to manage the dependency between a task and its customers (it makes the fix available to the persons that need it).

The tasks identified above are linked by sequential dependencies as shown in Figure 3. These dependencies were identified by considering the logical connection between tasks based on the flow of resources. For example, a patch

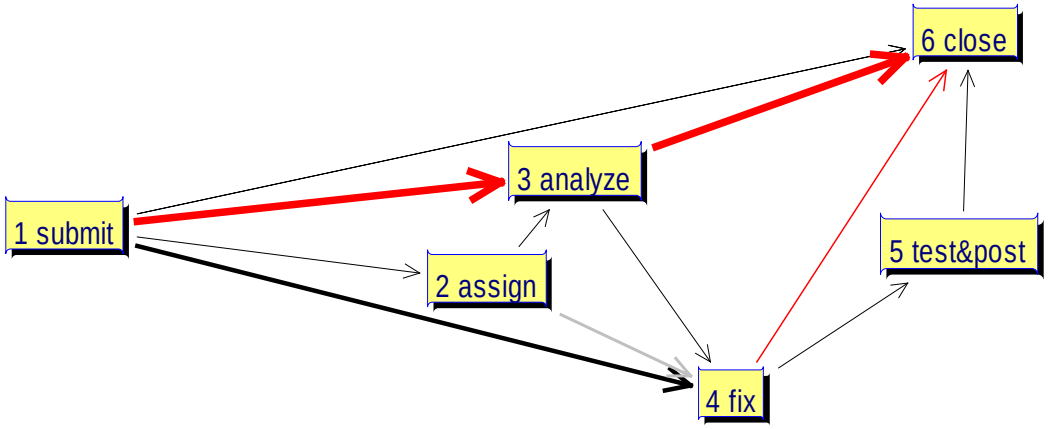
Table 3. The bug-fixing process: Main results

	Kicq	**DynAPI**	**Gaim**	**phpMyAdmin**
Bugs submitted by team members	9.7%	21%	0%	21.6%
Bugs submitted by members external to the team	90.3%	78.9%	100%	78.4%
Bug assigned/self-assigned of which:	9.7%	0%	2%	1%
Assigned to team members	0%	-	100%	100%
Self assigned	66%			0%
Assigned to members external to the team	33%	-	-	0%
Bug fixed	51,6%	42,1%	51%	80%
Fixed by team members	81,3%	50%	84%	90,2%
Bug fixed by members external to the team	18,7%	50%	16%	9.8%

Table 4. Observed characteristics of the bug-fixing processes in the four projects

	Kicq	**DynAPI**	**Gaim**	**phpMyAdmin**
Min task sequence	3	2	2	2
Max task sequence	8	12	9	13
Uncommon tasks (count)	Bug assignment (3)	Bug assignment (0)	Bug assignment (0)	Bug assignment (1)
Community members	18	53	23	20
Team members' participation	2 of 9	6 of 11	3 of 9	4 of 10
Most active team members Role/ name	Project mgr: denis; Developer: davidvh	Admin: rainwater; Ext member: dcpascal	Admin-developer: warmenhoven; Developer: robflynn	Admin-developer: loic1; Admin-developer lem9.
Max posting by single community member	2	6	4	3
Not fixable bug closed	8	5	5	-

Figure 3. Task dependencies in the bug-fixing process



can not be tested before it is created. Because the dependencies can be satisfied in different orders, different sequences of the activities are possible. The tasks and their sequence change from bug to bug. Figure 3 shows the most frequent sequences observed, as identified by tabulating and counting the sequences.

Table 5 shows the portion of processes that follow each possible paths, based on the collected ways the bug-fixing process is observed to be performed within the FLOSS teams. For example, row 1 of Table 5 is read as follows. In the Dynapi project, submission always occurs as the first task (as it does for all of the groups, by definition), while the second task is S in 26% of cases, An in 39% of cases, F in 19% of cases, TP in 1% of cases and C in 15% of cases, and so on.

In Table 6, we describe the occurrences per task for the four projects and the average number of tasks to fix bugs. A χ^2 test shows a significant difference in the distribution of task types across projects ($p<0.001$). On all projects, *submit* is the task that always appears first, while *analyze* is the most common second task and *fix*, third. The first three most frequent task sequences are reported in Table 7. As noted above, given the limited number of examined sequences, the sequences were manually identified. Finally, in Table 8 we show which tasks are carried out by which roles. Please notice that differences in percentage shown in Table 3 and Table 8 are due to the fact that results reported in Table 8 are calculated based on the total number of tasks carried out per bug. For example, in Table 3 the considered submissions are those carried out only as first task. In Table 8 all submissions tasks (i.e., also those carried out as second, third etc. task) are considered. As reported in Table 2, submissions tasks can be more than one per bug because submissions can occur also in the form of a *submit* sub-task. The same stands for the fixing tasks. In Table 3 only the final fixing tasks are considered.

A detailed description of the process as performed in the four cases is provided below considering both the sequence of tasks and the participation in the bug-fixing process.

Table 5. Portion of processes for each possible path

i	task i-1	task i	Kicq	Dynapi	Gaim	PhPmyadmin
2	S	S	42%	26%	4%	2%
		As	6%	-	2%	2%
		An	39%	39%	61%	41%
		F	13%	19%	24%	45%
		TP	-	1%	2%	8%
		C	-	15%	8%	2%
3	S	An	38%	36%	50%	100%
		F	62%	40%	50%	-
		TP	-	8%	-	-
		C	-	16%	-	-
	As	An	-	-		100%
		F	50%	-	100%	-
		TP	50%	-	-	-
	An	S	8%	-	-	5%
		An	25%	41%	58%	52%
		F	8%	11%	3%	29%
		TP	-	-	3%	-
		C	58%	49%	35%	14%
	F	An	-	11%	-	13%
		F	50%	22%	8%	4%
		TP	-	6%	-	4%
		C	50%	61%	92%	78%
	TP	An	-	-	-	50%
		F	-	100%	100%	-%
		TP	-	-	-	-50%
		C	-	-	-	-
	C	An	-	7%	-	-
		C	-	93%	-	-
4	S	S	-	-	-	-
		An	100%	-	-	-
		F	-	-	-	100%
		TP	-	-	-	-
		C	-	-	-	-
	An	S	-	4%	5%	-
		An	13%	48%	53%	50%
		F	25%	11%	21%	11%
		TP	-	4%	-	6%
		C	63%	33%	21%	33%
	F	S	-	-	-	-
		As	8%	-	-	-
		An		11%	20%	
		F	33%	16%	-	14%

Table 5. continued

i	task i-1	task i	Kicq	Dynapi	Gaim	PhPmyadmin
		TP	-	5%	-	29%
		C	58%	68%	80%	57%
	TP	S	-	-	-	-
		An	-	-	-	-
		F	-	33%	-	33%
		TP	-	-	-	33%
		C	-	67%	100%	33%
	C	C	-	-	100%	-
5	S	AN	-	-	100%	-
		F	-	-	-	-
		TP	-	100%	-	-
	As	F	100%	-	-	-
	An	S	-	-	-	-
		An	50%	27%	73%	67%
		F	-	13%	18%	11%
		TP	-	-	-	11%
		C	50%	60%	9%	11%
	F	An	17%	14%	-	20%
		F	--	-	25%	-
		TP	-	-	25%	-
		C	83%	86%	50%	80%
	TP	An	-	-	-	-
		F	-	-	-	50%
		TP	-	100%	-	-
		C	-	-	-	50%
6	An	S	-		11%	-
		As	50%	-	-	14%
		An	-	20%	22%	43%
		F	-	-	11%	29%
		TP	-	20%	-	-
		C	50%	60%	56%	14%
	F	S	-	-	-	-
		An	-	-	-	-
		F	-	-	-	-
		TP	-	-	-	33%
		C	100%	100%	-	67%
	TP	An	-	-	-	-
		F	-	100%	-	-
		TP	-	-	-	-
		C	-	-	-	100%
7	S	AN	-	-	50%	-
		©	-	-	50%	-

Table 5. continued

i	task i-1	task i	Kicq	Dynapi	Gaim	PhPmyadmin
	As	F	100%	-	-	100%
	An	S	-	-	-	33%
		An	-			33%
		F	-	100%	100%	-
		TP	-	-	-	-
		C	-	-	-	33%
	F	An	-	100%	-	-
		F	-	-	-	-
		TP	-	-	-	-
		C	-	-	100%	100%
	TP	F	-	100%	-	100%
8	S	An	-	-	-	100%
		F	-	-	-	-
	An	An	-	100%	-	-
		F	-		100%	100%
	F	An	-	50%	-	-
		TP	-	-	-	50%
		C	100%	50%	100%	50%
9	An	An	-	50%	-	100%
		C	-	50%	-	-
	F	AN	-	-	-	100%
		C	-	-	100%	-
	TP	TP	-	-	-	100%
10	An	An	-	100%	-	50%
		F	-	-	-	50%
	TP		-	-	-	100%
11	An	An	-	100%		50%
		F	-	-	-	50%
	F	C	-	-	-	100%
12	An	An	-	-	-	100%
		C	-	100%	-	-
	F	C	-	-	-	100%
13	An	C	-	100%	-	100%

Table 6. Task occurrences and average number of tasks per projects

Task / Project (bugs)	(S)	(Ag)	(An)	(F)	(TP)	(C)	Avr. tasks per bug
KICQ (31)	44	4	24	23	0	31	4.4
Dynapi (95)	121	0	94	54	9	95	3.8
Gaim (51)	71	1	77	28	4	51	4.2
Phpmyadmin (51)	54	2	66	45	15	51	4.6

Table 7. Most frequent task sequences

	First task	Second task	Third task	Fourth Task	Occurrences
Kicq	S	An	C	-	13
	S	F	C	-	11
	S	An	F	C	2
DynAPI	S	An	C	-	34
	S	F	C	-	24
	S	C	-	-	17
Gaim	S	An	C	-	21
	S	F	C	-	13
	S	An	F	C	6
phpMyAdmin	S	F	C	-	19
	S	An	C	-	8
	S	An	F	C	7
All projects	S	An	C	-	76
	S	F	C	-	67
	S	C	-	-	22

Kicq

The minimal sequence is composed of three tasks, the longest by eight. Bug fixing is usually the second task in the sequence, meaning that it is most common for bugs to be fixed immediately after they are submitted, which is different from the overall picture in which analysis was most common. *Bug assignment* is a quite rare task, as only three bugs are formally assigned. Eight bugs were closed because they were considered to be not fixable.

There are 18 identified users, but many (anonymous) users submitted bugs and contributed to analysis and fixing. Team members are not very active in bug fixing, except for one of the two project managers (denis), who is involved in all the tasks and, in particular, in bug analysis and fixing. Out of 23 fixed bugs, 16 are fixed by denis. Apart from a developer (davidvh), the other project members seem not take part in the bug-fixing process at all. However, it is noteworthy that the bug tracking system register three bugs as submitted and assigned to the administrator (bill), although he does not otherwise take part in the process. Most of the community members have posted just one bug, and only two of them posted 2 bugs each.

Dynapi

The minimal sequence is composed of two tasks, the longest by 12. Again, *bug assignment* is not explicitly carried out; apparently community or team members decide autonomously to take part to the bug-fixing process. However, the system reports that six bugs (out of 95) are assigned to an administrator and the rest to a member external to the team. Five bugs are closed because they are said to be not fixable. Bug fixing is usually the second or the third task in the sequence.

Table 8. Tasks carried out by different roles

task	**ROLES/PROJECT Kick**				
	devel	pm			% of total tasks
S		4			9%
As		4			100%
An		18			75%
F	1	15			70%
TP					
	Dynapi				
	devel	admin	admin/develop	no role	% of total tasks
S	9	6	1	10	21%
As					
An		27		3	32%
F		18	1	2	35%
TP		2	1		33%
	Gaim				
	admin/develop	develop	supp. mang.		% of total tasks
S					0%
As		1			100%
An	33	11	1		58%
F	17	6			82%
TP					100%
	Phpmyadmin				
	admin/develop	pm			% of total tasks
S	11	1			22%
As	2				100%
An	49				74%
F	40				89%
TP	10				93%

Team members are not very active except for an administrator (rainwater), who is involved in all the tasks and, in particular, in bug *analysis* and *fixing.* The other five team members (two without a specific role, one administrator/developer, one developer and one administrator) are mostly involved in bug *fixing*. The community members involved in the process are 47 persons plus some anonymous posts. Most of them submitted just one bug, but some submitted more (e.g., one submitted six bugs). Community members are mostly involved in bug *submission* but some also carry out

other tasks. In particular, one of them (dcpascal) is very active in all the process tasks. Out of 57 fixed bugs, 20 are fixed by a team member (the project manager).

Gaim

The minimal sequence is composed of two tasks, the longest by nine. *Bug assignment* is not explicitly carried out, as community or team members decide autonomously to take part to the bug-fixing process. However, the system reports that 24 bugs (out of 51) are assigned to an administrator (and the rest to member external to the team). Five bugs are directly closed because they are said to be not fixable.

Team members are not very active in bug fixing except for the administer/developer (warmenhoven) and a developer (robflynn), who are involved in many tasks and, in particular, in bug analysis and fixing. Apart from them, just another member of the project team, a developer (lschiere), is also involved in the bug fixing. The community members involved in the process are 21 persons plus some anonymous users. Most of them posted just one bug (2 of them posted five bugs, one 4 bugs). Some of them are also involved in bug analysis and fixing. Out of 29 fixed bugs, 23 are fixed by a team member (the project manager).

Phpmyadmin

The minimal sequence is composed of two tasks, the longest by thirteen. *Bug assignment* is a quite rare task, as only one bug is formally assigned. The assignment is carried out by an administrator/developer (lem9) and directed to a team member (loic1). However, the system reports that all 51 are assigned, of which 40 to team members. Bug fixing is usually the second or the third task.

Team members are not very active in the process, except for two administer/developers (loic1 and lem9), who are involved in all the tasks and, in particular, in bug analysis and fixing (but also submission). Apart from them, two team members take part to the process, a project manager/adminster (swix) and a developer (robbat2), that are involved (not heavily) in bug submission and analysis. The community is composed of 16 members plus some anonymous users. Most of them have just posted one bug (two of them posted 3 bugs), but some are also involved in bug analysis and fixing. Out of 49 fixed bugs, 44 are fixed by team member (administrator/developers).

DISCUSSION

In this section, we discuss the implications of our findings for understanding the coordination of bug fixing in FLOSS teams. Our findings provide some interesting insights on the bug-fixing process for FLOSS development in these teams. First, process sequences are on average quite short (four tasks) and they seem to be quite similar: submit, (analyze), fix and close. As shown in Table 3, formal task assignments are quite uncommon: only few bugs are formally assigned. Coordination seems rather to spontaneously emerge. From bug description and initial analysis, those who have the competencies autonomously decide to fix the bug and simply go ahead and do so. That activity is facilitated by the supplied bug report and analysis, which is often undertaken by several contributors. Apart from the procedure to submit bugs (we analyzed only bugs submitted through the bug tracking system), we do not observe any other formal process: roles are not predefined, delivery dates are not assigned nor are formal-interpersonal, formal-impersonal or informal-interpersonal procedures adopted. The lack of assignment is one of main aspects differentiating the process as it occurs in FLOSS development team from the traditional commercial bug-fixing process described above.

Testing is also quite an uncommon task in the data. Most of the proposed fixes are directly

posted, though presumably after personal testing that is not documented. If no one describes the emergence of new problems with these fixes, they are automatically posted and the relevant bug closed without a formal test process. It is important also to note that many of the posted problems do not represent real bugs (i.e., they have been already fixed, are not reproducible, have been intentionally produced, are associated to functions not yet supported or are associated to related programs), so they are directly closed with that explanation.

Another striking finding is that the bug-fixing process is apparently carried out without any explicit discussion about where knowledge is located in the team, contrary to the findings of Faraj and Sproull (2000), who stress the importance of expertise coordination for team effectiveness (they distinguish expertise coordination from what they call administrative coordination, which is the focus of this article). They define expertise coordination as the management of knowledge and skill dependencies. To manage knowledge it is necessary to know where it is located within development team, where it is needed and how to access it. However, in our observations, the knowledge needs seem to emerge by "(informal and asynchronous) electronic meetings".

The bug tracking system represents a sort of organizational memory, storing bug reports and solutions found to submitted problems (which not always are real bugs). However, as discussed in Cubranic (1999), the large number of emails stored makes it difficult for contributors to easily identify the solutions to their own problems, so making different users repeat the same (already fixed or addressed) submission more times. In those cases (i.e., for bugs closed without being fixed or the attended patches posted), it is usually the team members that act as "memory".

A further difference is that in these projects, the process is performed by few team members (usually not more that two or three) working with a member of the larger community. Team members (usually project managers, administrators or developers) are most involved in bug fixing, testing and posting. Surprisingly, only a few members of the team are involved in the process. The other participants are active users who submit bugs or contribute to their analysis. We also noted striking differences in the level of contribution to the process. The most active users in the projects carried out most of the tasks while most others contributed only once or twice. Most community members submit only one bug; only two or three members of the involved community are involved in fixing tasks and can be referred to as co-developers. As expected, the most widely dispersed type of action was submitting a bug, while diagnosis and bug-fixing activities were concentrated among a few individuals.

As we have few members of the team and few members of the community (co-developers) mostly involved in bug fixing and many users/members of the community (active users) mostly involved in bug submission, the organizational models proposed in the literature (Cox, 1998) seem to be valid for the bug-fixing process. It would be interesting to further investigate if those, among the active users also involved in bug fixing also contribute to software coding, for example, by analysis of contributions of source code independent of bug fixes.

As an apparently less effective project, we expected to find that DynAPI had a smaller active user base than the other projects. However, as noted above, our data shows the opposite. However, our estimation of the effectiveness of the projects is based on activity levels. It appears that DynAPI somehow does not benefit from its larger community in increased activity. One striking difference is the proportion of bugs fixed by the team members, shown in Table 3, which is much lower in DynAPI than in the other projects. This finding suggests that the contribution of core members may be particularly important in the effectiveness of the team. The case studies presented here are not sufficient to test this

hypothesis, so it is one that should be followed up in future studies.

CONCLUSION

In this article, we investigated the coordination practices adopted within four FLOSS development teams. In particular, we analyzed the bug-fixing process, which is considered central to the effectiveness of the FLOSS process. The article provided some interesting results. The task sequences we observed were mostly sequential and composed of few steps, namely *submit, fix and close*. Second, our data supports the observation that FLOSS processes seem to lack traditional coordination mechanisms such as task assignment. Third, effort is not equally distributed among process actors. A few contribute heavily to all tasks, while the majority just submit one or two bugs. As a result, the organization structure reflected in the process resembles the one proposed in the literature for the FLOSS development process. Few actors (core developers), usually team project managers or administrators, are mostly involved in bug fixing. Most of the involved actors are active users instead of developers, who just submit bug reports. In between are few actors, external to the team, who submit bugs and contribute to fixing them. Finally, while we did not find obvious associations between coordination practices and project effectiveness, we did notice a link to participation: our least effective team also had the lowest level of participation from core developers, suggesting their importance, even given the more widely distributed participation possible.

The article contributes to fill a gap in the literature by providing a picture of the coordination practices adopted within FLOSS development team. Besides, the article proposes an innovative research methodology (for the analysis of coordination practices of FLOSS development teams) based on the collection of process data by electronic archives, the codification of message texts, and the analysis of codified information supported by the coordination theory.

Based on the analysis of the tasks carried out and the attendant coordination mechanisms, we argue that the bazaar metaphor proposed by (Raymond, 1998) to describe the FLOSS organization structure is still valid for the bug-fixing process. As in a bazaar, the actors involved in the process autonomously decide the schedule and contribution modes for bug fixing, making a central coordination actor superfluous.

As with all research, the current article has some limitations that limit the scope of our current conclusions and suggests directions for further research. First, although the selected projects are quite different in terms of target audience and topic, other characteristics (not examined because they are not explicitly present on the project web sites) could be shared among projects so affecting the obtained results. In the future, we would like to deepen our knowledge about the coordination practices adopted by the four projects by directly interviewing some of the involved actors. Second, due to the limited number of examined bugs, the process sequences have been manually examined. In the future, we intend to enlarge the number of examined bugs and adopt automatic techniques (e.g., the optimal matching technique) to analyze and classify the task sequences. In particular, we plan to further explore the hypothesis about the importance of core group members by examining a larger number of projects (e.g., to examine the change in the population over time). Finally, in the article we only examined administrative coordination. In the future, we intend to examine also expertise coordination in more detail. A particular interesting consideration here is the development of shared mental models that might support the coordination of the teams' processes.

REFERENCES

Ahuja, M. K., Carley, K., & Galletta, D. F. (1997). *Individual performance in distributed design groups: An empirical study.* Paper presented at the SIGCPR Conference, San Francisco.

Alho, K., & Sulonen, R. (1998). *Supporting virtual software projects on the Web.* Paper presented at the Workshop on Coordinating Distributed Software Development Projects, 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98).

Anthes, G. H. (2000, June 26). Software Development goes Global. *Computerworld Magazine.*

Bandow, D. (1997). Geographically distributed work groups and IT: A case study of working relationships and IS professionals. In *Proceedings of the SIGCPR Conference* (pp. 87–92).

Bélanger, F. (1998). Telecommuters and Work Groups: A Communication Network Analysis. In *Proceedings of the International Conference on Information Systems (ICIS)* (pp. 365–369). Helsinki, Finland.

Bessen, J. (2002). *Open Source Software: Free Provision of Complex Public Goods*: Research on Innovation.

Bezroukov, N. (1999a). A second look at the Cathedral and the Bazaar. *First Monday, 4*(12).

Bezroukov, N. (1999b). Open source software development as a special type of academic research (critique of vulgar raymondism). *First Monday, 4*(10).

Boulding, K. E. (1956). General systems theory—The skeleton of a science. *Management Science, 2*(April), 197–208.

Britton, L. C., Wright, M., & Ball, D. F. (2000). The use of co-ordination theory to improve service quality in executive search. *Service Industries Journal, 20*(4), 85–102.

Brooks, F. P., Jr. (1975). *The Mythical Man-month: Essays on Software Engineering.* Reading, MA: Addison-Wesley.

Butler, B., Sproull, L., Kiesler, S., & Kraut, R. (2002). Community effort in online groups: Who does the work and why? In S. Weisband & L. Atwater (Eds.), *Leadership at a Distance*. Mahwah, NJ: Lawrence Erlbaum.

Carmel, E. (1999). *Global Software Teams.* Upper Saddle River, NJ: Prentice-Hall.

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*(March/April), 22–29.

Checkland, P. B., & Scholes, J. (1990). *Soft Systems Methodology in Action.* Chichester: Wiley.

Conway, M. E. (1968). How do committees invent. *Datamation, 14*(4), 28–31.

Cox, A. (1998). Cathedrals, Bazaars and the Town Council. Retrieved 22 March, 2004, from http://slashdot.org/features/98/10/13/1423253.shtml

Crowston, K. (1997). A coordination theory approach to organizational process design. *Organization Science, 8*(2), 157–175.

Crowston, K., & Howison, J. (2006). Hierarchy and centralization in free and open source software team communications. *Knowledge, Technology & Policy, 18*(4), 65–85.

Crowston, K., Howison, J., & Annabi, H. (2006a). Information systems success in Free and Open Source Software development: Theory and measures. *Software Process—Improvement and Practice, 11*(2), 123–148.

Crowston, K., & Kammerer, E. (1998). Coordination and collective mind in software require-

ments development. *IBM Systems Journal, 37*(2), 227–245.

Crowston, K., & Osborn, C. S. (2003). A coordination theory approach to process description and redesign. In T. W. Malone, K. Crowston & G. Herman (Eds.), *Organizing Business Knowledge: The MIT Process Handbook*. Cambridge, MA: MIT Press.

Crowston K., Scozzi B., (2003). Open Source Software projects as virtual organizations: competency rallying for software development. *IEE Proceedings Software,* 149(1), 3-17.

Crowston, K., Wei, K., Li, Q., Eseryel, U. Y., & Howison, J. (2005). *Coordination of Free/Libre Open Source Software development*. Paper presented at the International Conference on Information Systems (ICIS 2005), Las Vegas, NV, USA.

Crowston, K., Wei, K., Li, Q., & Howison, J. (2006b). *Core and periphery in Free/Libre and Open Source software team communications.* Paper presented at the Hawai'i International Conference on System System (HICSS-39), Kaua'i, Hawai'i.

Cubranic, D. (1999). *Open-source software development.* Paper presented at the 2nd Workshop on Software Engineering over the Internet, Los Angeles.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268–1287.

Curtis, B., Walz, D., & Elam, J. J. (1990). Studying the process of software design teams. In *Proceedings of the 5th International Software Process Workshop On Experience With Software Process Models* (pp. 52–53). Kennebunkport, Maine, United States.

Cutosksy, M. R., Tenenbaum, J. M., & Glicksman, J. (1996). Madefast: Collaborative engineering over the Internet. *Communications of the ACM, 39*(9), 78–87.

de Souza, P. S. (1993). *Asynchronous Organizations for Multi-Algorithm Problems.* Unpublished Doctoral Thesis, Carnegie-Mellon University.

DeSanctis, G., & Jackson, B. M. (1994). Coordination of information technology management: Team-based structures and computer-based communication systems. *Journal of Management Information Systems, 10*(4), 85.

Di Bona, C., Ockman, S., & Stone, M. (Eds.). (1999). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, CA: O'Reilly & Associates.

Drucker, P. (1988). The coming of the new organization. *Harvard Business Review,* 3-15.

Faraj, S., & Sproull, L. (2000). Coordinating Expertise in Software Development Teams. *Management Science, 46*(12), 1554–1568.

Finholt, T., Sproull, L., & Kiesler, S. (1990). Communication and Performance in Ad Hoc Task Groups. In J. Galegher, R. F. Kraut & C. Egido (Eds.), *Intellectual Teamwork*. Hillsdale, NJ: Lawrence Erlbaum and Associates.

Franck, E., & Jungwirth, C. (2002). *Reconciling investors and donators: The governance structure of open source* (Working Paper No. No. 8): Lehrstuhl für Unternehmensführung und -politik, Universität Zürich.

Gacek, C., & Arief, B. (2004). The many meanings of Open Source. *IEEE Software, 21*(1), 34–40.

Galbraith, J. R. (1973). *Designing Complex Organizations.* Reading, MA: Addison-Wesley.

Grabowski, M., & Roberts, K. H. (1999). Risk mitigation in virtual organizations. *Organization Science, 10*(6), 704–721.

Grinter, R. E., Herbsleb, J. D., & Perry, D. E. (1999). The Geography of Coordination: Dealing

with Distance in R&D Work. In *Proceedings of the GROUP '99 Conference* (pp. 306–315). Phoenix, Arizona, US.

Hallen, J., Hammarqvist, A., Juhlin, F., & Chrigstrom, A. (1999). Linux in the workplace. *IEEE Software, 16*(1), 52–57.

Hann, I.-H., Roberts, J., Slaughter, S., & Fielding, R. (2002). Economic incentives for participating in open source software projects. In *Proceedings of the Twenty-Third International Conference on Information Systems* (pp. 365–372).

Herbsleb, J. D., & Grinter, R. E. (1999a). Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*(September/October), 63–70.

Herbsleb, J. D., & Grinter, R. E. (1999b). *Splitting the organization and integrating the code: Conway's law revisited*. Paper presented at the Proceedings of the International Conference on Software Engineering (ICSE '99), Los Angeles, CA.

Herbsleb, J. D., Mockus, A., Finholt, T. A., & Grinter, R. E. (2001). *An empirical study of global software development: Distance and speed*. Paper presented at the Proceedings of the International Conference on Software Engineering (ICSE 2001), Toronto, Canada.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel. *Research Policy, 32*(7), 1159–1177.

Humphrey, W. S. (2000). *Introduction to Team Software Process*: Addison-Wesley.

Iannacci, F. (2005). Coordination processes in OSS development: The Linux case study. Retrieved 21 September, 2006, from http://opensource.mit.edu/papers/iannacci3.pdf

Jarvenpaa, S. L., & Leidner, D. E. (1999). Communication and trust in global virtual teams. *Organization Science, 10*(6), 791–815.

Jensen, C., & Scacchi, W. (2005). Collaboration, Leadership, Control, and Conflict Negotiation in the Netbeans.org Open Source Software Development Community. In *Proceedings of the Hawai'i International Conference on System Science (HICSS 2005).* Big Island, Hawai'i.

Kaplan, B. (1991). Models of change and information systems research. In H.-E. Nissen, H. K. Klein & R. Hirschheim (Eds.), *Information Systems Research: Contemporary Approaches and Emergent Traditions* (pp. 593–611). Amsterdam: Elsevier Science Publishers.

Kogut, B., & Metiu, A. (2001). Open-source software development and distributed innovation. *Oxford Review of Economic Policy, 17*(2), 248–264.

Kraut, R. E., Steinfield, C., Chan, A. P., Butler, B., & Hoag, A. (1999). Coordination and virtualization: The role of electronic networks and personal relationships. *Organization Science, 10*(6), 722–740.

Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM, 38*(3), 69–81.

Krishnamurthy, S. (2002). Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday, 7*(6).

Lawrence, P., & Lorsch, J. (1967). *Organization and Environment*. Boston, MA: Division of Research, Harvard Business School.

Leibovitch, E. (1999). The business case for Linux. *IEEE Software, 16*(1), 40–44.

Lerner, J., & Tirole, J. (2001). The open source movement: Key research questions. *European Economic Review, 45*, 819–826.

Madanmohan, T. R., & Navelkar, S. (2002). *Roles and Knowledge Management in Online Technology Communities: An Ethnography Study* (Working paper No. 192): IIMB.

Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *Computing Surveys, 26*(1), 87–119.

Markus, M. L., Manville, B., & Agres, E. C. (2000). What makes a virtual organization work? *Sloan Management Review, 42*(1), 13–26.

Markus, M. L., & Robey, D. (1988). Information technology and organizational change: Causal structure in theory and research. *Management Science, 34*(5), 583–598.

Massey, A. P., Hung, Y.-T. C., Montoya-Weiss, M., & Ramesh, V. (2001). When culture and style aren't about clothes: Perceptions of task-technology "fit" in global virtual teams. In *Proceedings of GROUP '01.* Boulder, CO, USA.

McCann, J. E., & Ferry, D. L. (1979). An approach for assessing and managing inter-unit interdependence. *Academy of Management Review, 4*(1), 113–119.

Metiu, A., & Kogut, B. (2001). *Distributed Knowledge and the Global Organization of Software Development* (Working paper). Philadelphia, PA: The Wharton School, University of Pennsylvania.

Mintzberg, H. (1979). *The Structuring of Organizations.* Englewood Cliffs, NJ: Prentice-Hall.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies Of Open Source Software development: Apache And Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309–346.

Mohr, L. B. (1971). Organizational technology and organizational structure. *16*, 444–459.

Mohr, L. B. (1982). *Explaining Organizational Behavior: The Limits and Possibilities of Theory and Research*. San Francisco: Jossey-Bass.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of Linux kernel. *First Monday, 5*(11).

Nejmeh, B. A. (1994). Internet: A strategic tool for the software enterprise. *Communications of the ACM, 37*(11), 23–27.

O'Leary, M., Orlikowski, W. J., & Yates, J. (2002). Distributed work over the centuries: Trust and control in the Hudson's Bay Company, 1670–1826. In P. Hinds & S. Kiesler (Eds.), *Distributed Work* (pp. 27–54). Cambridge, MA: MIT Press.

Orlikowski, W. J. (2002). Knowing in practice: Enacting a collective capability in distributed organizing. *Organization Science, 13*(3), 249–273.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM, 15*(2), 1053–1058.

Pfaff, B. (1998). Society and open source: Why open source software is better for society than proprietary closed source software. from http://www.msu.edu/user/pfaffben/writings/anp/oss-is-better.html

Pfeffer, J. (1978). *Organizational Design*. Arlington Heights, IL: Harlan Davidson.

Pfeffer, J., & Salancik, G. R. (1978). *The External Control of Organizations: A Resource Dependency Perspective*. New York: Harper & Row.

Prasad, G. C. (n.d.). A hard look at Linux's claimed strengths…. from http://www.osopinion.com/Opinions/GaneshCPrasad/GaneshCPrasad2-2.html

Raymond, E. S. (1998). The cathedral and the bazaar. *First Monday, 3*(3).

Robey, D., Khoo, H. M., & Powers, C. (2000). Situated-learning in cross-functional virtual teams. *IEEE Transactions on Professional Communication*(Feb/Mar), 51–66.

Sabherwal, R., & Robey, D. (1995). Reconciling variance and process strategies for studying information system development. *Information Systems Research, 6*(4), 303–327.

Sandusky, R. J., Gasser, L., & Ripoche, G. (2004). *Bug Report Networks: Varieties, Strategies, and Impacts in an OSS Development Community.* Paper presented at the Proceedings of the ICSE Workshop on Mining Software Repositories, Edinburgh, Scotland, UK.

Sawyer, S., & Guinan, P. J. (1998). Software development: Processes and performance. *IBM Systems Journal, 37*(4), 552–568.

Scacchi, W. (1991). The software infrastructure for a distributed software factory. *Software Engineering Journal, 6*(5), 355–369.

Scacchi, W. (2002). Understanding the requirements for developing Open Source Software systems. *IEE Proceedings Software, 149*(1), 24–39.

Scacchi, W. (2005). Socio-technical interaction networks in Free/Open Source Software development processes. In S. T. Acuña & N. Juristo (Eds.), *Software Process Modeling* (pp. 1–27). New York: Springer.

Stewart, K. J., & Ammeter, T. (2002). An exploratory study of factors influencing the level of vitality and popularity of open source projects. In *Proceedings of the Twenty-Third International Conference on Information Systems* (pp. 853–857).

Taylor, P. (1998, December 2). New IT mantra attracts a host of devotees. *Financial Times, Survey—Indian Information Technology,* p. 1.

Thompson, J. D. (1967). *Organizations in Action: Social Science Bases of Administrative Theory.* New York: McGraw-Hill.

Torvalds, L. (1999). The Linux edge. *Communications of the ACM, 42*(4), 38–39.

Valloppillil, V. (1998). Halloween I: Open Source Software. from http://www.opensource.org/halloween/halloween1.html

Valloppillil, V., & Cohen, J. (1998). Halloween II: Linux OS Competitive Analysis. from http://www.opensource.org/halloween/halloween2.html

Victor, B., & Blackburn, R. S. (1987). Interdependence: An alternative conceptualization. *Academy of Management Review, 12*(3), 486–498.

Walz, D. B., Elam, J. J., & Curtis, B. (1993). Inside a software design team: knowledge acquisition, sharing, and integration. *Communications of the ACM, 36*(10), 63–77.

Watson-Manheim, M. B., Chudoba, K. M., & Crowston, K. (2002). Discontinuities and continuities: A new way to understand virtual work. *Information, Technology and People, 15*(3), 191–209.

Wayner, P. (2000). *Free For All*. New York: HarperCollins.

Webb, E., & Weick, K. E. (1979). Unobtrusive measures in organizational theory: A reminder. *Administrative Science Quarterly, 24*(4), 650–659.

Weber, S. (2004). *The Success of Open Source*. Cambridge, MA: Harvard.

Weisband, S. (2002). Maintaining awareness in distributed team collaboration: Implications for leadership and performance. In P. Hinds & S. Kiesler (Eds.), *Distributed Work* (pp. 311–333). Cambridge, MA: MIT Press.

Zuboff, S. (1988). *In the Age of the Smart Machine*. New York: Basic Books.

ENDNOTE

[1] This research was partially supported by US NSF Grants 03-41475, 04–14468 and 05-27457. An earlier version of this article was presented at the *First International Workshop on Computer Supported Activity Coordination (CSAC 2004).* The authors thank previous anonymous reviewers of the article for their comments that have helped to improve the article.

This work was previously published in the Journal of Database Management, edited by K. Siau, Volume 19, Issue 2, pp. 1-30, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 2.30
Evaluating Performance of Software Architecture Models with the Palladio Component Model

Heiko Koziolek
Universität Oldenburg, Germany

Steffen Becker
University of Karlsruhe, Germany

Ralf Reussner
University of Karlsruhe, Germany

Jens Happe
Universität Oldenburg, Germany

ABSTRACT

Techniques from model-driven software development are useful to analyse the performance of a software architecture during early development stages. Design models of software models can be transformed into analytical or simulation models, which enable analyzing the response times, throughput, and resource utilization of a system before starting the implementation. This chapter provides an overview of the Palladio Component Model (PCM), a special modeling language targeted at model-driven performance predictions. The PCM is accompanied by several model transformations, which derive stochastic process algebra, queuing network models, or Java source code from a software design model. Software architects can use the results of the analytical models to evaluate the feasibility of performance requirements, identify performance bottlenecks, and support architectural design decisions quantitatively. The chapter provides

a case study with a component-based software architecture to illustrate the performance prediction process.

INTRODUCTION

To ensure the quality of a software model, developers need not only to check its functional properties, but also assure that extra-functional requirements of the system can be fulfilled in an implementation of the model. Extra-functional properties include performance, reliability, availability, security, safety, maintainability, portability, etc. Like functional correctness, these properties need to be addressed already during early development stages at the model level to avoid possible later costs for redesign and reimplementation.

Performance (i.e., response time, throughput, and resource utilization) is an extra-functional property critical for many business information systems. Web-based information systems rely on fast response times and must be capable of serving thousands of users in a short time span due to the competitive nature of internet businesses. Furthermore, the responsiveness of software used within companies is important to ensure efficient business processes.

Performance problems in large distributed systems can sometimes not be solved by adding more servers with improved hardware ("kill it with iron"). Large software architectures often do not scale linearly with the available resources, but instead include performance bottlenecks that limit the impact of additional hardware.

Therefore, it is necessary to design a software architecture carefully and analyse performance issues as early as possible. However, in the software industry, performance investigations of software systems are often deferred until an implementation of the system has been build and measurements can be conducted ("fix it later"). To avoid this approach, which might lead to expensive redesigns, software architects can use performance models for early, pre-implementation performance analysis of their architectures.

This chapter provides an overview of the Palladio Component Model (PCM), a domain specific modelling language for component-based software architectures, which is specifically tuned to enable early life-cycle performance predictions. Different developer roles can use the PCM to model the software design and its targeted resource environment. The models can be fed into performance analysis tools to derive the performance of different usage scenarios. Software architects can use this information to revise their architectures and quantitatively support their design decisions at the architectural level.

The chapter is structured as follows: Section 2 provides background and describes related work in the area of model-driven performance prediction. Section 3 introduces different developer roles and a process model for model-driven performance predictions. Section 4 gives an overview of the PCM with several artificial model examples, before Section 5 briefly surveys different model transformations to analysis models and source code. Section 6 describes the performance prediction for an example component-based software architecture and discusses the value of the results for a software architect. For researchers interested working in the area of model-driven performance prediction, Section 7 highlights some directions for future research. Section 8 concludes the chapter.

BACKGROUND AND RELATED WORK

Model-driven performance predictions aim at improving the quality of software architectures during early development stages (Smith et al., (2002)). Software architects use models of such prediction approaches to evaluate the response time, throughput, or resource utilization to be expected after implementing their envisioned

design. The prediction model's evaluation results enable analysing different architectural designs and validate performance-related requirements (such as maximum response times or minimum throughput) of software systems. The advantage of using prediction models instead of testing implementations is the lowered risk to find performance problems in already implemented systems, which require cost-intensive redesigns.

Researchers have put much effort into creating accurate performance prediction models for the last 30 years. Queuing networks, stochastic process algebras, and stochastic Petri nets are the most prominent prediction models from the research community. However, practitioners seldom apply these models due to their complexity and high learning curve. Therefore, focus of the research community has shifted to create more developer-friendly models and use model transformations to bridge the semantic gap to the above mentioned analytical models.

From the more than 20 approaches in this direction during the last decade (Balsamo et al., (2004)), most use annotated UML models as a design model and ad-hoc transformations to create (layered) queuing networks as analytical models. Tools encapsulate the transformation to the analytical models and their solution algorithms to limit the necessary additional skills for designers. For these approaches, the Object Management Group (OMG) has published multiple UML profiles (SPT profile cf. OMG, (2005); QoS/FT profile; MARTE profile) to add performance-related annotations to UML models. However, these profiles remain under revision, are still immature, and are still not known to have been used in practise in a broader scope.

Component-based software engineering (CBSE) adds a new dimension to model-driven performance prediction approaches. CBSE originally targeted at improved reusability, more flexibility, cost-saving, and shorter time-to-market of software systems (Szyperski et al. (2002)). Besides these advantages, CBSE might also ease prediction of extra-functional properties. Software developers may test components for reuse more thoroughly and provide them with more detailed specifications. These specifications may contain performance-related information.

Hence, several research approaches have tackled the challenge of specifying the performance of a software component (cf. survey by Becker et al., (2006)). This is a difficult task, as the performance of a component depends on environmental factors, which can and should not be known by component developers in advance. These factors include:

- **Execution environment:** The platform a component is deployed on including component container, application server, virtual machine, operating system, software resources, hardware resources
- **Usage profile:** User inputs to component services and the overall number of user requests directed at the components
- **Required services:** Execution times of additionally required, external services, which add up to the execution of the component itself

Component developer can only fix the component's implementation, but have to provide a performance specification, which is parameterisable for the execution environment, the usage profile, and the performance of required services. The following paragraph summarises some of the approaches into this direction.

Sitaraman et. al (2001) model the performance of components with an extension to the O-calculus, but do not include calls to required services. Hissam et. al (2002) aim at providing methods to certify component for their performance properties. Bertolino et. al (2003) use the UML SPT profile to model component-based systems including dependencies to the execution environment, but neglecting influences by the usage profile. Hamlet et al. (2003) investigate

the influence of the usage profile on component performance. Wu et al. (2004) model components with an XML-based language and transform this notation into layered queueing networks. The APPEAR method by Eskenazi et al. (2004) aims at predicting performance for changes on already built systems, and thus does neglect the influence of the execution environment. Bondarev et al. (2005) target components in embedded systems with the ROBOCOP component model. Grassi et al. (2005) develop an intermediate modelling language for component-based systems called KLAPER, which shall bridge the gap between different design and analytical models.

The Palladio Component Model (Becker et al., (2007)) described in this chapter is in line with these research approaches and tries to reflect all influences on component performance. Unlike some of the above listed approaches, the PCM does not use annotated UML as design model, but defines its own metamodel. This reduces the model to concepts necessary for performance prediction and does not introduce the high complexity of arbitrary UML models with a variety of concepts and views.

DEVELOPER ROLES AND PROCESS MODEL

The PCM metamodel is divided into several domain-specific modelling languages, which are aligned with developer roles in CBSE. This section introduces these roles and provides an overview of the process model for using the PCM.

An advantage of CBSE is the division of work between different developer roles, such as component developers and software architects. *Component developers* specify and implement components. They also have to provide a description of the component's extra-functional properties to enable software architects to predict their performance without deploying and testing them. *Software architects* compose components from different component developers to application architectures. They are supported by tools to predict the architecture's performance based on the performance specifications of the component developers. With the predicted performance metrics, they can support their design decisions for different architectural styles or components.

For performance predictions, the software architect needs additional information about the execution environment and the usage profile. The role of the *system deployer* provides performance-related information about the hardware/software environment of the architecture (such as processing rate of a CPU, throughput of a network link, scheduling policies of the operating system, configuration of the application server, etc.). Business *domain experts* provide knowledge about the anticipated user behavior (in terms of input parameters and call frequencies), and must assist software architects in specifying a usage model of the architecture.

Figure 1 depicts the overall development process of a component-based system including performance prediction (Koziolek et al. (2006)): Boxes model workflows, thick and grey arrows indicate a change of activity, and thin and black arrows illustrate the flow of artefacts. The workflows do not have to be traversed linearly; backward steps for revision are likely. After collecting and analysing requirements for the system to develop (Requirements), the software architect specifies components and the architecture based on input by component developers (Specification). With a fully specified architecture, performance predictions can be carried out by tools (QoS-analysis). The software architect can use the results to alter the specification or decide to implement the architecture. This is done either by obtaining existing components from third-party vendors or by implementing them according to their specification (Provisioning). Afterwards, the software architect can compose the component implementations (Assembly), test the full application in a restricted environment (Test), and then install and

operate it in the customer's actual environment (Deployment).

During "Specification", the above introduced roles interact as follows (cf. Figure 2): The PCM provides a domain-specific modelling language for each developer role, which is restricted to its known concepts. Component developers model performance-related component behaviour, software architects add an assembly model. System deployers model hardware/software resources and the components' allocation to these resources. Finally, domain experts provide a usage model.

Figure 1. Component-based development process (©2007 Heiko Koziolek. Used with permission)

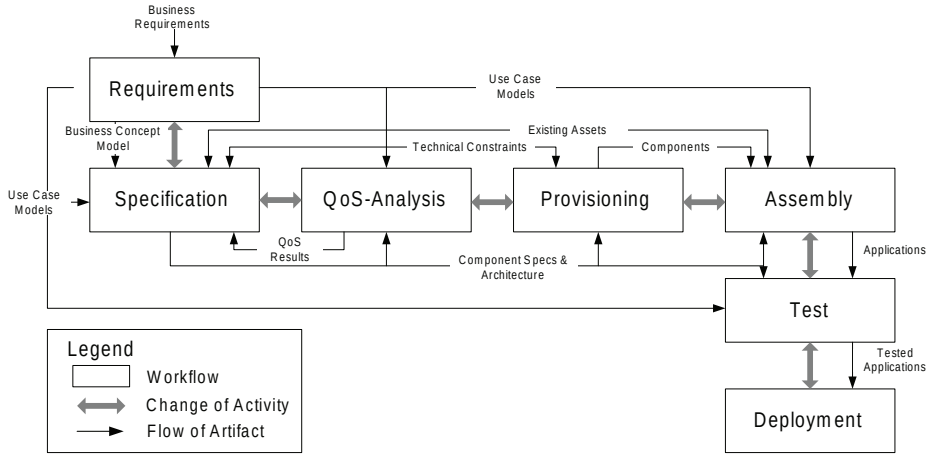
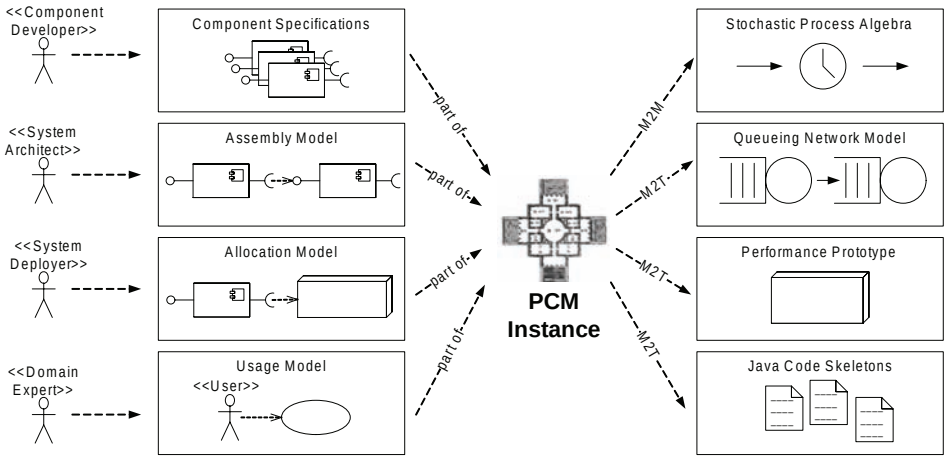


Figure 2. Specification and QoS analysis with the PCM (©2007 Heiko Koziolek. Used with permission)



All specifications can be combined to derive a full PCM instance. Section 4 will elaborate on the PCM's specification languages.

During "QoS-Analysis", this model can be transformed into different analysis models, such as a stochastic process algebra or a queueing network. These models provide capabilities to derive performance metrics such as response times, throughputs, or resource utilisations for specific usage scenarios. Additionally, the PCM can be transformed into a performance prototype, which simulates the specified resource demands. This prototype enables pre-implementation performance measurements on the target platform. Finally, the PCM instance can be converted into Java code skeletons via model-2-text transformation, as a starting point for implementing the system's business logic. Section 5 describes the analysis models and code transformations in more detail.

OVERVIEW PALLADIO COMPONENT MODEL

This section provides an overview of the modeling capabilities of the PCM to describe component-based software architecture. The PCM is a metamodel specified in Ecore from the Eclipse Modelling Framework (EMF). The following section will mainly use examples to introduce the concepts, and does not go into technical details of the metamodel, which are elaborated in (Reussner et al., 2007). The description of the PCM in this section is structured along the developer roles and their domain-specific languages.

Component Developer

Component developers specify the functional and extra-functional properties of their components. They put the specification as well as the implementation in repositories, where software architects can retrieve them. This section will first introduce all entities, which can be stored in repositories and then focus on service effect specifications, which model the abstract behavior and performance properties of component services.

Component Repositories

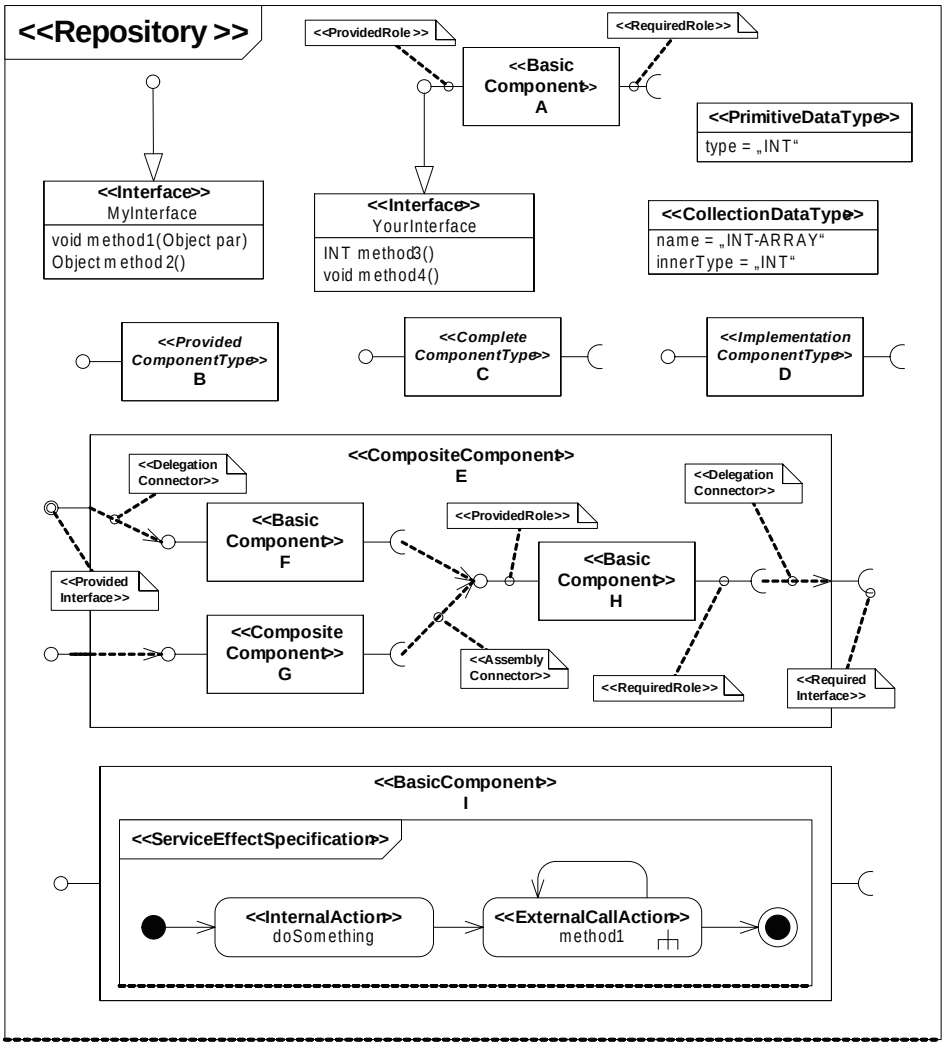
Figure 3 shows an example PCM repository, which includes all types of entities that can be specified. First class entities in PCM repositories are interfaces, data types, and components. They may exist on their own and do not depend on other entities.

The *interface* `MyInterface` is depicted on the upper left in Figure 3. It is not yet bound to a component, but can be associated as a provided or required interface to components. An example of interfaces existing without clients and an implementation in practice was the Java Security API, which had been specified by Sun before an implementation was available. Interfaces in the PCM contain a list of service signatures, whose syntax is based on CORBA IDL. Additionally, component developers may supplement an interface with protocols, which restrict the order of calling its services. For example, an I/O interface might force clients to first open a file (call service open()) before reading from it (call service read()).

Components may provide or require interfaces. The binding between a component and an interface is called "provided role" or "required role" in the PCM. For example, component `A` in Figure 3 is bound to `YourInterface` in a provided role. This means that the component includes an implementation for each of the services declared in the interface. Other components, which are bound to a compliant interface in a required role can use component `A` to execute these services.

Repositories need common data types, so that the service signatures refer to standardized types (e.g., INT, FLOAT, CHAR, BOOL, STRING, etc.). In the PCM, data types are either primitive types, collection types, or composite types

Figure 3. Example component repository (©2007 Heiko Koziolek. Used with permission)



(composed out of inner types). Figure 3 contains a primitive data type `INT` and a collection data type `INT-Array`, which contains `INT`s as inner elements.

The PCM supports modeling different types of components to a) reflect different development stages, and b) to differentiate between basic (atomic) components and composite components.

Different development stages are reflected by *provided, complete, and implementation component type*. Component developers can refine components during design from provided to implementation component types.

Provided component types (component `B` in Figure 3) only provide one or more interfaces, but include no mandatory required interfaces. Compo-

nent developers can use these type of components early during the development, when they know that a certain functionality has to be provided, but do not know whether other components are needed to provide this functionality.

Complete component types (component C in Figure 3) are provided component types, but additionally may contain mandatory required interfaces. However, the inner dependencies between provided and required interfaces are not fixed in complete component types, as different implementations can lead to different dependencies. Within a component architecture, a software architect may easily replace one component with another component, which conforms (i.e., implements the same provided and required interfaces) to the same complete component type, without affecting the system's functionality.

Implementation component types (component D in Figure 3) are complete component types, but additionally contain fixed inner dependencies between provided and required interfaces. Replacing implementation component types in an architecture ensures not only signature but also protocol compatibility at the required interface.

Implementation component types are either basic (i.e., implemented from scratch) or composite components (i.e., implemented by composing other components). Component E in Figure 3 is a *composite component.* It contains several inner components (F, G, H). Inner component may again be composite components (G) to build up arbitrary hierarchies. Assembly connectors bind the roles of inner components. Delegation connectors connect provided roles of composite components with provided roles of inner components, or required roles of composite components with required roles of inner components. From the outside, composite components look like basic components, as they provide and require services. The inner structure of a composite component should only be known to the component developer, but not to the software architect, who shall use the component as a unit and treat it the same as other components.

Finally, basic components are atomic and therefore cannot be further decomposed. They may contain a mapping for each provided service to required services, which is called resource demanding service effect specification.

Service Effect Specification

Resource demanding service effect specifications (RDSEFF) provide means to describe resource demands and calls to required services by a provided component service. Component developers use RDSEFFs to specify the performance of their components.

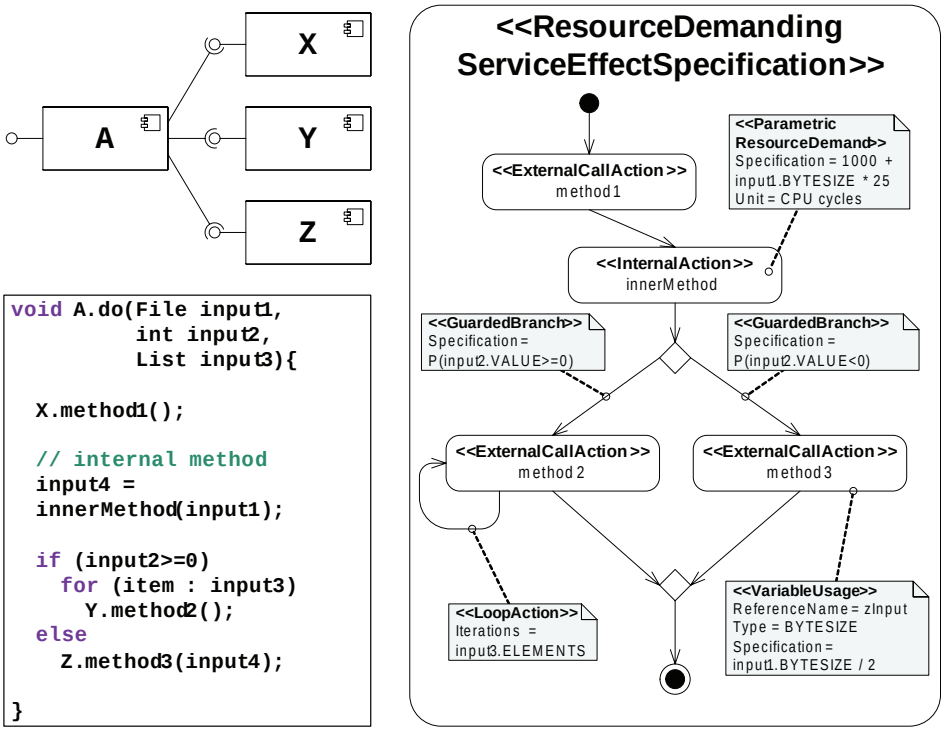
RDSEFFs reflect the environmental factors on component performance introduced in Section 2. These are external services, execution environment, usage profile, and component implementation as described in the following subsection.

RDSEFFs abstractly model the externally observable behavior of a component service. They only refer to method signatures and parameters that are declared in the interfaces and not to local, private variables. They abstractly model control flow between calls to required services, parametric dependencies, and resource usage. These specifications do not reveal any additional knowledge about the algorithms used to implement the service's functionality and thus retain the black-box principle.

Consider the artificial example in Figure 4 for a brief introduction into RDSEFFs. Component A invokes required functionality via its required X,Y, and Z. It provides a service called "`do`", whose source code is sketched in Figure 4. The service first calls a service from interface X, and then executes some internal code processing parameter "`input1`". Afterwards, depending on "`input2`", either services from interface Y or Z are called. "`method2`" from interface Y is located within a loop, whose number of iterations depends on the array length of "`input3`".

The corresponding RDSEFF for service "`do`" is located on the right hand side in Figure 4. As a

Figure 4. Resource demanding service effect specification (©2007 Heiko Koziolek. Used with permission)



graphical, concrete syntax, the illustration uses the UML activity diagram notation. However, in this case, the metamodel underlying the modeling constructs is not the UML metamodel, but the PCM, which is indicated by enclosing the PCM class names in brackets. In the following, the underlying concepts for control flow, resource demands, and parametric dependencies will be described.

Control Flow: Actions in RDSEFFs can either be internal actions (i.e., the component executes some internal code) or external call actions (i.e., the component calls a service declared in its required interface). RDSEFF offer as basic control flow constructs sequences, alternatives, loops, and parallel executions (forks). The order of these actions may influence performance properties of the service, because different services may concurrently use the same resources or synchronize each other, which induces delays for waiting.

Alternatives or branches split the control flow with an XOR semantic (with guards covering the whole input domain of parameters), while forks (not depicted in Figure 4) split the control flow with an AND semantic, i.e., all following actions are executed concurrently. Loops have to specify

the number of iterations, so that the execution times for actions within the loop can be added up a limited number of times.

Notice that the control flow in RDSEFFs is an abstraction from the actual inner control flow of the service. Internal actions potentially summarize a large number of inner computations and control flow constructs, which do not contain calls to required services.

Resource Demands: Besides external services, a component service accesses the resources of the execution environment it is deployed in. Ideally, component developers would provide measured execution times for these resource accesses in the RDSEFF. However, these measured times would be useless for software architects, who want to use the component, because their hardware/software environment can be vastly different from the component developer ones. The execution times of the service could be much faster or slower in the software architect's environment.

Therefore, component developers specify resource demands in RDSEFFs against abstract resource types such as a CPU or hard disk. For example they can provide the number of CPU cycles needed for execution or the number of bytes read from or written to a hard disk. The resource environment model supplied by the system deployer (Section 4.3) then contains execution times for executing CPU cycles or reading a byte from hard disk. These values can be used to calculate the actual execution times of the resource demands supplied by the component developers. As an example, the "ParametricResourceDemand" on the internal action "`method1`" in Figure 4 specifies that the service needs 1000 CPU cycles plus the amount of a parametric dependency (described in the next paragraph) to execute.

In addition to active resources, such as processors, storage devices, and network devices, component services may also acquire and release passive resources, such as threads, semaphores, database connections etc. Passive resources are not capable of processing requests and usually exist only a limited number of times. A service can only continue its execution, if the required amount of resources is available. Acquisition/Release of passive resources is not depicted in Figure 4.

Parametric Dependencies: To include the influence of the usage profile into the RDSEFF, component developers can specify parametric dependencies. When specifying an RDSEFF, component developers cannot know how the component will be used by third parties. Thus they cannot fix resource demands, branching probabilities or the number of loop iterations if those values depend on input parameters. Hence, RDSEFFs allow specifying dependencies to input parameters.

There are several forms of these dependencies. For example, in Figure 4, the resource demand of the internal action "`innerMethod`" depends on byte size of input parameter "`input1`" (e.g., because the method processes the file byte-wise). Once the domain expert characterizes the actual size of this parameter (cf. Section 4.4), this value can be used to calculate the internal action's actual resource demand.

Furthermore, branching probabilities are needed for the alternative execution paths in this RDSEFF. These probabilities are however not fixed, but depend on the value of input parameter "`input2`". Therefore, the RDSEFF includes no branching probabilities but guards (i.e., Boolean expressions) on the branches. Once the domain expert characterizes the possible values of "`input2`" and provides probabilities for the input domains "`input2>=0`" and "`input2<0`", these values can be mapped to the branching probabilities.

The RDSEFF in Figure 4 also contains a parametric dependency on the number of loop iterations surrounding the external call to "`method2`" of component `Y`. Loop iterations can be fixed in the code, but sometimes they depend on input

parameters. In this case the service iterates over the list "`input3`" and calls the external service for each of its elements. The RDSEFF specifies this dependency as the component developer cannot know in advance the lengths of the lists.

Finally, the service "`do`" executes the external call to "`method3`" in Figure 4 with an input parameter that in turn depends on an input parameter of the service itself. The service processes "`input1`", assigns it to a local variable "`input4`", and then forwards it to interface `Z` via "`method3`". While processing "`input1`", the service "`do`" reduces its byte size by 50% ("`input1.BYTESIZE / 2`"). The RDSEFF includes the specification of this dependency. Once the domain expert specifies the actual byte size of "`input1`", the byte size of the input parameter of "`method3`" can be calculated.

Software Architect

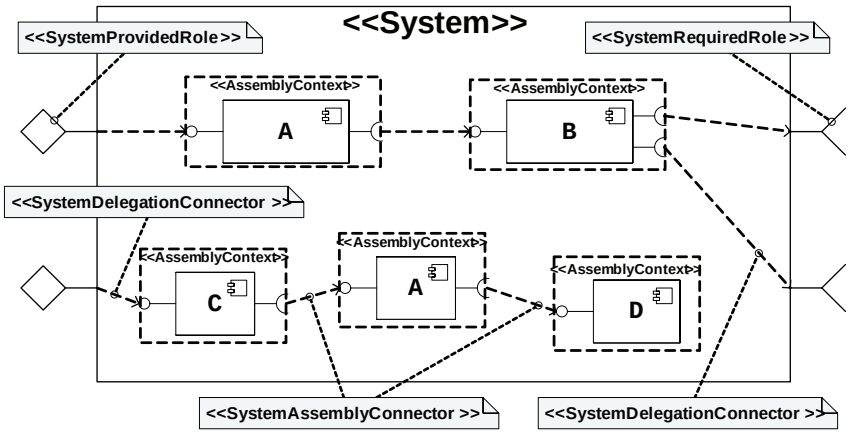
Software architects retrieve components (including their RDSEFFs) from repositories and compose them to architectures. They can use several component instances of the same type in an architecture at different places. Hence, in the PCM, software architects put component instances in so called assembly contexts, which save the connections of a single component instance.

Software architects bind the roles of components in assembly contexts with system assembly connectors, as illustrated in the example in Figure 5. Notice that the component type A is used in two assembly contexts in this example (once connected with component B and once with C and D).

A set of connected assembly contexts is called *assembly*. An assembly is part of a *system*, which additionally exposes system provided roles and system required roles (cf. Figure 5). System delegation connectors bind these system roles with roles of the system's inner components. Domain experts later use system provided roles to model the usage of the system (Section 4.4). System required roles model external services, which the software architect does not consider part of the architecture. For example, the software architect can decide to model a web service or a connected database as system external services.

There is a distinction between composite components and systems. For software architects

Figure 5. System example (©2007 Heiko Koziolek. Used with permission)



and system deployers, composite components hide their inner structure and the fact that they are composed from other components. The inner structure is an implementation detail and its exposure would violate the information hiding principle of components. Opposed to this, the structure of assemblies is visible to software architects and system deployers. Therefore, system deployers can allocate each component in a system to a different resource. However, they cannot allocate inner components of composite components to different resources, because these stay hidden from them at the architectural level.

System Deployer

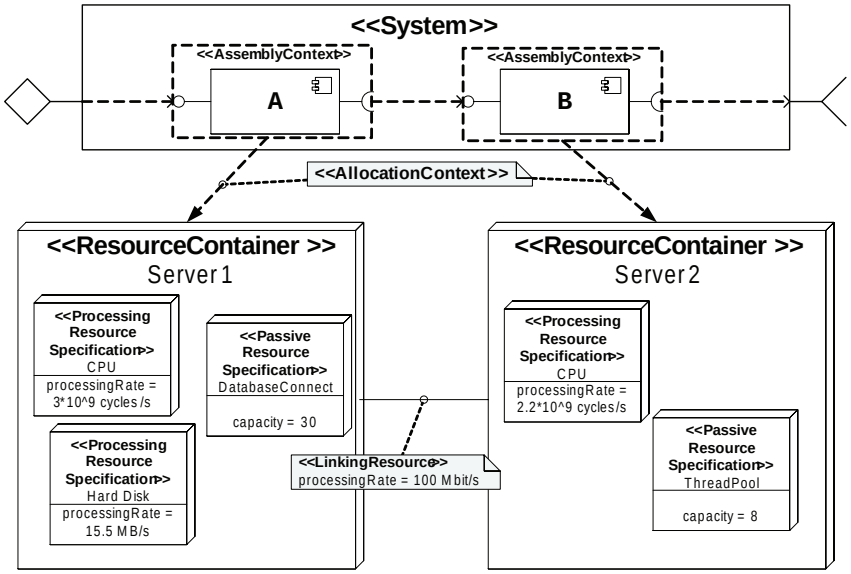
System deployers first specify the system's resource environment and then allocate assembly contexts (i.e., connected component instances) to resources.

In *resource environments*, resource containers group resources. For example, in Figure 6, the resource container "Server1" contains a CPU, a hard disk, and a database connection pool. The PCM differentiates between processing resources, which can execute requests (e.g., CPU, hard disk, memory), and passive resources, which cannot execute requests, but only be acquired and released (e.g., threads, semaphores, database connections).

Processing resources specify a processing rate, which can be used to convert the resource demands in RDSEFFs into timing values. Passive resources specify a capacity. If a component acquires a passive resource, its amount of available units (i.e., its capacity) decreases. Once the capacity reaches zero, further components requesting the passive resource must wait until other services release it again. Linking resources connect resource containers and are themselves special processing resources.

System deployers use *allocation contexts* to specify that a resource container executes an assembly context. In Figure 6, the system deployer

Figure 6. Resource environment and allocation (©2007 Heiko Koziolek. Used with permission)



has allocated component A's assembly context to "Sever1" and component B's assembly context to "Server2".

System deployers can specify different resource environments and different allocation contexts to answer sizing questions. The PCM's resource model is still limited to abstract hardware resources. We will extend it in the future with middleware parameter, operating system settings, and scheduling policies.

Domain Expert

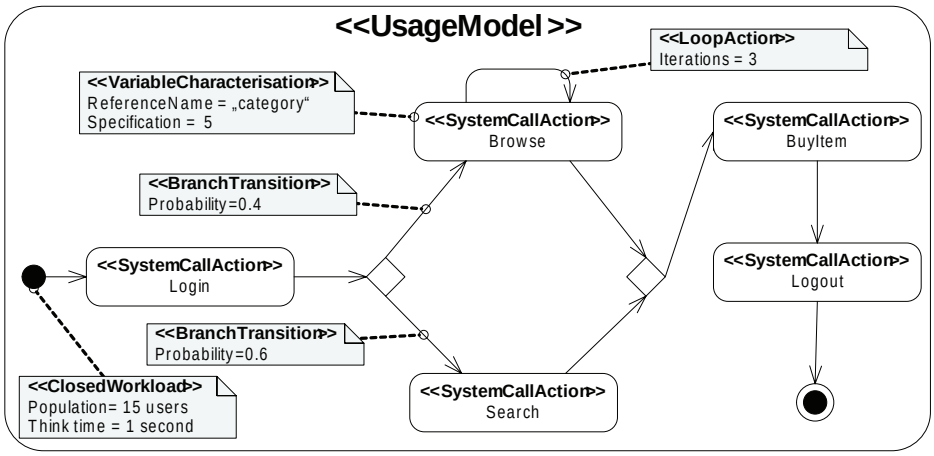
Domain experts create a usage model that characterizes user behavior and connects to system provided roles. In the example in Figure 7, users first log in to the system, then either browse or search, then buy an item, and finally log out. All actions target system provided roles (i.e., services exposed by the system, cf. Section 4.2).

Domain experts can specify user behavior with control flow constructs such as sequence, alternative, and loop. They must specify branching probabilities for alternatives and the number of iterations for loops.

Additionally, domain experts specify the user workload. Workloads are either closed or open. Closed workloads specify a fixed number of users (population) circulating in the system. In Figure 7, the domain expert has specified a closed workload with 15 users, which perform the specified actions and then re-enter the system after a think time of 1 second. Open workloads specify a user arrival rate (e.g., 5 users/second), and do not limit the number of users in the system.

The PCM usage model also enables domain experts to characterize the parameter values of users. In Figure 7, variable "category" of action browse has been characterized with a constant (5) meaning that users always browse in the category with id number 5. Besides constants, the usage model offers specifying probability distribution functions over the input domain of a parameter, so that domain experts can provide a fine-grained stochastic characterization of the user's input parameters. The reader may find details in Reussner et al. (2007).

Figure 7. Usage model example (©2007 Heiko Koziolek. Used with permission)



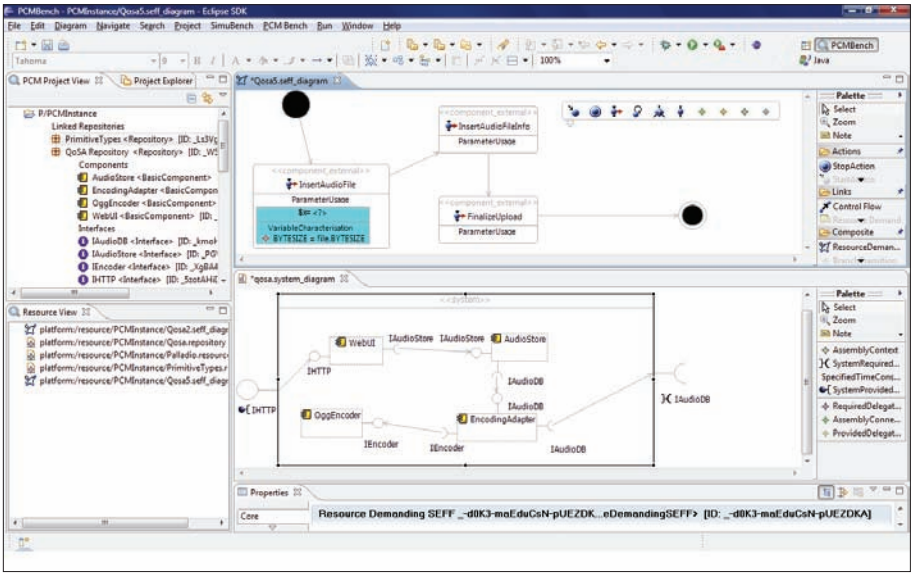
Tool Support

We have implemented an Eclipse-based open-source tool called "PCM-Bench", which enables software developers to create instances of the PCM metamodel and run performance analyses (cf. Figure 8). The tool offers a different view perspective for each of the four developer roles and provides graphical model editors. Models of the different developer roles reference each other in the editor, which enables the creation of a full PCM instance. The PCM-Bench is an Eclipse RCP application and its editors have been partially generated from the PCM Ecore metamodel with support of the Graphical Modelling Framework (GMF).

The graphical editors provide an intuitive way of modeling component-based architectures analogous to UML modeling tools. They offer model validation by checking OCL-constraints (Object Constraint Language). The PCM-Bench visualizes violated constraints directly in the model diagrams. The editors support entering performance annotations with special input masks that offer syntax highlighting and code completion. Model instances can be serialized to XMI-files.

Besides graphical editors, the PCM-Bench is a front-end for the performance analysis techniques described in Section 5. Software architects can configure and run simulations. They can retrieve different performance metrics such as response times for use cases, throughputs, and resource utilizations. The PCM-Bench visualizes probability distribution functions of response times as histograms and provides standard statistical values such as mean, median, standard deviation etc. Furthermore, the PCM-Bench supports Model-to-Text transformations to generate Java code from PCM instances.

Figure 8. Screenshot PCM-bench (©2007 Heiko Koziolek. Used with permission)



MODEL TRANSFORMATION AND PREDICTION METHODS

The PCM offers different performance evaluation techniques, which are still subject to research. For analyzing use cases without concurrency, a PCM instance can be transformed into a stochastic process algebra (SPA), which offers a fast way of predicting response times (Section 5.1). A PCM instance can alternatively be transformed into a queuing network based simulation model (Section 5.2). The simulation model is less restricted than the SPA, but its execution is usually more time consuming than solving the SPA. Finally, there are transformations to derive Java code skeletons from a PCM instance, to provide a starting point for implementing the modeled architecture (Section 5.3).

Stochastic Process Algebra

The PCM-Bench supports a model-2-model transformation of a PCM instance into an SPA called Capra (Happe et. al (2007), Koziolek et. al (2007c)). Capra includes concurrent processes, resources with current operating systems scheduling policies, and is able to efficiently incorporate timing values specified as arbitrary probability distributions into the analysis process. It employs a hybrid approach of analysis and simulation to conduct the performance prediction.

To transform a PCM instance into a Capra expression, a tool first solves the parametric dependencies within the RDSEFFs. They use the parameter characterizations provided by the domain expert in the usage model to transform parametric resource demands to resource demands, guards on branches to probabilities, and parameter dependent loop iteration specifications into iteration numbers. Afterwards, the transformation into Capra is straightforward (Koziolek et. al (2007a)). The remainder of this section gives a brief overview on Capra's syntax, semantics, and analytical capabilities.

The basic entities of a process are actions. In Capra, the set of actions *Act* is divided into a set of event actions *EAct* and demand actions *DAct*. Event actions represent atomic, indivisible behavior which is mainly used for process communication. Demand actions on the other hand represent the usage of resources, like processors and hard disks, for a certain time span. During the execution of a demand action other activities might happen in the system. A random variable $D_{a,r}$ specifies the demand issued by a demand action *a* to a resource *r*. It is characterized by a probability density function (pdf) $f_{a,r}(t)$. To create more complex behavioral structures, Capra offers a set of different operators for sequential and alternative composition as well as for the specification of recursive process behavior. The following describes the available operators.

Process $P \cdot Q$ denotes the *sequential composition* of two processes *P* and *Q*. It first executes *P* and then, after the successful termination of *P*, it executes *Q*.

The *alternative composition* models the execution of only one of the two processes *P* and *Q*. Capra distinguishes two types of alternative composition, depending on where the decision for the next activity is made. If the process itself decides on the next activity, the internal or probabilistic choice is used. Process $P \oplus_p Q$ selects process *P* with probability π and process *Q* with probability $1 - \pi$. On the other hand, the external or non-deterministic choice $P + Q$ models different possible behaviors. Here, the selection is determined by event actions issued by the process' environment, i.e. other processes running in parallel.

A process variable *X* can be used to define *recursive processes*. For example, $Do := a \cdot Do$ specifies a process that executes an infinite number of *a* actions. In real systems, the number of recursive calls is usually bounded. Furthermore, the limit is usually not fixed to a single value, but depends on the system's parameters. To approximate such behavior, Capra models the number

of recursive calls as a discrete random variable specified by a probability mass function (pmf). Process $P^{*(Iter)}$ describes the *bounded loop* of a processes *P*. The random variable $Iter$ characterized by a pmf $P(Iter = n)$ denotes the probability of executing the recursion *n* times (Koziolek et. al., 2007c).

Process $P \mid_A Q$ denotes the *parallel composition* of the processes *P* and *Q*. The processes communicate (and synchronize) over the event actions in the set *A*. Both processes compete for the available resources, which might delay their execution.

To reduce the complexity of the simulation, the total demand of some operations can be determined in advance. If two processes issue only demands to a single resource, their total demand can be computed for the operations sequential composition, probabilistic choice, and finite recursion. The following gives an impression on the possible analyses.

The total demand of a sequence of demand actions is the sum of the single demands. So, the random variable for the sequential demand is given by $D_{P \cdot Q,r} = D_{P,r} + D_{Q,r}$. The sum of two random variables is the convolution of its pdfs. Thus, the pdf of $D_{P \cdot Q,r}$ is $f_{P \cdot Q,r}(t) = (f_{P,r} \otimes f_{Q,r})(t)$, where $\otimes$ denotes the convolution.

For the probabilistic choice $P \oplus_p Q$, the demand is either $D_{P,r}$ with probability π or $D_{Q,r}$ with probability $1 - \pi$. Thus, the pdf of $D_{P \oplus_p Q}$ is the weighted sum of the probability density functions:

$$f_{P \oplus_\pi Q,r}(t) = \pi \cdot f_{P,r}(t) + (1 - \pi) f_{Q,r}(t).$$

Finite recursion can be considered as large probabilistic choices over the n-time sequential composition of processes *P* and *Q*. The pmf $P(I = n)$ defines the probabilities for the probabilistic choice. Thus, function:

$$f_{\langle P[I]Q \rangle,r}(t) = (\sum p_I(i) \otimes^i_{j=1} (f_{P,r} \otimes f_{Q,r}))(t)$$

computes the pdf for demand $D_{\langle P[I]Q \rangle}$.

With such combined resource demands, the number of required simulation steps is reduced significantly. The simulation itself is an event-discrete simulation based on the Desmo-J framework and yields as result the response time of a usage scenario as a probability distribution. Details on Capra and the simulation can be found in Happe et al. (2007).

Queuing Network Simulation

Many performance analysis methods use queuing networks as underlying prediction models because of their capability to analyze concurrent system interactions. Queuing models contain a network of service centers with waiting queues which process jobs moving through the network. When applying queuing networks in performance predictions with the PCM, some of the commonly used assumptions need to be dropped. As the PCM uses arbitrary distribution functions for the random variables, generalized distributed service center service times, arrival rates, etc. occur in the model. Additionally, the requests are routed through the queuing network according to the control flow specified in the RDSEFF. In contrast, common queuing networks assume probabilistic movement of the jobs in the network. As a result, only simulation approaches exist, which solve such models.

Hence, we use a model-to-text transformation to generate Java code realizing a custom queuing network simulation based on the simulation framework Desmo-J. The simulation generates service centers and their queues for each active resource. Passive resources are mapped on semaphores initialized with the resource's capacity. The transformation generates Java classes for the components and their assembly. Service implementations reflect their respective SEFF.

For the usage model workload drivers for open or closed workloads simulating the behavior of

users exist in the generated code. For any call issued to the simulated system, the simulation determines the parameter characterizations of the input parameters and passes them in a so called virtual stackframe to the called service. Originally, the concept of a stackframe comes from compiler construction where they are used to pass parameters to method calls. In the PCM simulation, stackframes pass the parameter characterizations instead.

Utilizing the information in the simulated stackframes, the simulated SEFF issues resource demands to the simulated resources. If the resource is contented, the waiting time increases the processing time of the demand.

The simulation runs until simulation time reaches a predefined upper limit or until the width of the estimation for the confidence interval of the mean of any of the measured response times is smaller than a predefined width. After the end of a simulation run, the simulation result contains different performance indicators (response times, queue lengths, throughputs …) which the software architect can analyze to determine performance bottlenecks in the software architecture.

The computations described here reduce the complexity of Capra expressions allowing a more efficient and more accurate simulation. In the special case, that all demands are issued to the same resource and no concurrency is used, the whole expression can be solved analytically.

Java Code & Performance Prototype

The effort spent into creating a model of a software architecture should be preserved when implementing the system. For this, a model-2-text transformation based on the openArchitectureWare (oAW) framework generates code skeletons from PCM model instances. The implementation uses either Plain Old Java Objects (POJOs) or Enterprise Java Beans (EJBs) ready for deployment on a J2EE application server.

The transformation uses as much model information as possible for the generation of artifacts. Repository models result in components, assembly connectors in distributed method calls, the allocation is used to generate ant scripts to distribute the components to their host environment and finally, the usage model results in test drivers.

A particular challenge is the mapping of concepts available in the PCM to objects used in Java or EJB. Consider for example the mapping of composite components to Java. As there is no direct support of composed structures in Java, a common solution to encapsulate functionality is the application of the session façade design pattern.

Another issue with classes as implementing entities for components is the missing capabilities to explicitly specify required interfaces of classes in object oriented languages. A solution for this is the application of the component context pattern by Völter et al. (2006). This pattern moves the references to required services into a context object. This object is injected into the component either by an explicit method call or by a dependency injection mechanism offered by the application server.

Finally, we can combine the EJB and the simulation transformation. This way, users can generate a prototype implementation which can be readily deployed and tested on the final execution environment. Internal actions of the prototype only simulate resource demands by executing dummy code which offers quality characteristics as specified in the model. By using the prototype, early simulation results can be validated on the real target environment to validate early performance estimates.

EXAMPLE

To illustrate the performance prediction approach with the PCM, this section provides a

case study, in which we predicted the response time of a usage scenario in a component-based software architecture and compared the results with measured response times from executing an implementation.

The system under analysis is the "MediaStore" architecture, a web-based store for purchasing audio and video files, whose functionality is modeled after Apple's iTunes music store. It is a three-tier architecture assembled from a number of independently usable software components (Figure 9). Users interact with the store via web browsers, and may purchase and download different kinds of media files, which are stored in a database connected to the store's application server via Gigabit Ethernet.

We analysed a scenario, in which users purchase a music album (10-14 files, 2-12 MB per file) from the store. As a measure for copy protection, a component "DigitalWatermarking" shall be incorporated into the store. This component unrecognisable attaches the user's ID to the audio files via digital watermarking. In case the audio files would appear illegally in public file sharing services, this enables tracking down the responsible user. However, this copy protection measure has an influence on performance, as it decreases the response time of the store when downloading files. With the model-driven performance prediction, we analysed whether the store is capable of answering 90% of all user download requests in less than 8 seconds.

Each component in the store provides RDSEFFs to enable performance analyses (three examples in Figure 10). The execution time in this use case mainly depends on the number and size of the files selected for download, which influences network traffic as well as CPU utilisation for the watermarking algorithm. The specifications of the the components' RDSEFFs have been calibrated with measurements on the individual components. In this case, we carried out the predictions using Capra.

Besides modelling the store, we also implemented the architecture assisted by the introduced model-to-text transformations to Java code (EJB3). After generating code skeletons from the design, we manually added the implementation of the business logic of forwarding requests and watermarking audio files. The code

Figure 9. MediaStore architecture (©2007 Heiko Koziolek. Used with permission)

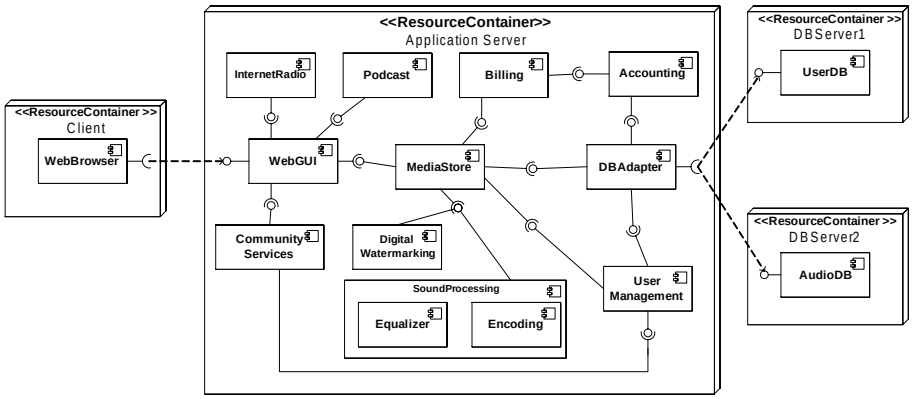


Figure 10. MediaStore service effect specifications (©2007 Heiko Koziolek. Used with permission)

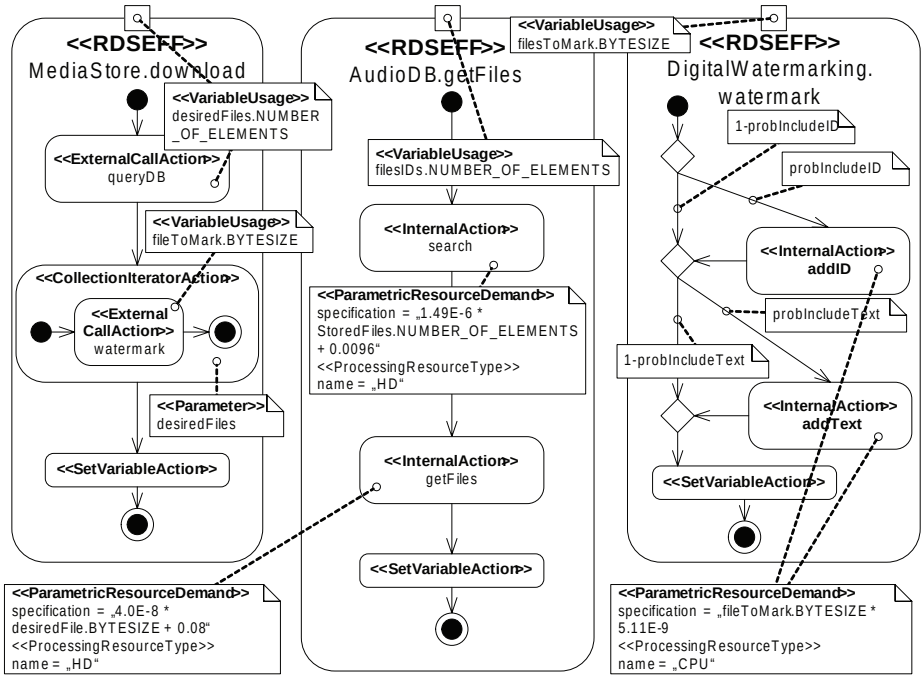
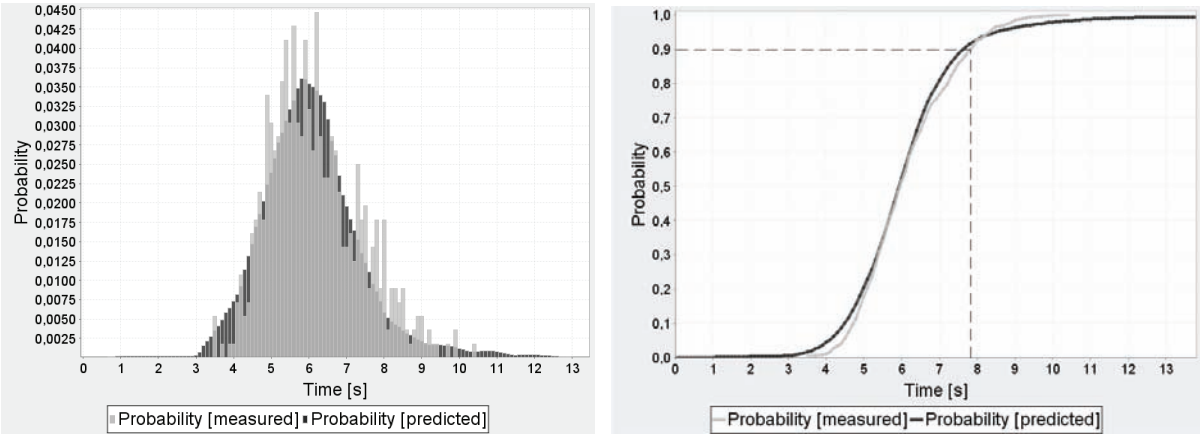


Figure 11. Case study results (©2007 Heiko Koziolek. Used with permission)



generation also creates build scripts, test drivers, deployment descriptors, and configuration files. We weaved measurement probes into the code using AspectJ.

The results of prediction and measurement are compared in Figure 11. The diagram on the left hand side visualises the histograms of the response times. The dark columns indicate the prediction, while the bright columns on top of the dark columns indicate the measurement. The highest probability of receiving a response from the store with the mentioned parameters is at around 6 second. In this case, the prediction and the measurement widely overlap.

The diagram on the right hand side visualises the cumulative distribution functions of the response time prediction and measurements. This illustration allows to easily check our constraint of at least 90% of all responses in less than 8 seconds. It was predicted that 90% of all requests would be responded in 7.8 seconds even if watermarking was used in the architecture. The measurements confirmed the predictions, because in our tests 90% of the request could be answered less than 7.8 seconds. There is a difference of 0.1 seconds or 1.3 percent.

In this case, the predictions were useful to quantitatively support the software architect's decision to introduce watermarking without violating a service level agreement. Note, that the predictions are not meant to be real-time predictions for safety-critical systems. They are useful at early development stages on the architectural level to support design decisions and lower the risk of performance problems in implementations. Safety-critical systems (e.g., airbag controls) instead need formal verifications of predictions to prevent harming human lives. That requires more fine grain specifications at lower abstraction levels, which developers can only create if most details of the system are known.

FUTURE RESEARCH DIRECTIONS

Model-driven performance prediction and quality assurance of software architecture models is still in its infancy and provides lots of opportunities for future research. Woodside et al. (2007) recently commented on the future of software performance engineering. We provide a list of future research directions from our viewpoint complementing their ideas:

- **Intermediate languages:** To bridge the gap between designer-friendly model notations and analytically-oriented formalisms, many approaches have developed ad-hoc model transformations. Several approaches aim at providing a standard interface, i.e., an intermediate modelling language, to ease the implementation of model transformations (Grassi et al. (2005), Petriu et. al. (2005))
- **Dynamic architectures:** The PCM is only targeted at static architectures, and does not allow the creation/deletion of components during runtime or changing links between components. With the advent of web services and systems with dynamic architectures changing during runtime, researchers pursuit methods to predict the dynamic performance properties of such systems (Caporuscio et al. (2007), Grassi et al. (2007)).
- **Layered resource models:** With OMG's MDA vision of platform independent models and platform specific models, it seems straight forward to follow this approach in performance modelling. For different system layers (e.g., component containers, middleware, virtual machine, operating system, hardware resources), individual models capturing performance-relevant properties could be built. These models could be com-

posed with architectural models to predict the performance (Woodside et. al (2007)).

- **Combination of modeling and measurement:** Developers can only carry out performance measurements if the system or at least parts of it have been implemented. Measurement results could be used to improve models. In component-based performance modelling, measurements are useful to deduce the resource demands of components. A convergence of early-life cycle modelling and late-life cycle measurement can potentially increase the value of performance evaluations (Woodside et. al (2007)).
- **Performance engineering knowledge database:** Information collected by using prediction models or measuring prototypes tends to get lost during system development. However, the information is useful for future maintenance and evolution of systems. Systematic storage of performance-related information in a knowledge database could improve performance engineering (Woodside et. al (2007)).
- **Improved automated feedback:** While today's model-transformations in software performance engineering bridge the semantic gap from the developer-oriented models to the analytical models, the opposite direction of interpreting performance result back from the analytical models to the developer-oriented models has received sparse attention. Analytical performance results tend to be hard to interpret by developers, who lack knowledge about the underlying formalisms. Thus, an intuitive feedback from the analytical models to the developer-oriented models would be appreciated (OMG (2005), Woodside et. al (2007)).

CONCLUSION

This chapter provided an overview of the Palladio Component Model, a modelling language to describe component-based software architectures aiming at early life cycle performance predictions. The PCM is aligned with developer roles in CBSE, namely component developers, software architects, system deployers, and domain experts. Therefore, the PCM provides a domain specific modelling language for each of these developer roles. Combining the models from the roles leads to a full PCM instance specification, which can be transformed to different analysis models or Java code. An hybrid analysis model (SPA) provides a fast way to predict response times. Simulation of PCM instances is potentially more time-consuming, but offers more expressiveness than the hybrid approach. Finally, developers may use generated Java code skeletons from a PCM instance as a starting point for implementation. To illustrate the PCM's capabilities the chapter included a case study predicting the performance for a small component-based architecture.

The PCM is useful both for component developers and software architects. Component developers can specify the performance of their components in a context-independent way, thereby enabling third party performance predictions and improving reusability. Software architects can retrieve component performance specification from repositories and assemble them to architectures. With the specifications they can quickly analyse the expected performance of their designs without writing code. This lowers the risk of performance problems in implemented architectures, which are a result of a poor architectural design. The approach potentially saves large amounts of money because of avoided re-designs and re-implementations.

The chapter provided pointers for future directions of the discipline in Section 7. Future work

for the PCM includes improving the resource model, supporting dynamic architectures and reverse engineering. The resource model needs to be improved to support different scheduling disciplines, concurrency patterns, middleware parameters, operating system features etc. Dynamic architectures complicate the model as they allow changing links between components and allow the creation and deletion of components during runtime. However, this is common in modern service-based systems, and thus should be incorporated into performance predictions. Finally, reverse engineering to semi-automatically deduce performance models from existing legacy code seems an interesting pointer for future research. Reducing the effort for modelling would convince more developers of applying performance predictions. The inclusion of legacy systems enables predicting the impact on performance of planned system changes.

REFERENCES

Balsamo, S. , DiMarco, A., Inverardi, P. & Simeoni, M. (2004). Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering, 30*(5), 295-310.

Becker, S.; Grunske, L.; Mirandola, R. & Overhage, S. (2005). Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective. In *Springer Lecture Notes in Computer Science Vol. 3938* (pp. 169-192).

Becker, S., Koziolek, H. & Reussner, R. (2007). Model-based Performance Prediction with the Palladio Component Model. In *Proceedings of the 6th Workshop on Software and Performance WOSP'07* (pp. 56-67). ACM Press

Bertolino, A. & Mirandola, R. (2004). CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In Crnkovic, I., Stafford, J. A., Schmidt, H. W. & Wallnau, K. C. (Ed.), *Proceedings of the 7th International Symposium on Component-Based Software Engineering, CBSE2004* (pp. 233-248). Springer Lecture Notes in Computer Science, Vol. 3054

Bondarev, E., de With, P., Chaudron, M. & Musken, J. (2005). Modelling of Input-Parameter Dependency for Performance Predictions of Component-Based Embedded Systems. In *Proceedings of the 31th EUROMICRO Conference (EUROMICRO'05)*

Caporuscio, M., DiMarco, A. & Inverardi, P. (2007), Model-based system reconfiguration for dynamic performance management. *Journal of Systems and Software, 80*(4), (pp. 455-473). Elsevier

Eskenazi, E., Fioukov, A. & Hammer, D. (2004). Performance Prediction for Component Compositions. In Crnkovic, I., Stafford, J. A., Schmidt, H. W. & Wallnau, K. C. (Ed.), *Proceedings of the 7th International Symposium on Component-Based Software Engineering, CBSE2004*. Springer Lecture Notes in Computer Science, Vol. 3054 Grassi, V., Mirandola, R. & Sabetta, A. (2005). From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *Proceedings of the 5th international workshop on Software and performance, WOSP '05* (pp. 25-36). ACM Press

Grassi, V., Mirandola, R. & Sabetta, A. (2007). A Model-Driven Approach to Performability Analysis of Dynamically Reconfigurable Component-Based Systems. In *Proceedings of the 6th international workshop on Software and performance, WOSP '07* (pp. 142-153). ACM Press

Happe, J., Koziolek, H., & Reussner, R. H. (2007). Parametric Performance Contracts for Software Components with Concurrent Behaviour. In *Electronical Notes of Theoretical Computer Science*, Vol. 182 (pp. 91-106), Elsevier.

Hamlet, D., Mason, D. & Woit, D. (2004). Properties of Software Systems Synthesized from Components. In Lau, K. (Ed.), *Component-Based Software Development: Case Studies* (pp. 129-159). World Scientific Publishing Company

Hissam, S. A., Moreno, G. A., Stafford, J. A. & Wallnau, K. C. (2002). Packaging Predictable Assembly. In *CD'02: Proceedings of the IFIP/ACM Working Conference on Component Deployment* (pp. 108-124). Springer-Verlag

Koziolek, H., Happe, J. & Becker, S. (2006). Parameter Dependent Performance Specifications of Software Components. In Hofmeister, C., Crnkovic, I., Reussner, R. & Becker, S. (Ed.) *Proceedings of the 2nd International Conference on the Quality of Software Architecture, QoSA2006* (pp. 163-179). Springer Lecture Notes in Computer Science, Vol. 4214

Koziolek, H., Happe, J. & Becker, S. (2007). Predicting the Performance of Component-based Software Architectures with different Usage Profiles. In Szyperski, C. & Overhage, S. (Ed.) *Proceedings of the 3rd International Conference on the Quality of Software Architecture, QoSA2007.* Springer Lecture Notes in Computer Science, To Appear

Koziolek, H. & Firus, V. (2007). Parametric Performance Contracts: Non-Markovian Loop Modelling and an Experimental Evaluation. In *Electronical Notes of Theoretical Computer Science*, Vol. 176 (pp. 69-87), Elsevier

OMG: Object Management Group (2005). UML Profile for Schedulability, Performance and Time. *http://www.omg.org/cgi-bin/doc?formal/2005-01-02*

Petriu, D. B. & Woodside, M. (2005). An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Springer Journal on Software and Systems Modeling*

Reussner. R. H., Becker, S., Happe, J., Koziolek, H., Krogmann, K. & Kuperberg. M. (2007). *The Palladio Component Model*. Internal Report Universität Karlsruhe (TH)

Sitaraman, M., Kuczycki, G., Krone, J., Ogden, W.F. & Reddy, A. (2001). Performance Specifications of Software Components. In *Proceedings of the Symposium on Software Reusability 2001* (pp. 3-10).

Szyperski, C., Gruntz, D. & Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley

Wu, X. & Woodside, M. (2004). Performance modeling from software components. In *Proceedings of the 4th International Workshop on Software Performance, WOSP2004* (pp. 290-301). ACM SIGSOFT Software Engineering Notes

Völter, M. & Stahl, M. (2006). *Model-driven Software Development.* Wiley & Sons

Woodside, M., Franks, G. & Petriu D. C. (2007). The Future of Software Performance Engineering. In *Proceedings of 29th International. Conference on Software Engineering, ICSE'07.* Track: Future of Software Engineering.

ADDITIONAL READING

Bolch, G., Greiner, S., de Meer, H. & Trivedi, K. S. (2006). *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* Wiley-Interscience, 2nd Edition

Cecchet, E., Marguerite, J. & Zwaenepoel, W.(2002) Performance and scalability of EJB applications. *ACM SIGPLAN Notes, 37*(11), 246-261

Chen, S., Liu, Y., Gorton, I. & Liu, A. (2005). Performance prediction of component-based

applications. *Journal of Systems and Software, 74*(1), 35-43.

DiMarco, A. & Mirandola, R. (2006). Model transformations in Software Performance Engineering. *Springer Lecture Notes in Computer Science,* Vol. 4214, 95-110

Dumke, R., Rautenstrauch, C., Schmietendorf, A. & Scholz, A. (2001). *Performance Engineering: State of the Art and Current Trends.* Springer Lecture Notes in Computer Science, Vol. 2047

Grassi, V., Mirandola, R. & Sabetta, A. (2006). Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software, 80*(4), 528-558.

Hermanns, H., Herzog, U. & Katoen, J. (2002) Process algebra for performance evaluation. *Theorectical Computer Science, 274*(1-2), Elsevier Science Publishers Ltd., 43-87

Jain, R. K. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley

Kounev, S. (2006). Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering, 32*(7), 486-502.

Lazowska, E.; Zahorjan, J.; Graham, G. & Sevcik, K. (1984). *Quantitative System Performance,* Prentice Hall

Liu, Y., Fekete, A. & Gorton, I. (2005). Design-Level Performance Prediction of Component-Based Applications. *IEEE Transactions on Software Engineering, 31*(11), 928-941.

Menasce, D. A. & Gomaa, H. (2000). A Method for Design and Performance Modeling of Client/Server Systems. *IEEE Transactions on Software Engineering, 26*(11), 1066-1085

Menasce, D. A. & Almeida, V. A.(2000) *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning,* Prentice Hall

Menasce, D. A. & Almeida, V. A.(2002) *Capacity Planning for Web Services,* Prentice Hall

Menasce, D. A., Dowdy, L. W. & Almeida, A.F. (2004). *Performance by Design: Computer Capacity Planning By Example,* Prentice Hall PTR

Reussner, R. H., Schmidt, H. W. & Poernomo, I. H. (2003). Reliability prediction for component-based software architectures. *Journal of Systems and Software, 66*(3), 241-252.

Rolia, J. A. & Sevcik, K. C. (1995). The Method of Layers. *IEEE Transactions on Software Engineering, 21*(8), 689-700

Smith, C. U. & Williams, L. G. (2001). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley Professional

Verdickt, T., Dhoedt, B., Gielen, F. & Demeester, P (2005). Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE Transactions on Software Engineering, 31*(8), 695-771.

Woodside, C. M., Neilson, J. E., Petriu, D. C. & Majumdar, S. (1995) The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Transactions on Computers,* 44(1), 20-34.

This work was previously published in Model-Driven Software Development: Integrating Quality Assurance, edited by J. Rech & C. Bunse, pp. 95-118, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Section III
Tools and Technologies

This section presents extensive coverage of the technology that both derives from and informs software applications. These chapters provide an in-depth analysis of the use and development of innumerable devices and tools, while also providing insight into new and upcoming technologies, theories, and instruments that will soon be commonplace. Within these rigorously researched chapters, readers are presented with examples of the tools that facilitate and support software design and engineering. In addition, the successful implementation and resulting impact of these various tools and technologies are discussed within this collection of chapters.

Chapter 3.1
Knowledge Management Toolkit for SMEs

Kerstin Fink
University of Innsbruck, Austria

Christian Ploder
University of Innsbruck, Austria

ABSTRACT

The discipline of knowledge management is no longer emerging in large organizations, but also small and medium-sized enterprises (SMEs) are focusing on finding the right process that will allow them to make advantages of their intellectual capital. Using survey data from 219 small and medium-sized enterprises in Austria and Switzerland, this chapter illustrates the four key knowledge processes (1) knowledge identification, (2) knowledge acquisition, (3) knowledge distribution, and (4) knowledge preservation for SMEs and also reports the findings of the empirical study designed to allocate cost-efficient software products to each of the four knowledge processes. As a result a knowledge toolkit for SMEs that integrates knowledge processes, methods and software tool for decision support making is given. Finally, the social view of knowledge management to SMEs is discussed, showing that the use of information technology is currently far more important than the integration of a social-cognitive perspective.

INTRODUCTION

The academic literature on knowledge management has become a major research field in different disciplines in the last ten years (Nonaka & Takeuchi, 1995; Ruggels, 1997; Sveiby, 1997; Davenport & Prusak, 1998; Back, Enkel, & Krogh, 2007). Through knowledge management, organizations are enabled to create, identify and renew the company's knowledge base and to deliver innovative products and services to the customer. Knowledge management is a process of systematically managed and leveraged knowledge in an organization. In a global and interconnected society, it is more difficult for companies to know where the best and most valuable knowledge is.

The term knowledge has a wide range of definitions in the knowledge management literature. The authors follow the definition by Davenport and Prusak (1998, p. 5) "knowledge is a fluid mix of framed experiences, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the mind of knowers." For a better understanding of knowledge management Jennex (2007, p. 4) points out that the concepts of organizational learning and memory should be integrated. Therefore, knowledge management can be defined as "the practice of selecting applying knowledge from pervious experiences of decision making to current and future decision-making activities with the express purpose of improving the organization´s effectiveness" (Jennex, 2007, p. 6).

Knowledge management is more than the technological solutions provided to give people access to better and more relevant information (Wang & Plaskoff, 2002, p. 113). It is important that the design of the knowledge management systems reflect the mindset of the knowledge workers and their way of offering highly qualitative knowledge solutions with quick solution processes. An effective knowledge management system must integrate people, processes, technology and the organizational structure.

Historically, knowledge management focused on the domain of larger organizations and issues of culture, networking, organizational structure and technological infrastructure are applied upon the implementation of knowledge management initiatives in large multi-national organizations and seem to give little relevance (Delahaye, 2003) to small and medium-sized enterprises (SMEs). SMEs are playing a key role in European economic performance because they account for a high proportion of the gross domestic product (GDP) and employ some two- thirds of the European workforce. According to the OECD (Organisation for Economic Co-operation and Development) *Small and Medium-sized Enterprise Outlook 2002 and 2005* (OECD, 2005), the role of SMEs in OECD economic is very important for strengthening economic performances. SMEs represent over 95% of enterprises in most OECD countries, and generate over half of private sector development. A similar impact of SMEs to economic value can be found in the report of the Asia-Pacific Economic Cooperation (APEC, 2006), where about 90% of enterprises are SMEs. During their 2006 meeting in Beijing, the members agreed to strengthen the SME`s competitiveness for trade and investment. For example, SMEs account for more than 95% of companies in Australia. Of the 624.010 SMEs in Australia, more than two-thirds employ between one and four people. A further 180,880 SMEs employ between five and 19 people meaning that 93.5% of people employed by SMEs in Australia are employed by what can be described as 'micro-SMEs,' namely companies with less than 20 employees. However, the success and growth of SMEs depends on how well they manage the knowledge of their knowledge workers. In 2000, the European Council set the clear strategic goal for the European Union (EU) of becoming "the most competitive and dynamic economy in the world, capable of sustaining economic growth with more and better jobs and greater social cohesion" by the year 2010 (EC, 2000). Dezouza and Awazu (2006) point out that SMEs have to compete on the know-how in order to gain competitive advantages. Even more, SMEs do not have much money to spend on knowledge management initiatives, so knowledge must be leveraged that goals can be achieved in an effective and efficient manner (Fink & Ploder, 2007c). Ordanini (2006) discusses the issue that the adoption of information technology by SMEs began to be discussed during the 1980s. Furthermore, it has to be stated that the adoption of information technology for SMEs was slower than that of larger organizations, which can be referred to as the so-called digital divide phenomenon.

Looking to the European countries of Austria and Switzerland including Liechtenstein, a similar

landscape of SMEs can be identified. According to the *Austrian Statistical Year Book* (SA, 2006) and the Austrian Institute for SMEs Research (ASME, 2006) for the year 2006, 99.7% which are 297,800 companies are SMEs in Austria. In Switzerland, also 99.7% of the companies are SMEs, looking at the data from CHSME (2006). There are several research articles dealing with knowledge management in SMEs (Sveiby, 1997; Beijerse, 2000; McAdam & Reid, 2001; Salojärvi, Furu, & Sveiby, 2005), but only a few empirical studies are conducted to see the impact of knowledge processes in SMEs. McAdam and Reid (2001) found out that the time is right for knowledge management within the SME-sector. The results of their comparative study of large organizations and SMEs showed that both sectors have much to gain by the development of knowledge management systems. Salojärvi, Furu and Sveiby (2005) have observed that SMEs should be able to enhance their performance and competitive advantages by a more conscious and systematic approach to knowledge management.

In this chapter, the focus lies on the impact of knowledge process modeling for SMEs to help them getting a framework to be more innovative (Donnellan, Conboy, & Hill, 2006). In research methodology, the theoretical framework for the identification of knowledge processes in SMEs will be discussed. The section that follows that covers the use of cost-efficient software products for the implementation of knowledge processes in SME and introduces a SME knowledge toolkit. Discussion and future research gives an outlook of future research and a discussion of social factors influencing knowledge management in SMEs.

RESEARCH METHODOLOGY

Definition of SMEs

There are several quantitative and qualitative definitions of the term small and medium-sized enterprise (SME) depending on regional and national differences. In the United States, the definition of small business is set by a government department called the Small Business Administration (SBA) Size Standards Office. The SBA uses the term "size standards" which is a numerical definition to be considered as a small or medium-sized business. It must also be independently owned and operated. Unlike the European Union, which has simple definitions applied to all industries, the United States has chosen to set size standards for each individual industry. This distinction is intended to better reflect industry differences. SMEs are also of high importance for in the U.S. Economy. Similar to Europe, more than 97% of the firms in the U.S. can be defined as SMEs.

The definition of SMEs of the European Commission 2005 (EC, 2000) is used for this research design. The European Commission analyzes SMEs by using the following three characteristics: (1) number of employees, (2) annual revenue and (3) total assets. Characterized through these three factors, the European Commission differs: (1) middle enterprises (less than 250 employees and less than 50 million euro annual revenue or less than 43 million total assets); (2) small enterprises (less than 50 employees and less than 10 million euro annual revenue or less than 10 million euro total assets); and (3) micro enterprises (less than 10 employees and less than 2 million euro annual revenue or less than 2 million euro total assets).

Since the authors focus on the definition of the European Commission of SMEs, they follow the research view of a quantitative perspective of SMEs. This means that all enterprises with less than 250 employees and less than 50 million euro annual revenue or less than 43 million euro total assets in Austria and Switzerland including Liechtenstein are the target population.

Research Framework

The basic research model is the "building block" approach by Probst, Raub and Romhardt (2002)

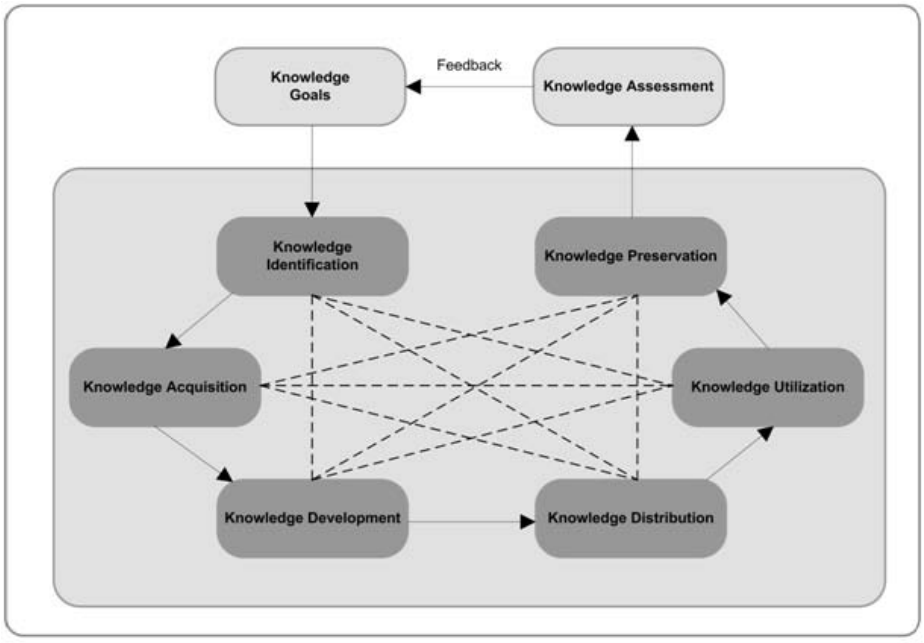
with the description of the knowledge processes (Figure 1). Involved are eight components that form two cycles: an inner cycle and an outer cycle. The inner cycle is composed of six key knowledge processes:

- *Knowledge identification* is the process of identifying external knowledge for analyzing and describing the company's knowledge environment.
- *Knowledge acquisition* refers to what forms of expertise the company should acquire from outside through relationships with customers, suppliers, competitors and partners in co-operative ventures.
- *Knowledge development* is a building block which complements knowledge acquisition. It focuses on generating new skills, new products, better ideas and more efficient processes. Knowledge development includes all management actions consciously aimed at producing capabilities.
- *Knowledge distribution* is the process of sharing and spreading knowledge which is already present within the organization.
- *Knowledge utilization* consists of carrying out activities to ensure that the knowledge present in the organization is applied productively for its benefit.
- *Knowledge preservation* is the process where the selective retention of information, documents and experience required by management takes place.

In addition, there are two other processes in the outer cycle, knowledge assessment and knowledge goals, which provide the direction to the whole knowledge management cycle:

- *Knowledge assessment* completes the cycle, providing the essential data for strategic control of knowledge management.
- *Knowledge goals* determine which capabilities should be built on which level.

Figure 1. Knowledge building block approach by Probst, Raub and Romhardt (2002)



Among other knowledge process models (Nonaka & Takeuchi, 1995; Laudon & Laudon, 2006; Jennex, 2007), the building block approach of Probst, Raub and Romhardt (2002) has the advantage that it is well known in European companies as well as in SMEs and furthermore it is a very unique and complete design of knowledge processes. Business process modeling (Hammer & Champy, 1993) has become a major research field in the information systems discipline in the last ten years. Davenport sees the term business process as 'a structured, measured set of activities designed to produce a specified output from a particular customer or market' (Davenport, 1995). The linkage of business process modeling and knowledge management is called knowledge process modeling. Richter-von Hagen et al. describes knowledge-intensive processes as sequences of activities based on knowledge intensive acquisitions and handling (Richter-von Hagen, Ratz, & Povalej, 2005). Edwards and Kidd (Edwards & Kidd, 2003, p. 124) named the following five characteristics to enforce the argument that knowledge management and business process management should be integrated:

- Knowledge management is important for business if the initiative implied an advantage for the customers. The idea to implement the customer's requests—may be internal or external—is the base for including the customer (Fink, Roithmayr, & Ploder, 2006).
- Knowledge does not follow the business borders. Business processes also model activities by global trading companies and build the base for modeling knowledge intensive processes.
- Knowledge management can only be efficient if it follows a structured model. Business processes are modeled by structured actions and they are necessary to deduce knowledge-intensive processes.
- The success of knowledge management depends on the measurement of knowledge. There exists a similarity to the measurement of business processes. The measurement of the knowledge potential provides a central position and biases the success (Fink, 2004).
- Knowledge management is affected by a holistic approach. Every part of the business process modeling is important for success but every aspect should be considered.

Data Collection for Knowledge Processes

At the beginning of our research was the identification of knowledge processes for SMEs. Therefore in the first step, the authors conducted a study with the objective to find the key knowledge processes—based on the framework of (Probst et al., 2002)—for SMEs. In a second step, the authors conducted empirical studies in order to identify which software products support the identified knowledge processes and can be used in practice. The interviews were conducted at the Department of Information Systems (University of Innsbruck) in 2005. The key objective of the expert interview was to analyze which of the eight knowledge processes from Probst et al. (2002) are relevant for SMEs. The data sample was 20 expert interviews which were conducted by the authors in summer 2005. Ten experts from science and ten experts from practice were asked about the most important knowledge processes in SMEs. The result of these expert interviews showed that SMEs are only determined by three key knowledge processes: (1) knowledge identification, (2) knowledge acquisition, and (3) knowledge distribution. In addition, for SME the concept of (4) knowledge disposal played a key role during the expert interviews and was extended as the fourth process called "knowledge preservation" indicating the disposal as well as actualization of knowledge. This knowledge process model for SMEs was the first step in the research work and the general research framework. The second step

was to find out which process can be supported by which knowledge method. The research method was a literature review for the identification of knowledge methods (Coakes & Clarke, 2006; Smolnik, Kremer, & Kolbe, 2007) in order to build a method repository for each knowledge process.

Figure 2 describes the four key knowledge processes with the corresponding method repositories for SMEs (Fink & Ploder, 2006; 2007a) illustrating the validated research framework through interviews by the experts.

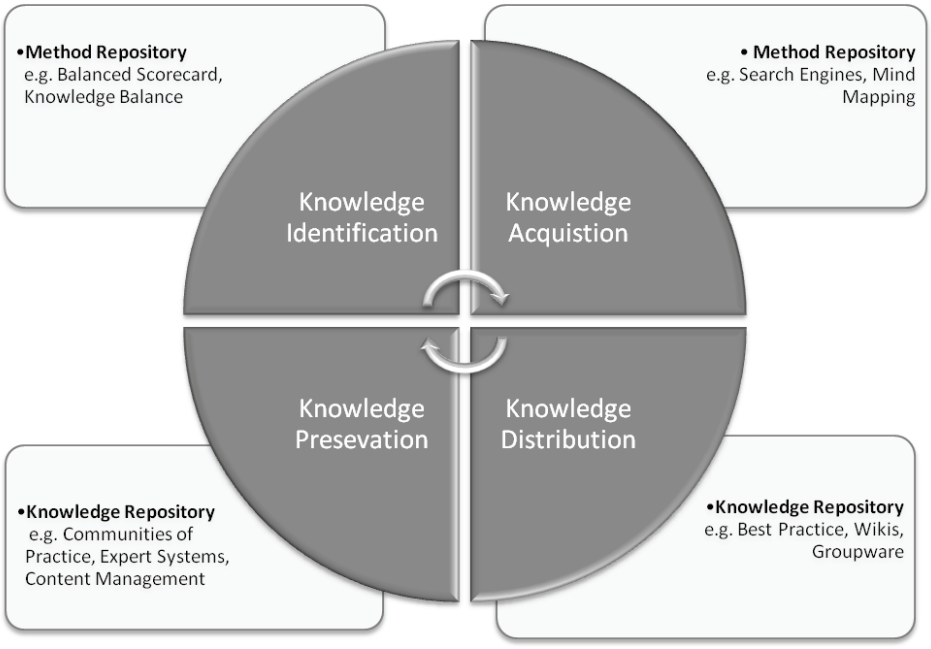
Quality Model (ISO/IEC 9126) for the Evaluation of Software Products

In a third step, the objective was to match a cost-efficient software product to each knowledge method which are usable in practice. In the research design the focus lies on Freeware and Shareware software products in order to fulfil the pre-setting of cost-efficient software support. The research method was an online research with the result of a list of evaluated cost-efficient software products. The evaluation of each software product was conducted by applying the ISO/IEC 9126 norm. The quality model of the norm (ISO/IEC 9126-1) is divided into two parts which are important for the evaluation of the software products to support knowledge methods:

- the internal and the external quality of the software
- the quality for use

The ISO norm lists five characteristics to evaluate software products: (1) functionality, (2) reliability, (3) usability, (4) efficiency and (5) assign ability. For each characteristic, a different number of items were assessed by a Likert scale from -2 to +2. The data sample of the quality model was

Figure 2. Theoretical concept of knowledge process model and method repositories for SMEs



more than 200 different software products. A key research finding was that some of the software products cannot be used in practice because their quality was not sufficient. Finally there were 45 software products which are efficient for use in SME.

Data Collection for Cost-Efficient Software Products

The 2006 survey was developed and executed by the authors and is an update and extension of the empirical study conducted in 2004-2005. Balmisse et al. (2007) present a knowledge management tool review including the vendor perspective. The article focuses on eight tool clusters. In comparison to our research, the authors focus primarily on the tool with the corresponding vendor side, while our research framework concentrates on the methods used in SMEs with a cost-efficient software support. So the objective of the empirical study was to find out which cost-efficient software products can support the efficient methods of the four knowledge processes. For this approach the authors differ between two categories of software. At the one hand side there are the standard software products which are already in use in SMEs (for example: MS Office, Internet Explorer, Operating System, etc.) and on the other hand there are software products like Shareware, Freeware and Open Source products characterized by the issue of cost-efficient software installation in SME. This described issue is the key objective of the empirical study. The research method for this study was the online questionnaire technique. The questionnaire was built with HTML, PHP and based on a MySQL database.

The data sample of 537 SMEs were the average allocated over the regional federal states of Austria, Switzerland and Liechtenstein to get a representative result for the whole sector and were opted stochastically. The total number of SMEs was 540,000. The online questionnaire was carried out in summer 2006 after a successful pre-test with 20 respondents. The online questionnaire was partitioned into three parts:

- Generally questions referring to the IT support and application of knowledge management within the enterprise itself
- Rating relevance of the methods concerning the four knowledge processes for SME and get an idea of the favor supporting software tool
- Information about future capital investment plans referring to knowledge management

The return quote of the survey was 40%. This means that 220 SME filled out the questionnaire. The failure rate was calculated as 6.63%. So, all statements out of the survey are correct at a percentage of 93.61%. In the following section the research findings of the methods and the supporting software tools are presented and discussed. The knowledge toolkit for SMEs integrates processes, methods and software tools in a decision support framework.

RESULTS

The distribution of industries can be described as follows. Thirty percent of the SMEs were from the industry sector and 22% from the consulting and information technology sector. The retail sector was represented by 13% and the trade and handcraft sector by 19%. The rest of 21% were divided to banks and insurances with 9%, transportation 2% and Tourism 10%. Fifty-seven percent of the SMEs already use knowledge management and 85% of the SMEs use a connected infrastructure. A Web space is (hosted intern or extern) available in 78% which is necessary to deal with software products which need such an IT-infrastructure.

Mapping of Cost-Efficient Software Products with Knowledge Processes

Table 1 gives an overview of all methods supporting the four knowledge processes for SMEs and the corresponding cost-efficient software products (Fink & Ploder, 2007c). The ranking of each method is the calculated value based on the Likert scale ("absolutely adequate": +2 pt. – "not adequate": -2 pt.). The "ISO ranking" illustrates the assessment of the software based on the quality model of the ISO/IEC 9126 norm ("absolutely appropriate": +2 – "not appropriate´: -2). For example, the Software Tools Gama for supporting business games is ranked with 15 points in the ISO/IEC 9126 and 75 of the data sample find Gamma is a good tool to support business games. The absolute frequency of naming of the software through the respondents can be seen in the last column (ranking survey). The naming of "no cost-efficient software support" indicates that only high-priced software products are available on the market.

Knowledge balance (92) was ranked highest among the methods for the first process of the *identification of knowledge*. Fifty-six percent of SMEs think that this is the best method. Further methods are the balanced scorecard (89) and the Skandia navigator (74). The methods market-asset value-method (-5) and Tobin`s q (-15) were rated by less than 30% to be of good use in SMEs.

Brainstorming (225) and knowledge network (203) are popular methods for the *knowledge acquisition*. Mind mapping (195), e-mail systems (134), scenario technique (126) and system simulation (98) are also suitable methods for this knowledge process, while business games (91) are also a possibility. The method of "Synektik" was rated very low because of its complexity. The favorite for the knowledge acquisition is the search engine (232) with over 70% for efficient use in SMEs. In this case, the Google desktop search engine was the prior selection software. Sixty percent of the respondents chose e-mail systems which can be supported, for example, by the software Thunderbird1.5. For brainstorming a practical tool is Concept X7, while for mind mapping the tools Free Mind (42%) and Think Graph (41%) were rated highly. As support software to a business game, 64% rated Gamma as a possible software tool.

As illustrated in Table 1 the methods e-mail system (185), handbook FAQs (159), communities of practice (152), groupware (139), questionnaire (110) and best practice (108) are the favorites for the k*nowledge distribution*. It has to be pointed out, that the methods micro article (2) and chat room (29) are rated not as well in the survey. The software products for the methods of transferring knowledge are InfoRapid supporting knowledge maps, EasySurvey supporting questionnaire, Skype and MSN supporting chat room, eGroupware 1.2 and AlphaAgent 1.6.0 supporting groupware, CUCards 2000 supporting checklists and Pegasus Mail, Thunderbird 1.5 and Amicron Mailoffice 2.0 for the support of e-mail systems.

Databases (242) are a recognized method of *the knowledge preservation process*. Eighty percent of the SMEs think that they will organize their knowledge with databases. Mind mapping (200), document management system (195) and checklists (164) are further efficient methods. Content management systems (126), project review (122), expert systems (74) and conceptualization (40) are methods which can be chosen but are not the prior choice. Neural network (-10) is not an adequate method for preserving knowledge. There were many different software products to support this process. MySQL is the favorite database software followed by the MSDE from Microsoft. Document management can be realized by Office Manager, UDEX dotNETContact or QVTutto.

Table 1. Ranking of methods and cost-efficient software products (Fink & Ploder, 2007a)

	Ranking	Supporting cost-efficient software products	ISO Ranking	Ranking Survey
Knowledge Identification				
Knowledge Balance	92	no cost-efficient software product, Office similar products		
Balanced Scorecard	89	no cost-efficient software product, Office similar products		
Skandia Navigator	74	no cost-efficient software product, commercial Software		
Market - Asset Value - Method	-5	no cost-efficient software product, Office similar products		
Tobin's q	-15	no cost-efficient software product, Office similar products		
Knowledge Acquisition				
Search Engine	232	Google Desktop Search; MSN Toolbar; Yahoo Desktop Search	not possible	25; 12; 10
Brainstorming	225	Brainstorming Toolbox; Concept X7	6;17	44; 88
Knowledge Network	203	no cost-efficient software product		
Mind Mapping	195	Free Mind; Think Graph, Tee Tree Office	16; 12; 8	69; 53; 28
E-mail System	134	Pegasus Mail; Thunderbird Mail; Amicron Mailoffice 2.0	21; 21; 12	63; 165; 26
Scenario Technique	126	no cost-efficient software product, Office similar products		
System Simulation	98	no cost-efficient software product, commercial Software		
Business Game	91	Gamma	15	75
Synektik	-17	no cost-efficient software product, commercial Software		
Knowledge Distribution				
E-mail System	185	Pegasus Mail; Thunderbird Mail; Amicron Mailoffice 2.0	16; 12; 8	63; 165; 26
Handbook FAQs	159	no cost-efficient software product, Office similar products		
Communities of Practice	152	no cost-efficient software product, Office similar products		
Groupware	139	eGRoupware1.2; AlphaAgent 1.6.0; Tiki CMS - Groupware	15; 14; 16	40; 26; 24
Questionnaire	110	Easy Survey	10	61
Best Practice	108	no cost-efficient software product, Office similar products		
Checklist	103	CUEcards 2000	8	128

continued on following page

Table 1. continued

Lessons Learned	103	no cost-efficient software product, Office similar products		
Knowledge Maps	82	InfoRapid KnowledgeMap	13	69
Story Telling	42	no cost-efficient software product, Office similar products		
Chat room	29	Skype; MSN, ICQ	not possible	71; 33; 25
Micro article	2	no cost-efficient software product, Office similar products		
Knowledge Preservation				
Database	242	MySQL; MSDE		86; 44
Mind Mapping	200	Free Mind; Think Graph, Tee Tree Office	16; 12; 8	69; 53; 28
Document Management System	195	Office Manager; UDEX dotNETContact; QVTutto	15; 15; 14	74; 35; 22
Checklist	164	CUEcards 2000	8	128
Content Management	126	CONTEX; ContentKit; VIO MATRIX	16; 13; 13	0; 47; 13
Project Review	122	no cost-efficient software product, Office similar products		
Experts System	74	KnowIT; KnowME	10; 7	38; 52
Conceptualization	40	no cost-efficient software product		
Neural Network	-10	no cost-efficient software product, commercial Software		

Investment Allocation

Seventy-five percent of the respondents assumed that they are still using knowledge management in their SME. As shown in Figure 3, the attendance to invest into knowledge management in the next year exists with 60% and up to 40% cannot imagine to invest into knowledge management in the next year. Twenty percent of SMEs are planning an investment in the future and want to spend less than 500 euro, 58% will spend between 500 euro and 3,000 euro and only 22%will invest more than 3,000 euro.

SME Knowledge Toolkit

The SME knowledge toolkit (Figure 4) is one of the results of our empirical research. Tiwana (2002) discusses in her book *The Knowledge Management Toolkit* a 10-step roadmap for implementing knowledge management in a company. The key objective of the toolkit is to guide a company through the complex process of analyzing the infrastructure, designing the knowledge system and linking the business strategy to knowledge management and make a performance evaluation. Jashapara (2004, p. 92) uses the term knowledge management suite that should offer a multitude of knowledge management systems in order to build an individualized toolset. The toolkit developed by the authors is an implementation for portraying knowledge processes, methods and software tools for SMEs. Therefore, the use of the toolkit can reveal the gap between the defined knowledge processes and implemented knowledge methods that could trigger further evidence to use different knowledge methods or to rearrange the knowledge processes. The explicit identification of knowledge processes allows the user of the toolkit to match knowledge methods with cost-efficient software products that fit into the culture of SMEs. At the

Figure 3. Investment allocation in SMEs for knowledge initiatives

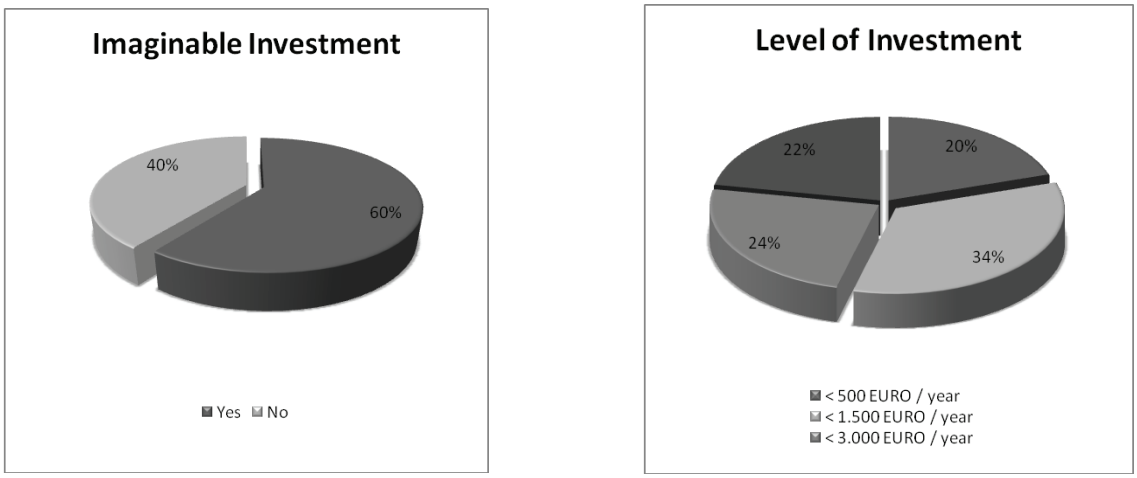
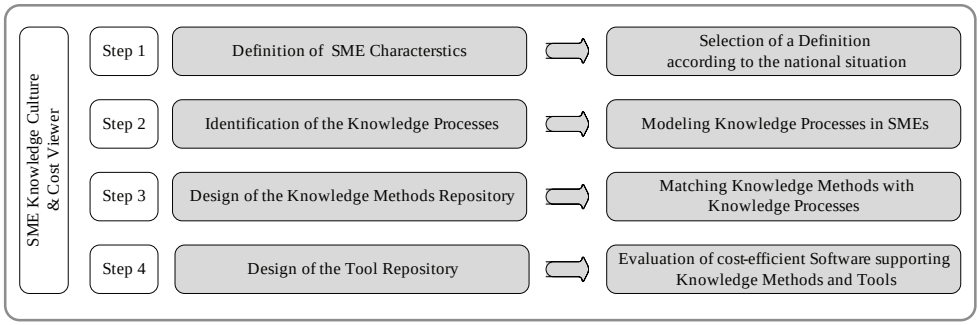


Figure 4. SME knowledge toolkit



same time, the knowledge toolkit calculates the estimated costs for changes. The SME knowledge toolkit (Figure 4) supports all aspects of knowledge management relevant for SMEs and is divided into four key steps:

Step 1: Definition of the characteristics by applying the national limitations and definitions
Step 2: Identification of the knowledge processes and modeling of these processes in the SME context
Step 3: Designing a knowledge method repository which enables SMEs to match the methods with the supporting knowledge processes
Step 4: Designing of a software repository which helps to evaluate cost-efficient software products for knowledge management in SMEs

These four key steps are supported by the knowledge culture in an SME and by a knowledge cost viewer which has the function of a controlling instrument. The SME knowledge toolkit is tailored toward the special needs of SMEs during knowledge management initiatives.

DISCUSSION AND FUTURE RESEARCH

Knowledge process modeling for SMEs uses the building block approach from Probst, Raub and Romhardt (2002) and models corresponding knowledge methods for the SME-domain. A significant emphasis in this empirical study has been the development of a flexible and usable knowledge management toolkit to implement knowledge initiatives in SMEs. This chapter addresses one of the currently perceived issues surrounding knowledge management, namely the lack of defining key knowledge processes for SMEs to handle knowledge methods in specific settings. The study has drawn on an extensive review of the literature as well as reported on empirical studies concerning the Austrian and Swiss SME sector.

Social Impact to SMEs

Even if the information technology infrastructure is a key dimension of knowledge management, the social and cognitive aspect may also be considered. Cognitive activity takes place in a complex, information-rich, and ever-changing environment. Tomasello (1999) notes that all human beings share their cognitive skills and knowledge with other individuals. The try to understand other persons, changes the way of interaction and social learning. A successful implementation of knowledge management only can be achieved in a culture (Holden, 2002) that supports knowledge sharing and transfer. Nakra (2000) addresses the issue that a knowledge culture is the most important value for the implementation of knowledge management because one important aspect of knowledge management is having a culture that fosters collaboration and sharing. Organizations often fail to acknowledge that it is the people, not technologies, that are the source of knowledge" (Nakra, 2000, p. 54). Organizational knowledge resides in the culture, structure, and individuals who make up the organization.

Especially in SMEs the success depends on the social system in which a knowledge worker operate. SMEs need a culture that facilitates a knowledge transfer through a more human factor because knowledge workers want to share their knowledge through communication and interaction. The social ecology of the company drives people's expectations, defines their motivation to share knowledge and pursues actions to interact with others inside or outside the organization. Ein-Dor (2005, p. 848ff) defines taxonomies of knowledge and sees the social-individual dimension as a key success factor to knowledge diffusion. A similar view is presented by Alavi und Leidner (2001). SMEs need a culture that facilitates a knowledge transfer through a more human factor because knowledge workers want to share their knowledge through communication and interaction (Trompenaars & Hampden-Turner, 2006). The use of information technology, especially Web sites, is recognized as a critical success factor for knowledge management initiatives in the SME sector (Wong & Aspinwall, 2005). Wong (2005) sees information technologies as a key enabler for the implementation of knowledge management, and considers in the development of a knowledge management system factors such as the simplicity of technology, ease of use, suitability of users' needs, relevancy of knowledge content, and standardization of a knowledge structure as key factors for knowledge diffusion. The SME culture greatly influences the communication processes inside and outside the organization. The impact of the culture to networking processes is highly significant. Skyrme (1999, p. 15) lists two fundamental descriptions of networking:

- Networking organizations are less about organizational structures per se, and more about the informal human networking processes

- The technology of computer networking undergirds and enhances human networking.

Knowledge networking is a dynamic process in which knowledge, experiences, and expertise are shared, developed, and evolved. A knowledge-sharing culture can be developed through human interaction supported by information technology to foster new and innovative knowledge. Knowledge networking is connectivity to achieve new levels of interactivity, usability, and understanding across organizations and communities. The first results of an empirical study conducted by the authors (Fink & Ploder, 2007b) indicate that the social view is almost not taken into consideration for European SMEs. The focus lies primarily on the technical solution of knowledge management, meaning that an IT support is essential for knowledge initiatives.

Therefore, future research will direct into the development of a framework for SMEs that integrates the information technology view with the social-cognitive view. One future problem for the implementation of different software products are the interfaces of the different applications. With service-oriented architecture (SOA), it would be possible to solve this problem of interoperability and the problem of security (Kang, Kim, Lo, Montrose, & Khashnobisch, 2006). Future research will deal with SOA and should also consider open source software (OSS) (Ploder & Fink, 2007) and extending the knowledge toolkit for SMEs.

ACKNOWLEDGMENT

This chapter is an extended version of the paper presented at the International Resource Management Conference (IRMA) May 2007 in Vancouver. Sincere thanks to Dr. Murray Jennex for the opportunity of consideration it in the *International Journal of Knowledge Management.*

REFERENCES

Alavi, M., & Leidner, D. (2001). Review: Knowledge management and knowledge management systems. *MIS Quarterly, 25*(1), 107-136.

APEC. (2006). Asia-Pacific Economic Corporation. http://www.apec.com.

ASME. (2006). Austrian Institute for SME Research. http://www.kmuforschung.ac.at.

Back, A., Enkel, E., & Krogh, G. (Eds.). (2007). *Knowledge networks for business growth*. Heidelberg: Springer Verlag.

Balmisse, G., Meingan, D., & Passerini, K. (2007). Technology trends in knowledge management tools. *International Journal of Knowledge Management, 3*(2), 118-131.

Beijerse, R. (2000). Knowledge management in small and medium-sized companies: Knowledge management for entrepreneurs. *Journal of Knowledge Management, 4*(2), 162-179.

CHSME. (2006). SME-Portal of Switzerland. http://www.kmu.admin.ch/index.

Coakes, E., & Clarke, S. (2006). Communities of practice. In: D. Schwartz (Ed.), *Encyclopedia of knowledge management*, (pp. 30-33). Hershey, PA: Idea Group Publishing.

Davenport, T. (1995). *Process Innovation. Reengineering work through information technology.* Boston: Harvard Business School Press.

Davenport, T., & Prusak, L. (1998). *Working knowledge. How organizations manage what they know.* Boston, MA: Harvard Business School Press.

Delahaye, D. (2003). Knowledge management at SMEs. *International Journal of Organizational Behavior, 9*(3), 604-614.

Dezouza, K., & Awazu, Y. (2006). Knowledge management at SMEs: Five peculiarities. *Journal*

of Knowledge Management, 10(1), 32-43.

Donnellan, B., Conboy, K., & Hill, S. (2006). IS to support innovation: Weapons of mass discussion. In: M. Khosrow-Pour (Ed.), *Emerging trends and challenges in information technology management,* (pp. 623-626). Hershey, PA: Idea Group Publishing.

EC. (2000). Report of the Meeting of the European Commission 23.03.2000. http://ec.europa.eu/growthandjobs.

Edwards, J., & Kidd, J. (2003). Bridging the gap from the general to the specific by linking knowledge management to business process management. In: V. Hlupic (Ed.), *Knowledge and business process management,* (pp. 124-132). Hershey, PA: Idea Group Publishing.

Ein-Dor, P. (2005). Taxonomies of knowledge. In: D. Schwartz (Ed.), *Encyclopedia of knowledge management,* (pp. 848-854). Hershey, PA: Idea Group Publishing.

Fink, K. (2004). *Knowledge potential measurement and uncertainty.* Wiesbaden: Dt. Univ.-Verl.

Fink, K., & Ploder, C. (2006). The impact of knowledge process modeling on small and medium-sized enterprises. In: K. Tochtermann & H. Maurer (Eds.), *Proceedings of I-KNOW '06: 6th International Conference on Knowledge Management,* (pp. 47-51). Graz: J.UCS.

Fink, K., & Ploder, C. (2007a). A comparative study of knowledge processes and methods in Austrian and Swiss SMEs. In: H. Österle, J. Schelp, & R. Winter (Eds.), *Proceedings of the 15th European Conference on Information Systems (ECIS2007),* (pp. 704-715). St. Gallen.

Fink, K., & Ploder, C. (2007b). Knowledge diffusion through SME Web sites. In: C. Stary, F. Brarachini, & S. Hawamdeh (Eds.), *Knowledge management: Innovation, technology and cultures,* (pp. 91 - 100). New Jersey: World Scientific.

Fink, K., & Ploder, C. (2007c). Knowledge process modeling in SME and cost-efficient software support: Theoretical framework and empirical studies. In: M. Khosrow-Pour (Ed.), *Managing worldwide operations and communications with information technology,* (pp. 479-484). Hershey, PA: IGI Publishing.

Fink, K., Roithmayr, F., & Ploder, C. (2006). Multi-functional stakeholder information system for strategic knowledge management: Theoretical concept and case studies. In: M. Khosrow-Pour (Ed.), *Emerging trends and challenges in information technology management,* (pp. 152-155). Hershey, PA: Idea Group Publishing.

Hammer, M., & Champy, J. (1993). *Reengineering the corporation: A manifesto for business revolution.* New York: Harper Business.

Holden, N. (2002). *Cross-cultural management: A knowledge perspective*. Harlow: Prentice Hall.

Jashapara, A. (2004). *Knowledge management: An integrated Approach*. Upper Saddle River, NJ: Prentice Hall.

Jennex, M. (2007). *Knowledge management in modern organizations.* Hershey, PA: Idea Group Pub.

Kang, M., Kim, A., Lo, J., Montrose, B., & Khashnobisch, A. (2006). Ontology-based security specification tools for SOA. In: M. Khosrow-Pour (Ed.), *Emerging trends and challenges in information technology management,* (pp. 619-622). Hershey, PA: Idea Group Publishing.

Laudon, K., & Laudon, J. (2006). *Management information systems: Managing the digital firm* (9. , 2. print. ed.). Upper Saddle River, NJ: Pearson Education.

McAdam, R., & Reid, R. (2001). SME and large organization perception of knowledge management:

Comparison and contrast. *Journal of Knowledge Management, 5*(3), 231-241.

Nakra, P. (2000). Knowledge management: The magic is in the culture. *Competitive Intelligence Review, 11*(2), 53-60.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge creating company. How Japanese create the dynamics of innovation*. New York/Oxford: Oxford University Press.

OECD. (2005). OECD SME and Entrepreneurship Outlook 2002/2005. Retrieved November 4, 2006, from http://www.oecd.org/document/.

Ordanini, A. (2006). *Information technology and small businesses: Antecedents and consequences of technology adoption*. MA: Edward Elgar Publishing.

Ploder, C., & Fink, K. (2007). An orchestration model for knowledge management tools in SMEs. In: K. Tochtermann & H. Maurer (Eds.), *Proceedings of I-KNOW '07: 7th International Conference on Knowledge Management,* (pp. 176-183). Graz: J.UCS.

Probst, G., Raub, S., & Romhardt, K. (2002). *Managing knowledge: Building blocks for success*. Chichester, UK: Wiley.

Richter-von Hagen, C., Ratz, D., & Povalej, R. (2005). Towards self-organizing knowledge- intensive processes. *Journal of Universal Knowledge Management, 0*(2), 148-169.

Ruggels, R. (1997). *Knowledge management tools*. Boston: Butterworth-Heinemann.

SA. (2006). Statistical Yearbook of Austria. http://www.statistik.at.

Salojärvi, S., Furu, P., & Sveiby, K. (2005). Knowledge management and growth in Finnish SMEs. *Journal of Knowledge Management, 9*(2), 103-122.

Skyrme, D. (1999). *Knowledge networking. Creating the collaborative enterprise*. Oxford: Butterworth-Heinemann.

Smolnik, S., Kremer, S., & Kolbe, L. (2007). The role of context and its explication for fostering knowledge transparency in modern organizations. In: M. Jennex (Ed.), *Knowledge management in modern organizations,* (pp. 256-277). Hershey, PA: Idea Group Publishing.

Sveiby, K. (1997). *The new organizational wealth: Managing and measuring knowledge-based assets*. San Francisco: Berrett-Koehler.

Tiwana, A. (2002). *The knowledge management toolkit: Orchestrating IT, strategy, and knowledge platforms* (2nd ed.). Upper Saddle River, N.J.: Prentice Hall.

Tomasello, W. (1999). *The cultural origins of human cognition*. Boston/London: Harvard University Press.

Trompenaars, F., & Hampden-Turner, C. (2006). *Riding the waves of culture: Understanding cultural diversity in business* (2. reprint. with corr. ed.). London: Brealey.

Wang, F., & Plaskoff, J. (2002). An integrated development model for KM. In: R. Bellaver & J. Lusa (Eds.), *Knowledge management strategy and technology,* (pp. 113-134). Boston: Artech House.

Wong, K. (2005). Critical success factors for implementing knowledge management in small and medium enterprises. *Industrial Management & Data Systems, 105*(3), 261-279.

Wong, K., & Aspinwall, E. (2005). An empirical study of the important factors for knowledge—management adoption in the SME sector. *Journal of Knowledge Management, 9*(3), 64-82.

This work was previously published in International Journal of Knowledge Management, Vol. 5, Issue 1, edited by M. Jennex, pp. 46-60, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 3.2
Information Communication Technology Tools for Software Review and Verification

Yuk Kuen Wong
Griffith University, Australia

INTRODUCTION

While information communication technology (ICT) can be considered a well-established discipline, software development projects are still prone to failure. Even if a software project is not classified as a failure, the general level of software quality leaves room for much improvement. It has been suggested that one of the most prevalent and costly mistakes made in software projects today is deferring the activity of detecting and correcting software problems until the end of the project. Hence, the cost of rework in the later stages of a project can be greater than 100 times the project costs. About 80% of avoidable rework comes from 20% of defects. As a result, techniques such as software review for improving software quality are important.

Software review (inspection/verification) was originally introduced by Fagan (1976). The review process essentially includes six major steps:

1. **Planning:** Organize and prepare the software review, typically for preparing the review materials and review procedure, forming review team and scheduling review meeting, selecting review participants, and assigning roles.
2. **Overview:** Author explains overall scope and the purpose of the review.
3. **Individual preparation:** Individual reviewers analyze and review the software artefact.
4. **Group review meeting:** Find errors, sometimes also called "logging meeting." Review teams correct and the reader summarizes the work.
5. **Rework:** Defect correction, which involves the author in resolving problems by reviewing, revising, and correcting the identified defect or by decreasing the existence of errors of the software artefact.
6. **Follow-up:** Validate the correction quality and decide if re-inspection is required

Since Fagan (1976) introduced software review and verification as an important technique to assure the quality of software projects, researchers have investigated ways to improve software review performance. It has been suggested that ICT software review tools are one of the important elements to the support software review process. This article overviews ICT tools to assist software review and verification during this software review process. The overall objective of this article is to identify various ICT tools that support software review and verification. This includes a discussion of the importance of software quality and identifies ICT tools for effective software reviews and verification.

ICT TOOLS FOR SOFTWARE REVIEW AND VERIFICATION

Intelligent Code Inspection in a C Language Environment (ICICLE)

The ICICLE (Intelligent Code Inspection in a C Language Environment) is the first published software review tool, which was developed at Bellcore (Brothers, Sembugamoorthy, & Muller, 1990). The ICICLE tool is designed to support code review and assists reviewers in both individual preparation and group meetings. ICICLE provides a synchronous communication support to group meetings. It has been argued that traditional code review meeting is manually documented (i.e., using paper and pen to record defects detected). This documentation procedure is very time consuming, tedious, and could be inconsistent recording (Brothers et al., 1990). One of the aims of this tool helps software reviewers to find obvious defects. Brothers and his team (1990) suggested that ICICLE provide several benefits to code review:

- To detect routine sorts of errors, with the goal of freeing the code inspector (reviewer) to concentrate on verifying the correct implementation of requirement, specifications, and designs.
- To offer various forms of knowledge about the code being inspected (reviewed), including domain and environment knowledge, and information from various forms of analysis such as cross-referencing.
- To allow code inspectors (reviewers) to easily traverse source code in a windowed environment instead of riffling through hard copy from many different files.
- To render the code inspection (review) meeting paperless through a shared window interface which enables the code inspectors (reviewers) to fulfill their roles electronically.

The ICICLE tool consists of two phases in the review process: the individual review and group review meeting. The group review meeting takes in the same location/venue, usually a reviewers' seat at nearby computers. An individual reviewer allows entering comments on each line of code. According to MacDonald, Miller, Brooks, Roper, and Wood (1995), the researcher found that "the computer supported meeting format appeared to cause substantial changes in the dynamics of the code inspection (review) meeting." In other words, the procedures of the code review meeting using ICICLE can enable roles during the group meeting process (Brothers et al., 1990). For example, the additional duty of a moderator is to record statistics relating to coding defects discovered during code review. The reader can direct the attention of the other reviewers to areas of interest in the source code. The scribe's records must be agreed on by the review team. The author should present in the code review meeting and answer the reviewers' question. Any additional reviewers can participate and share meeting discussions.

Scrutiny

Scrutiny is an online collaborative software review tool, which was developed at Bull HN Information Systems in conjunction with the University of Illinois (Gintell, Houde, & Mckenney, 1993). It is a synchronous meeting review tool. It is one of the early comprehensive collaborative process software review tools (MacDonald et al., 1995). Scrutiny currently supports text documents only.

Scrutiny can be used in the formal review process, and it supports multi-users review but does not support for rules and checklists. It provides a "petri-net based process modeling language" that allows the system to implement alternative software review methods, such as a "shared preparation" phase in which reviewers have access to each other's preliminary findings (Gintell et al., 1993; MacDonald et al., 1995). However, in comparison with the ICICLE, the Scrutiny usage can depart radically from manual software review processes, such as geographically distributed software reviews (MacDonald et al., 1995).

Collaborate Software Inspection (CSI)

Collaborate software inspection (CSI) was built and used in a case study to compare online distributed computer-mediated software review meetings vs. face-to-face software review meetings at the University of Minnesota (Mashayekhi, Feulner, & Riedl, 1994). As with other software review tools, the CSI provides the similar process characteristics of Humphrey's software review method with hypertext capability. CSI is developed for group review meetings in the (1) same time and place, (2) same time and place, (3) same time and different place, and (4) different time and same place. CSI supports both asynchronous and synchronous activities that include materials distribution, individual preparation/individual review, group review meeting, recording, and reporting (Mashayekhi et al., 1994).

InspeQ

InspeQ was developed to support the phased software review process (Knight & Myers, 1993). The InspeQ was executed on "Sun 3, Sun 4, and IBM RS/6000 computers running various forms of Unix and the X-window display system and the OSF/Motif widget set" (Knight & Myers, 1993). Although the InspeQ achieves the goals of efficiency and rigor in the phased review process, it is not viewed as essential to the "phased inspection" method (MacDonald et al., 1995). Similar to other software review tools, InspeQ provides numbers of facilities to support software review process. These include work product display (views the documents), checklist display (allows the status of each checklist to be displayed and modified), standard display (review rational and a detailed descriptions), highlight display (helps locating particular aspects of the documents), and comments display (comments on the documents) (Knight & Myers, 1993).

CSRS

CSRS is developed to support computer-mediated communication formal software review process (Johnson, 1994). The CSRS is heavily used in academic research and laboratory experiments studies. The goals of CSRS is to reduce the human effort in review process by conducting software review incrementally during the development and provide online capabilities to collect metrics (Stein, Riedl, Harner, & Mashayekhi, 1997). The CSRS is similar to Scrutiny; it provides an internal process modeling mechanism to support a variety of review methods. CSRS's primary method is FTArm, which "is unique among methods by being designed explicitly to support properties of computer-mediated communication and review"

(Johnson, 1994). CSRS automatically collects data (e.g., number of defects/issues found, comments made, time spent on software review, starting time, finishing time, event logs, etc.). Another feature of CSRS is that it supports a variety of software review processes and handles several types of documents and languages.

Requirement Traceability Tool (RADIX)

Requirement traceability refers to the "ability to describe and follow the life of a requirement, in both a forwards and backward direction" (Gotel & Finkelstein, 1994). The requirement traceability tool (RADIX) is designed for verifying software requirements (Yu, 1994; Yu, Smith, & Huang, 1990). A requirement tracing method is a systematic method to assist 5ESS Switch scientist and engineers to deliver quality software (Yu, 1994).

Asynchronous Inspector of Software Artefacts (AISA)

Asynchronous Inspector of Software Artefacts (AISA) was the early Web-based software review tool (Stein et al., 1997). AISA supports asynchronous and distribution software review as well as reviewing both textual and graphical documents (e.g., entity-relationship diagram or class diagram) (Stein et al., 1997). The AISA Web-based tools can be built using existing structures and were reasonably easy to develop.

Web Inspection Prototype (WiP)

Web inspection prototype (WiP) is another Web-based tool, which provides a set of functions for distributing the requirement documents to be reviewed (Harjumaa & Tervonen, 1998). The WiP is developed to support distributed software review processes. It utilizes the WWW to distribute the review document, tailors the software review process model according to the development environment, can assign roles, adds or removes a checklist, allows reviewers to add, modify, and remove comments, has searching, cross-referencing, e-mail notification, combining comments/annotations using hypertext and reporting capabilities, and generates metrics and statistics summary (Harjumaa & Tervonen, 1998).

InspectA

InspectA is a completed automation tool, which allows the whole software review process to be automated (Murphy & Miller, 1997) from planing, such as selecting a moderator and communications between software review team, through follow-up stage, where the moderator verified and finalized that all the defects raised during the software review have been resolved.

InspectA is an e-mail-based and asynchronous tool for software review (Murphy, Ferguson, & Miller, 1999). The tool is not based on the World Wide Web approach (all reviewers can view other individual reviewer comments). The argument is when reviewers can view other reviewers members defect lists during the individual preparation that they may discourage (Murphy et al., 1999). Another reason is that reviewers may focus or discuss other reviewers' comments rather than focus their own individual reviews (Murphy et al., 1999). Miller and their team believed that e-mail-based tools provide a number of advantages. These include (Stein et al., 1997, p. 108):

- **Sharing information:** Allow exchange and sharing information via e-mail tool
- **Threads of discussion:** Allow individual reviewers to contribute and free feel to comments during the individual preparation
- **Visual cues:** Easy to format in the document such as bold or italics
- **Train of thought:** Offer other reviewers to response and reply

- **Reaching a consensus:** Voting can be sent via e-mail tool
- **Coordination:** Moderator can easily send information to all reviewers
- **History:** Allow reviewers to keep a record of their comments

HyperCode

HyperCode is another one of the earliest Web-based software review tools (Perpich, Perry, Porter, Votta, & Wade, 1997). It uses the common gateway interface (CGI), and the HyperCode system allows software reviewers using WWW browsers for software review (Perpich et al., 1997). It is used for code review and very similar to the WiP. Reviewers can comment on the Web, and material is delivered via the Web (Perpich et al., 1997). The process contains only asynchronous phases, and other reviewers are able to see the comments, so there is no software review meeting required when in HyperCode (Perpich et al., 1997). After individual review, the results are collated together into a report containing links from the original material to the comments (Perpich et al., 1997).

Perry and his team (Perry, Porter, Wade, Votta, & Perpich, 2002) suggested that there are four primary differences between paper-based and HyperCode software review processes. First, HyperCode provides an automated approach to the support software review process; this can reduce time and amount of review effort. Second, notification between software review team can be a useful e-mail tool. Third, all comments or annotations are visible to all participated software reviewers in the whole review process. Fourth, there is no meeting requirement needed in the HyperCode process. In other words, the review discussion is asynchronous communication.

ASYNCHRONOUS/SYNCHRONOUS SOFTWARE INSPECTION SUPPORT TOOL (ASSIST)

Asynchronous/Synchronous Software Inspection Support Tool (ASSIST) is built to provide both individual and group software review (MacDonald et al., 1995). It can perform both synchronously or asynchronously with either different place or same place meeting. It uses a costumed designed modeling language, the Review Process Definition Language (RPDL) and a flexible document type system to allow support of any software review process (MacDonald & Miller, 1999). ASSIST is a client/server architecture, where the server is a central database to store documents and data. Table 1 shows details of four features of the ASSIST tool.

Fine-Grained Software Inspection Tool/CodeSurfer

The fine-grained software review tool is designed for "exposing the results of sophisticated whole-program static analysis" (Anderson, Reps, & Teitelbaum, 2003) to the software review. This is also known as CodeSurfer. The idea was originally developed from the "dependence graphs" which applications activities include parallelization (Burke & Cytron, 1986), optimization (Ferrante, Ottenstein, & Warren, 1987), program testing (Bates & Horwitz, 1993), and software assurance (Horwitz & Reps, 1992). A description of how this tool work is summarized in the following section.

CORD

The CORD is developed for increasing the consistency between requirements and a detailed design (Chechik & Gannon, 2001). The CORD creates a "finite state abstraction" of a detailed design and checks it against a set of properties

automatically generated from the requirements (Anderson et al., 2003). Its features are similar to other static analysis tools such as CodeSuper. The aim of CORD is to "simplify the verification of properties of program; this system abstract the forms of their formal specification notations or create abstract models from program that could be analysed with state-exploration" (Chechik & Gannon, 2001, p. 669).

Agent-Based Software Tool

Agent-based software tool in code review is designed by Chan (2001). Chan (2001) recently proposed the agent-based software tool that can help reduce the cost and increase the number of defects detected in the software review process. The main focus of the intelligent agent software tool (Chan, 2001) is to:

- Automate as much of the paper work and administrative tasks as possible
- Enable the inspection (review) to perform inspection (review) according to their schedules. This aims to reduce the inspection (review) interval, and thus reducing the turn around time for obtaining inspection (review) results
- Provide as much assistance to the inspector (reviewer) as possible during preparation
- Maximize the number of major faults found while keeping the time costs of the process low

INTERNET-BASED INSPECTION SYSTEM (IBIS)

Internet-based inspection system (IBIS) is developed to support geographically distributed software review (Lanubile & Mallardo, 2002). IBIS was originally designed by Cavivano, Lanubile, and Visaggio (2001). This tool is another Web-based application which is "based on a lightweight architecture to achieve the maximum of simplicity of use and deployment" (Lanubile & Mallardo, 2002). The IBIS can support Fagan's Software Review Process. There are a several advantages of deploying IBIS (Lanubile & Mallardo, 2002):

- IBIS is Web-based software review tool; it allows reviewers access from their desktop. This could improve the chance of reviewers participating in software review
- It allows the software review to be performed in different places, even in different countries
- Allow different experts participate in the software review process. Those experts could from outside the organization or different department.

VisionQuest

VisionQuest aims to support the experimentation formal anonymity technical review (Vitharana & Ramaurthy, 2003). The advantages of anonymity in group collaboration are (Er & Ng, 1995):

- Each reviewers has equal weight
- Avoid the dominance group or status effect during the review process
- Since the comments are made by an anonymous person, criticisms are the issues rather than people
- Voting is anonymous; the final decision is more likely to be objective and based on merit

FUTURE

A number of computer support tools have been developed to support the software review meeting process. Many tools provide documentation facilities that allow software review documents to be shared across networks, browsed online,

and edited by reviewers. The current trend of software review is for using software review tools to the support software review process. Past research has shown a spectrum of advantages of using software review tools supporting technical review. First, a computer-support tools review environment can reduce paper work and clerical costs, decrease error rates of recording review meeting and comments, and allow computerised data collection and data analysis. Software review tools can integrate the review method with other components of the specific software development method such as asynchronous review and facilitating both metrics collection. Companies will adopt software review tools simply because of such potential benefits over manual software review techniques.

CONCLUSION

In summary, this article described modern software review tools and techniques and the different types of software review tools and how they work. To achieve better software review performance, it is important to understand use of inputs and the software review process.

REFERENCES

Anderson, P., Reps, T., & Teitelbaum, T. (2003). Design and implementation of a fine-grained software inspection tool. *IEEE Transaction on Software Engineering*, 721-733.

Bates, S., & Horwitz, S. (1993). Incremental program testing using program dependence graphs. In *Proceedings of the Symposium on Principles of Programming Language* (pp. 384-396).

Brothers, L., Sembugamoorthy, V., & Muller, M. (1990, October). ICICLE: Groupware for code inspection. In *Proceedings of the 1990 ACM Conference on Computer Supported Cooperative Work* (pp. 169-181).

Burke, M., & Cytron, R. (1986). Inter-procedural dependence analysis and parallelization. In *Proceedings of SIGPLAN '86 Symposium Complier Construction* (pp. 162-175).

Cavivano, D., Lanubile, F., & Visaggio, G. (2001). Scaling up distributed software inspections. In *Proceedings of 4th ICSE Workshop on Software Engineering over the Internet*, Toronto, Canada.

Chan, K. (2001, August 27-28). An agent-based approach to computer assisted code inspections. In *Proceedings of Australian Software Engineering Conference* (pp. 147-152).

Chechik, M., & Gannon, J. (2001, July). Automatic analysis of consistency between requirement and design. *IEEE Transactions on Software Engineering, 27*(7), 651-672.

Er, M., & Ng, A. (1995). The anonymity and proximity factors in group decision support systems. *Decision Support Systems, 14*(1), 75-83.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM System Journal, 15*(3), 182-211.

Ferrante, J., Ottenstein, K., & Warren J. (1987). The program dependence graph and its use in optimization. *Transaction Programming Languages and System, 3*(9), 319-349.

Gintell, J. W., Houde, M. B., & Mckenney, R. F. (1993, July 10-14). Lessons learned by building and using scrutiny: A collaborative software inspection systems. In *Proceedings of the 7th International Workshop on Computer-Aided Software Engineering* (pp. 350-357).

Gotel, O. C. Z., & Finkelstein, C. W. (1994, 18-22 April). An analysis of the requirements traceability problem. In *Proceedings of the 1st International*

Conference on Requirement Engineering (pp. 94-101).

Harjumaa, L., & Tervonen, I. (1998, January 6-9). A WWW-based tool for software inspection. In *Proceedings of the 31st Hawaii International Conference on System Sciences* (pp. 379-388).

Horwitz, & Reps, T. (1992, May). The use of program dependence graphs in software engineering. In *Proceedings of the 14th International Conference on Software Engineering* (pp. 392-411).

Johnson, P. M. (1994, May). An instrumented approach to improving software quality through formal technical review. In *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy.

Knight, J. C., & Myers, A. E. (1993, November). An improved inspection technique. *Communications of the ACM, 36*(11), 50-69.

Lanubile, L., & Mallardo, T. (2002). Preliminary evaluation of tool-based support for distributed inspection. In *Proceedings of the ICSE International Workshop on Global Software Development,* Orlando, FL.

MacDonald, F., & Miller, J. (1997). A comparison of tool-based and paperbased software inspection (EfoCS-25-97, RR/97/203). University of Strathclyde, Department of Computer Science, Empirical Foundations of Computer Science (EFoCS).

MacDonald, F., & Miller, J. (1999). ASSIST: A tool to support software inspection. *Information and Software Technology, 41,* 1045-1057.

MacDonald, F., Miller, J., Brooks, A., Roper, M., & Wood, M. (1995, July 10-14). A review of tool support for oftware inspection. In *Proceedings of the 7th International Workshop on Computer-Aided Software Engineering,* Toronto, Canada (pp. 340-349).

Mashayekhi, V., Feulner, C., & Riedl, J. (1994). CAIS: Collaborative asynchronous inspection of software. In *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering, 19*(5), 21-34. New York.

Murphy, P., Ferguson, J. D., & Miller, J. (1999). Groupware support asynchronous document review. In *Proceedings of the 17th Annual International Conference on Computer Documentation,* New Orleans, LA (pp. 185-192).

Murphy, P., & Miller, J. (1997). A process for asynchronous software inspection. In *Proceedings of the 18th IEEE International Workshop on Software Technology and Engineering* (pp. 96-104).

Perpich, J. M., Perry, E. D., Porter, A. A., Votta, L. G., & Wade, M. W. (1997). Anywhere, anytime code inspections: Using the Web to remove inspection bottlenecks in large-scale software development. In *Proceedings of the International Conference on Software Engineering, ICSE* (pp. 14-21).

Perry, D. E., Porter, A., Wade, M. W., Votta, L. G., & Perpich, J. (2002, July). Reducing inspection interval in large-scale software development. *IEEE Transactions on Software Engineering, 28*(7), 695-705.

Stein, M. V., Riedl, J., Harner, S. J., & Mashayekhi, V. (1997). A case study of distributed, asynchronous software inspection. In *Proceedings of the 19th International Conference on Software Engineering* (pp. 107-117).

Yu, W. D. (1994, February). Verifying software requirement: A requirement tracing methodology and its software tools: RADIX. *IEEE Journal on Selected Areas in Communications, 12*(2).

Yu, W. D., Smith, D. P., & Huang, S. T. (1990, May/June). Software productivity measurements. *AT&T Technology Journal, 69*(3), 110-120.

KEY TERMS

Asynchronous Review: The review activities can be performed at the same time. Asynchronous review must rely on the ICT tools supported.

Defects: The term "defect" is defined as any issue or problem that does not meet the requirements.

Fagan Inspection: Software review was originally proposed by Michael Fagan at IBM in the early 1970s. Fagan's software review and forms of review structures. Fagan's software review includes six-step review processes: planning, overview, preparation, group meeting, re-review, and follow up.

Information Communication Technology (ICT) Review Tools: Tools provide documentation facilities that allow software review documents to be shared across networks, browsed online, and edited by reviewers. The current trend of software review is for using software review tools to support the software review process.

Online Software Review: Online software review tools support some major functions: document support, individual preparation, and meeting support. It can meet online and share data and documents in flexible time, thus it may reduce time of review.

Software Review: Software review or inspection is one of the techniques for improving software quality. Software review is an industry-proven process for eliminating defects. It has been defined as a non-execution-based technique for scrutinizing software products for defects, deviations from development standards.

Software Verification: Aims to find and remove defects during the software development cycle.

Synchronous Review: All review activities in the synchronous software review happen in a linear fashion, and the meetings are located in same place at the same time.

This work was previously published in Encyclopedia of Information Communication Technology, edited by A. Cartelli & M. Palma, pp. 429-435, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 3.3
SurveyTracker E-Mail/ Web Survey Software

Eun G. Park
McGill University, Canada

ABSTRACT

This chapter offers an introductory description of SurveyTracker software. Comparisons are made to competitor software programs. The central focus is an explanation of the functions, features, and advantages of the software application.

INTRODUCTION

SurveyTracker is one of the most popular software applications in the current field of online survey, developed from Pearson NCS Inc., and currently registered from Training Technologies, Inc. This company provides the integrated and powerful SurveyTracker line of software applications on survey, including SurveyTracker, SurveyTracker Plus, SurveyTracker E-Mail/Web, SurveyTracker Plus E-Mail/Web and Survey-by-Disk. One advantage of using this comprehensive software is to provide many functions as one module, and offer flexibility and extensibility for prewritten survey modules, specialized training, survey design and distribution services, and survey consulting.

As an introductory description of SurveyTracker, this chapter explains the functions, features, and advantages of the software application. The software supports functional requirements that educational software needs, including survey authoring and design, interface, data management, and multiple electronic and paper data collection technologies. It can especially integrate all forms of data collected from e-mails in text and images, the Internet, networks, and scanning forms. This software can be easily applicable to a variety of survey-based research on schools, higher education, government, business, and healthcare.

FUNCTIONS

The latest versions of the SurveyTracker line of software are SurveyTracker E-Mail/Web 4.5, SurveyTracker Classic 4.5, and SurveyTracker E-Mail/Web Network 4.5. These software applications support several advantageous features in terms of interface, data management, reporting, and distribution.

Interface. SurveyTracker is based on user-friendly and graphic-based design so that users can design a survey form with the overview, questions, notes, sections, and a summary, quickly and easily with a simple click of the mouse. Users can get screen views as designed and displayed in the mode of what-you-see-is-what-you-get in 32-byte interface. Regarding editing, the software is flexible, with text-editing functions including character sets, fonts, color, size, and other editing settings. Graph provides enhanced color support, a variety of data marker shapes, and improved flexibility. Convenient shortcuts and standard text-editing functions allow survey designers to create and modify texts fast. It displays overall updates of ongoing projects. The built-in question library is chosen by a drag-and-drop interface. The built-in image library also provides quick access to commonly used graphics. Users can create global layout settings for all current and future surveys.

Data collection and management. File import is fast and easy from the previous version or other software applications. Data and report export is easy for use in Microsoft Office and other products. Automatic recording of returned surveys is possible to directly send reminder messages. The survey forms and reports are printed, distributed, collected, and read back into SurveyTracker. There are many options for creating, managing, distributing, responding to, analyzing, and reporting from response data. Spreadsheet-style data collection for rapid manual data entry can be changed to different layouts. The data collection screen offers instant electronic survey retrieval, and convenient manual response entry for paper-based surveys. Open-ended questions can be coded for quantitative data analysis, or printed out verbatim in a report. Regarding coding, the built-in codebook supports up to 300 codes per question, as defined as single response or multiple response for multiple code entry. Automatic filtering and batch reporting are possible for the Web. It is easy to customize score values after data collection. Identification bar codes on scannable forms and unique lithocodes on each survey are used to trace printed forms. A number of statistical analyses are built in, such as frequency, variance, minimum, sum, maximum, skewness, range, correlation, chi-square, standard deviation, significance, and so on. SurveyTracker can analyze through manual configuration or autofiltration for maximum reporting flexibility. Tables and graphs for reports are used with multitable forms. All distribution is handled by conducting multiple distributions. E-mail responses can be sent directly through e-mail systems.

Access and retrieval. A single source database holds all project information for faster access and better organization. The message library allows users to store and retrieve survey and report notes/instructions, as well as distribution.

SYSTEM REQUIREMENTS

To install SurveyTracker, systems require an IBM PC or compatible with at least a Pentium® 200 MHz processor (Pentium class processor running at 300MHz or better strongly recommended), at least 64MB RAM (128MB or more is strongly recommended), in Microsoft Windows 95, 98 SE, ME, Windows NT® or 2000 (Professional), at least 100MB hard disk space (more may be needed depending on the size of the survey and the number of respondents).

FEATURES FOR EDUCATIONAL RESEARCH

SurveyTracker has competitive advantages over the following three aspects:

Flexibility. Most of all, SurveyTracker can support a variety of survey forms. It is compatible with traditional paper surveys, e-mail surveys, and Web surveys. Each respondent opens a program on the disk to access and complete the survey. When the survey is returned, the survey administrator reads it into SurveyTracker. Survey answers are created in SurveyTracker and then converted to scannable forms, using the fully integrated features of the other software application. In network environments, surveys are possible to distribute on network or placed on a disk individually. Surveys can also be carried in intranet or extranet. Paper surveys directly support optical mark recognition scanners and data sheets.

Compatibility. Different survey forms are compatible with Web survey forms. For example, disk surveys, and html-based e-mail surveys deliver the same look and feel directly to the respondent's e-mail inbox. The software works directly with a personal computer's e-mail system to send out the survey questionnaire in text-based messages, form-based messages, or html-based message.

Extensibility. The new SQL database engine handles millions of records for larger projects, audience lists, and more extensive reporting. Audience supports up to 200 fields per record. It is easy to enter a list of potential respondents directly, and to import an existing list from another software program up to 1.6 million people in an audience list. The audience list can be narrowed down as well. Users can create surveys with six scales per question up to 300 choices per scale. SurveyTracker includes a comprehensive library of prewritten questions and scales. Scale types in SurveyTracker include horizontal numerical, Likert, multiple choices, semantic differential, fixed sum, forced ranking, ordinal, and paired comparison.

COST AND LOCATION

Information on price and additional consulting services is available at the Web site of Training Technologies, Inc (http://www.SurveyTracker.com).

COMMENTARY

Currently, there are other survey software applications that are available in the market. For example, popular alternatives include WebSurveyor 5.0, Perseus SurveySolutions, Ultimate Survey, and WebSurveyor. Among them, SurveyTracker is recommended as a better way to integrate all forms of separate surveys, and to maximize many compatible functions. Next step will be needed to examine their comparative usage and implementation in depth on how to apply these software applications' functions to research on schools, higher education, government, business, and healthcare for ultimate results.

REFERENCES

KeySurvey (n.d). Retrieved March 1, 2005, from http://www.keysurvey.com

Perseus SurveySolutions (n.d). Retrieved March 1, 2005, from http://www.perseus.com

SurveyTracker (n.d.). Retrieved March 1, 2005, from http://www.SurveyTracker.com

Ultimate Survey (n.d). Retrieved March 1, 2005, from http://www.prezzatech.com

WebSurveyor 5.0 (n.d). Retrieved March 1, 2005, from http://www.websurveyor.com

KEY TERMS

Compatibility: The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.

Extensibility: A property of a system, format, or standard that allows changes in performance or format within a common framework, while retaining partial or complete compatibility among systems that belong to the common framework.

Flexibility: The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Online Survey: To conduct a statistical survey by means of online tools or methods in the procedures of survey, such as data gathering, data analysis, summary reports, distribution of results, and so forth.

What-You-See-is-What-You-Get (WYSIWYG): A user interface that allows the user to view something very similar to the end result, while the document or image is being created.

This work was previously published in Handbook of Research on Electronic Surveys and Measurements, edited by R. Reynolds, R. Woods, & J. Baker, pp. 416-17, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 3.4
A Survey of Selected Software Technologies for Text Mining

Richard S. Segall
Arkansas State University, USA

Qingyu Zhang
Arkansas State University, USA

ABSTRACT

This chapter presents background on text mining, and comparisons and summaries of seven selected software for text mining. The text mining software selected for discussion and comparison in this chapter are: Compare Suite by AKS-Labs, SAS Text Miner, Megaputer Text Analyst, Visual Text by Text Analysis International, Inc. (TextAI), Magaputer PolyAnalyst, WordStat by Provalis Research, and SPSS Clementine. This chapter not only discusses unique features of these text mining software packages but also compares the features offered by each in the following key steps in analyzing unstructured qualitative data: data preparation, data analysis, and result reporting. A brief discussion of Web mining and its software are also presented, as well as conclusions and future trends.

INTRODUCTION

The growing accessibility of textual knowledge applications and online textual sources has caused a boost in text mining and Web mining research. This chapter presents comparisons and summaries of selected software for text mining. This chapter reviews features offered by each package in the following key steps in analyzing unstructured qualitative data: data preparation including importing, parsing, and cleaning; data analysis including association and clustering; and result presenting/reporting including plots and graphs.

BACKGROUND OF TEXT MINING

Hearst (2003) defines text mining (TM) as "the discovery of new, previously unknown information, by automatically extracting information from

different written sources." Simply put, text mining is the discovery of useful and previously unknown "gems" of information from textual document repositories. Also Hearst (2003) distinguishes text mining from data mining by noting that with "text mining the patterns are extracted from natural language rather than from structured database of facts." A more technical definition of text mining is given by Woodfield (2004) author of SAS Notes for Text Miner, as a process that employs a set of algorithms for converting unstructured text into structured data objects and the quantitative methods used to analyze these data objects.

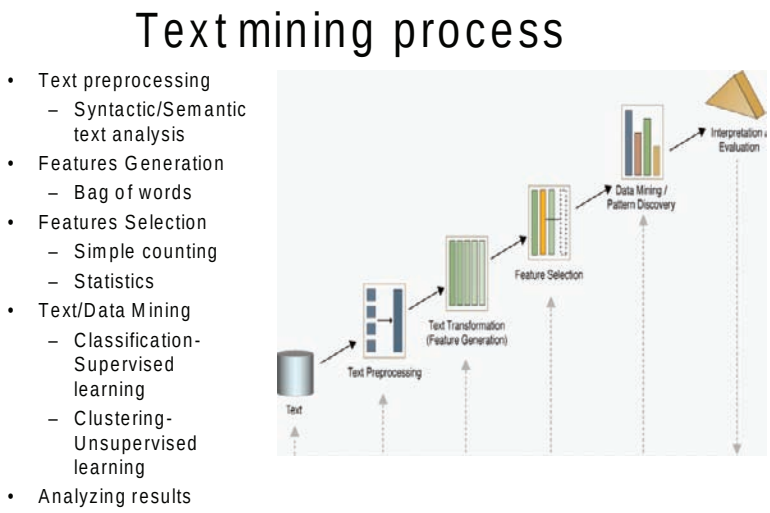
Text mining (TM) or text data mining (TDM) has been discussed by numerous investigators that include Hearst (1999), Cerrito (2003) for the application to coded information, Hayes et al. (2005) for software engineering, Leon (2007) for identifying drug, compound, and disease literature, and McCallum (1998) for statistical language modeling. Firestone (2005) emphasizes the importance of text mining in the future knowledge work. Romero and Ventura (2007) survey text mining applications in the educational setting. Kloptchenko et al. (2004) use data and text mining techniques for analyzing financial reports. Mack et al. (2004) describe the value of text analysis in biomedical research for life science. Baker and Witte (2006) discuss the mutation mining to support activities of protein engineers.

Uramoto et al (2004) utilized a text-mining system adopted from that developed by IBM and named TAKMI (Text Analysis and Knowledge Mining) for use with very large text biomedical text documents. In fact the extension of TAKMI was named MedTAKMI and was capable of mining the entire MEDLINE of 11 million biomedical journal abstracts. The TAKMI system allows extracting deeper relationships among biomedical concepts by the use of natural language techniques. Scherf et al. (2005) discuss the applications of text mining in literature search to improve accuracy and relevance. Kostoff et al. (2001) combine data mining and citation mining to identify user community, and its characteristics by categorizing articles.

There is a Text Mining Research Group (TMRG) (2002) at the University of Waikato in New Zealand that maintains a Web page of related publications, links, and software. Similarly there is National Centre for Text Mining (NaCTeM) at the University of Manchester in United Kingdom (UK). The Aims and Objectives of NACTeM is described in article by Ananiandou et al (2005) in which it extensively discusses a need for text mining in biology. According to their Web site of 2002, "text mining uses recall and precision (borrowed from the information retrieval research community) to measure the effectiveness of different information extraction techniques, allowing quantitative comparisons to be made." A Text Mining Workshop was held in 2007 in conjunction with the Seventh Society of Industrial and Applied Mathematics (SIAM) Conference on Data Mining (SDM 2007). Textbooks in text mining have included applications to biology and biomedicine by Ananiadou and McNaught (2006).

Figure 1 of this paper from Liang (2003) shows the text mining process from text preprocessing to analyzing results. Saravanan et al. (2003) discuss how to automatically clean data, i.e., summarizing domain-specific information tailored to user's needs, by discovering classes of similar items that can be grouped into prescribed domains. Hersh (2005) evaluates different text-mining systems for information retrieval. Turmo et al. (2006) describe and compare different approaches to adaptive information extraction from textual documents and different machine language techniques. Amir et al. (2005) describe a new tool called maximal associations which allows the discovering of interesting associations often lost by regular association rules. Spasic et al. (2005) discuss ontologies and text mining to automatically extract information and facts, discover hidden associations and generate hypotheses germane to user needs. Ontologies specify the interpretations of terms, echo the structure of the domain, and thus can be used to

Figure 1. Text mining process source: Liang (2003)



support automatic semantic analysis of textual information. Seewald et al. (2006) describe an application for relevance assessment for multi-document summarization. To characterize certain document collections by a list of pertinent terms, they have proposed a term utility function, which allows a user to define parameters for continuous trade-off between precision and recall.

Visa et al. (2002) develop a new methodology based on prototype matching to extract the document contents. Hirsch et al. (2005) describe a novel approach for using genetic programming to create classification rules and provide a basis for text mining applications. Yang and Lee (2005) develop an approach to automatically generate category themes and reveal the hierarchical structure. Category themes and their hierarchical structures are most determined by human experts, however, with this approach; text documents can be categorized and classified automatically. Fan et al. (2005) describe a method using genetic programming to discover new ranking functions in the information-seeking task for better precision and recall. Wu at al. (2006) describe a key phrase identification program to extract document key phrases for effective document clustering, automatic text summarization, development of search engines, and document classification. Cody et al. (2004) discuss the integration of business intelligence and knowledge management based on an OLAP model enhanced with text analysis.

Trumbach (2006) uses text mining to narrow the gap between the information needs of technology managers and analysts' derived knowledge by analyzing databases for trends, recognizing emerging activity, and monitoring competitors. Srinivasan (2006) develops an algorithm to generate interesting hypotheses from a set of text collections using Medline database. This is a fruitful path to ranking new terms representing novel relationships and making scientific discoveries by text mining. Metz (2003) indicates text mining "applications are clever enough to run conceptual searches, locating, say, all the phone numbers and places names buried in a collection of intelligence communiqués. More Impressive,

the software can identify relationships, patterns, and trends involving words, phrases, numbers, and other data." Guernsey (2003) in an article that appeared in *The New York Times* stated Text-mining programs go further than Google and other Web search engines by "categorizing information, making links between otherwise unconnected documents and providing visual maps (some look like tree branches or spokes on a wheel) to lead users down new pathways that they might have been aware of."

MAIN FOCUS OF THE CHAPTER: BACKGROUND OF TEXT MINING SOFTWARE

A comprehensive list of text mining, text analysis, and information retrieval software is available on Web page of KDnuggets (2007a) and similarly for Web mining and Web usage mining software of KDnuggets (2007b). Selected software from these and other resources are discussed in this chapter.

Some of the popular software currently available for text mining include Compare Suite, SAS Text Miner, Megaputer Text Analyst, Visual Text by Text Analysis International, Inc. (TextAI), Megaputer PolyAnalyst, WordStat, and SPSS Clementine for text mining. These software provide a

Table 1. Text mining software

Features	Software	Compare Suite	SAS Text Miner	Text Analyst	Visual Text	Megaputer PolyAnalyst	WordStat	SPSS Clementine
Data Preparation	Text parsing and extraction	x	x	x	x	x	x	x
	Define dictionary			x	x	x	x	x
	Automatic Text Cleaning		x		add-on			
Data Analysis	Categorization	x			x	x	x	
	Filtering				x	x	x	
	Concept Linking		x		add-on	x		x
	Text Clustering		x	x	add-on	x	x	x
	Dimension reduction techniques		x			x		
	Natural language query			x	x			
Results Reporting	Interactive Results Window		x	x	x	x	x	
	Support for multiple languages		x		x	x	x	x
Unique features		Report Generation, compare two folders feature			Multi-path multi-paradigm analyzer		Export any table to Excel	Linguistic approach rather than statistics-based approach

variety of graphical views and analysis tools with powerful capabilities to discover knowledge from text databases. The main focus of this chapter is to compare, discuss, and provide sample output for each as visual comparisons.

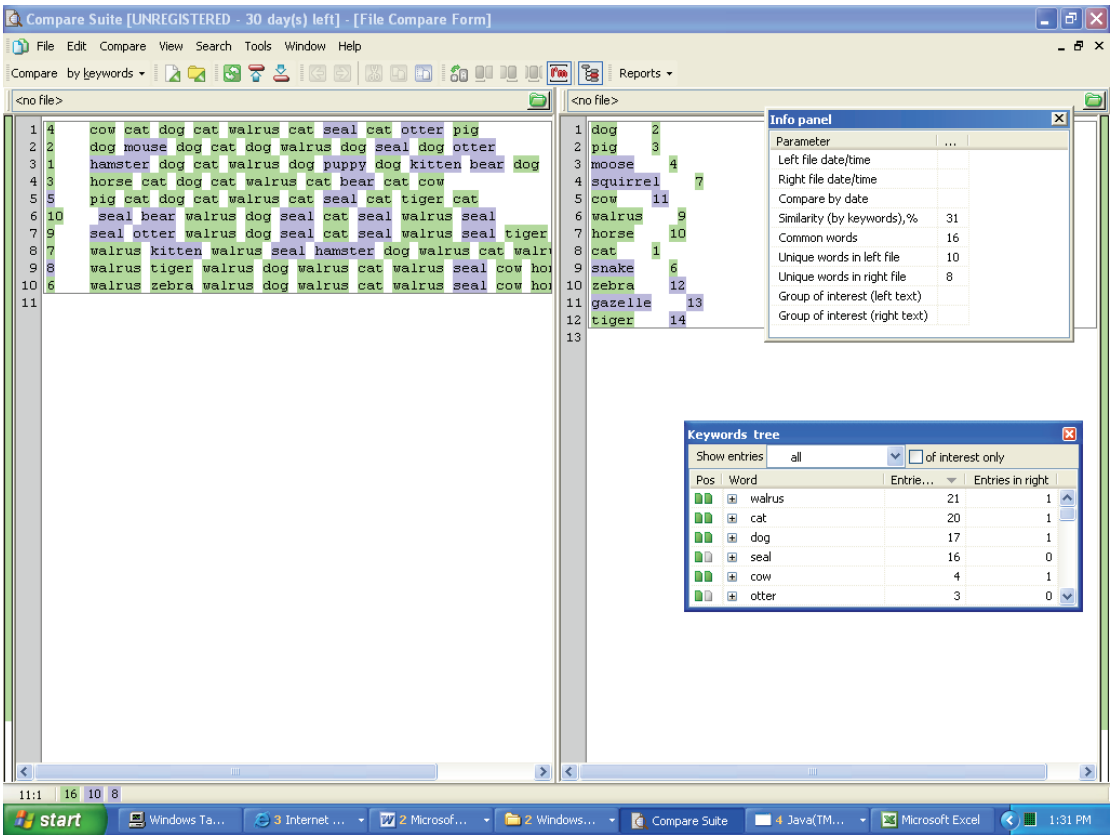
As a visual comparison of the features for these seven selected text mining software, the authors of this chapter constructed Table 1, where essential functions are indicated as being either present or absent with regard to data preparation, data analysis, results reporting, and unique features. As Table 1 shows, Compare Suite and Text Analyst have minimal text mining capabilities while Megaputer PolyAnalyst, SAS Text Miner, WordStat, and SPSS Clementine have extensive text mining capabilities. Visual Text has add-ons to make this software have versatile.

RESULTS

1. Compare Suite

Compare Suite is text mining software developed by AKS-Labs, headquarters of which are located in Raleigh, USA. The software allows comparing any to any file including formats such as text file, MS Word, MS Excel, pdf, Web pages, zip archives, and binary files. It allows comparing two files character by character, word by word, or by key words. Two folders can also be compared to find changes made and contained files. A report can be created after comparison including detailed comparison information. Documents can be compared online by server-side comparison.

Figure 2. Compare Suite with two animal text files



Compare Suite is able to provide to user a window for option of file comparison of text. Illustrated in Figure 2, two animal files are compared by words and results are reported. From the results, the same words that appear in both the files are highlighted with green color and the words that only appear in one of two files are highlighted with purple color.

2. SAS Text Miner

SAS Text Miner is actually an "add-on" to SAS Enterprise Miner with the inclusion of an extra icon in the "Explore" section of the tool bar. SAS Text Miner performs simple statistical analysis, exploratory analysis of textual data, clustering, and predictive modeling of textual data. SAS Text Miner uses the "drag-and-drop" principle by dragging the selected icon in the tool set to dropping it into the workspace.

The workspace of SAS Text Miner was constructed with a data icon of selected animal data that was provided by SAS in their Instructor's Trainer Kit. Figure 3 shows the results of using SAS Text Miner with individual plots for "role by frequency", "number of documents by frequency", "frequency by weight", "attribute by frequency", and "number of documents by frequency scatter plot." Figure 4 shows the interactive mode of SAS Text Miner, which includes the text for each document, the respective weight for each term and also the concept linking figure using SASPDFSYNONYMS text file. Figure 5 shows regression results window in SAS Text Miner

Figure 3. Results of SAS Text Miner for animal text

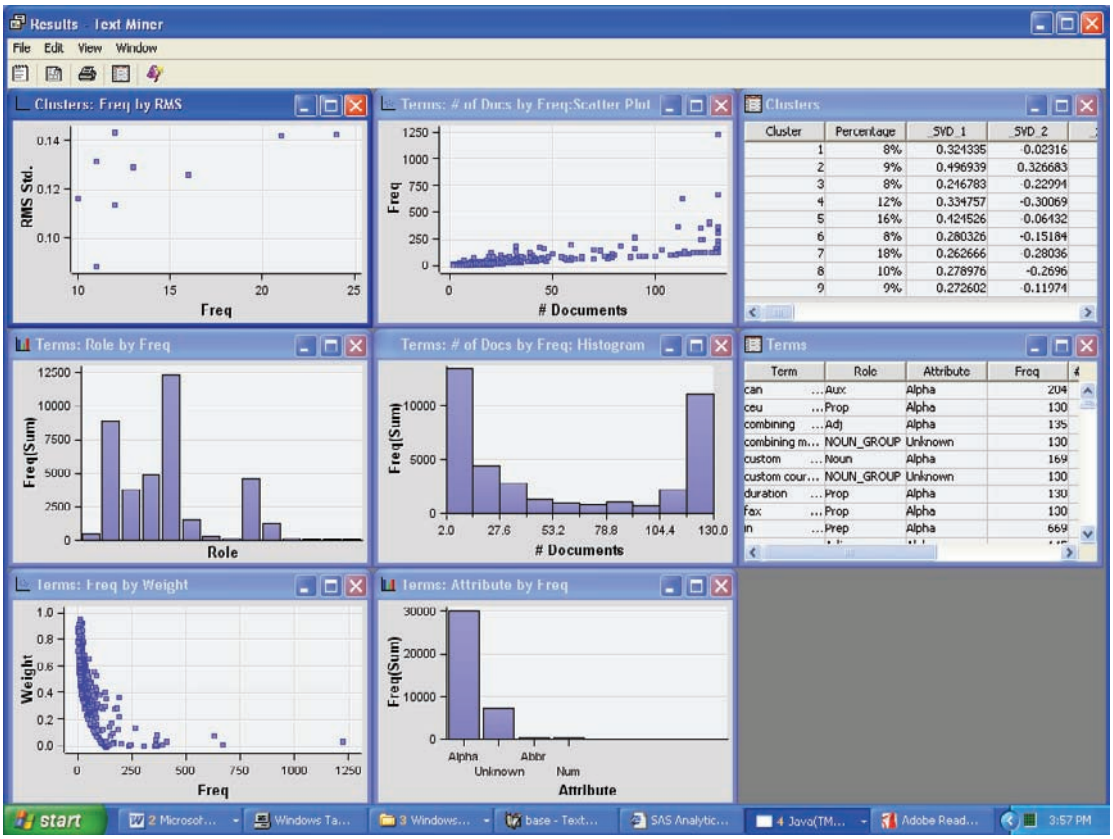


Figure 4. Interactive window of SAS Text Miner for animal text

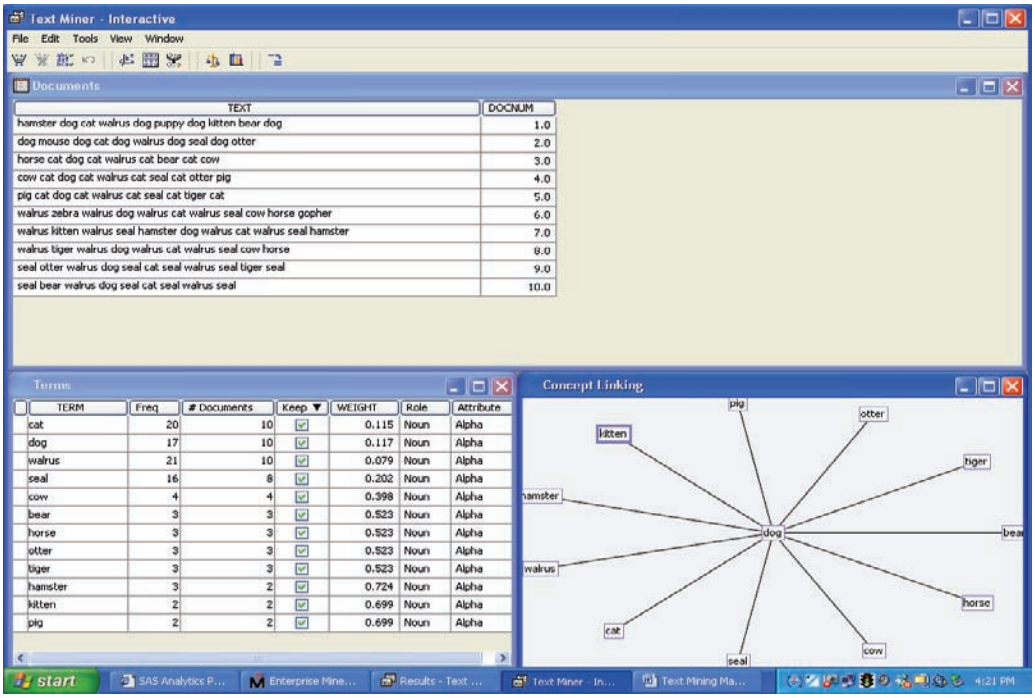
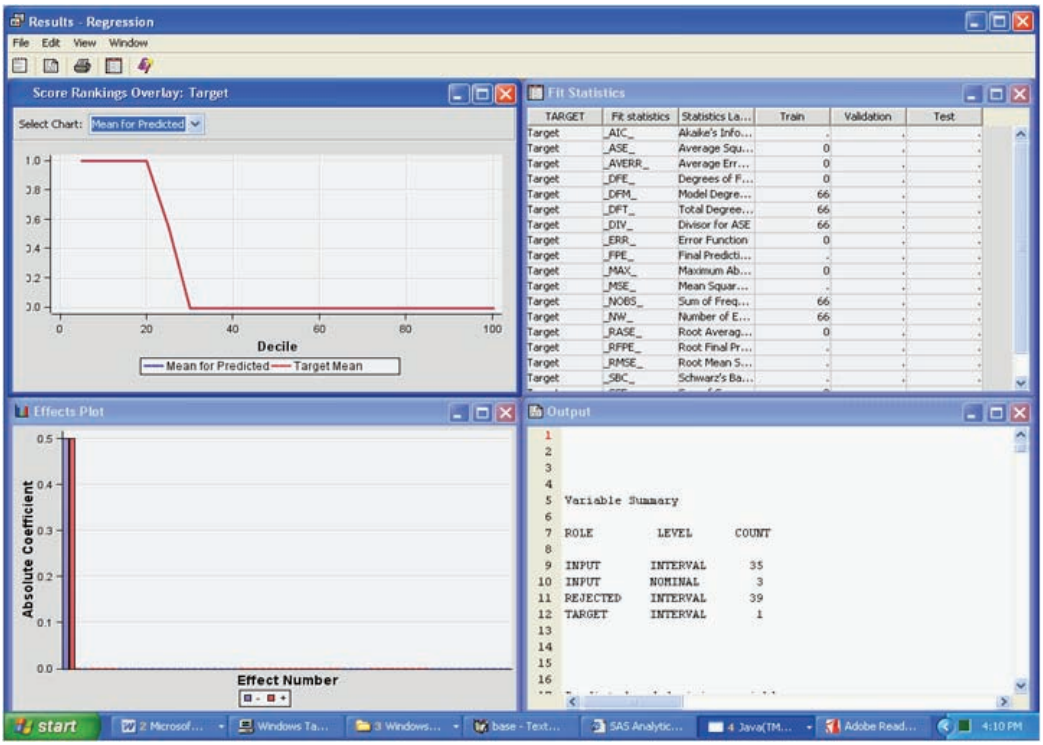


Figure 5. Regression results of SAS Text Miner using Federalists Papers



using data for *Federalists' Papers* from their instructor's trainer kit.

3. Megaputer Text Analyst

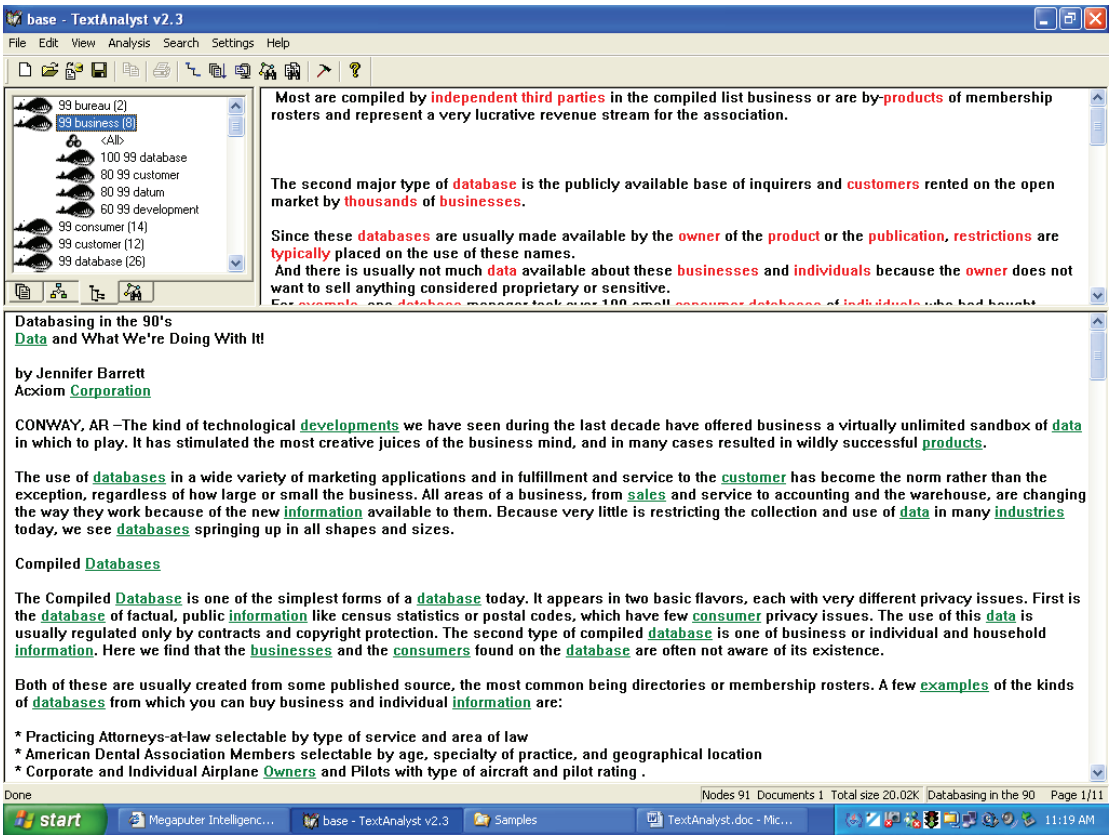
TextAnalyst has an ActiveX suite for dealing with text and semantic analysis. According to Megaputer (2007c) Web page, TextAnalyst uses a Semantic Network similar to a molecular structure, and determines the relative importance of a text concept, solely by analyzing its connection to other concepts in the text, and also implements algorithms similar to those used for text analysis in the human brain.

Figure 6 show a representative screen shot of Megaputer TextAnalyst which consists of a view pane in top left, results pane in top right, and a text pane in the bottom part of the window. The view pane shows each of the nodes in the semantic tree which each can be expanded. Megaputer Text Analyst uses a semantic search window where a query can be entered either as full sentences or questions instead of having to determine key word or phrases. A summary file in the top left pane of Megaputer Text Analyst window can list the most important sentences in the context of the original document. The Summary chooses the sentences on the basis of concepts and relationships between concepts in the full text.

4. VisualText by TextAI

VisualText by TextAI (Text Analysis International, Inc.) uses national language processing including "analyzers" for extracting information from text repositories. Some applications

Figure 6. Text Analyst semantic network window of document databasing in the 90's



include databases of resumes, Web pages, or even e-mails or Web chat databases. VisualText allows the user to create their own text analyzer, and also includes a TAI Parse for tagging parts of speech and chunking. Voice processing needs to be converted to text first before text processing can be performed.

According to VisualText Web page (2005), it can be used for combating terrorism, narcotic espionage, nuclear proliferation, filtering documents, test grading, and automatic coding. VisualText also allows natural language query, which is the ability to ask a computer questions using plain language.

Figures 7 and 8 provide screen shots of VisualText. Figure 7 shows the analyzer on the left and the root of the text zone on the right. Figure 8 shows parsing trees. A window in VisualText can show dictionary with an expansion of the root of the text zone on the right part of this window.

5. Megaputer PolyAnalyst

Previous work by the authors Segall and Zhang (2009) have utilized Megaputer PolyAnalyst for data mining. The new release of PolyAnalyst version 6.0 includes text mining and specifically new features for Text OLAP (on-line analytical processing) and Taxonomy-based categorization which is useful for when dealing with large collections of unstructured documents as discussed in Megaputer Intelligence Inc. (2007). The latter cites that taxonomy-based classifications are useful when dealing with large collections of unstructured documents such as tracking the number of known issues in product repair notes and customer support letters.

Figure 7. Screen-shot of Visual Text window

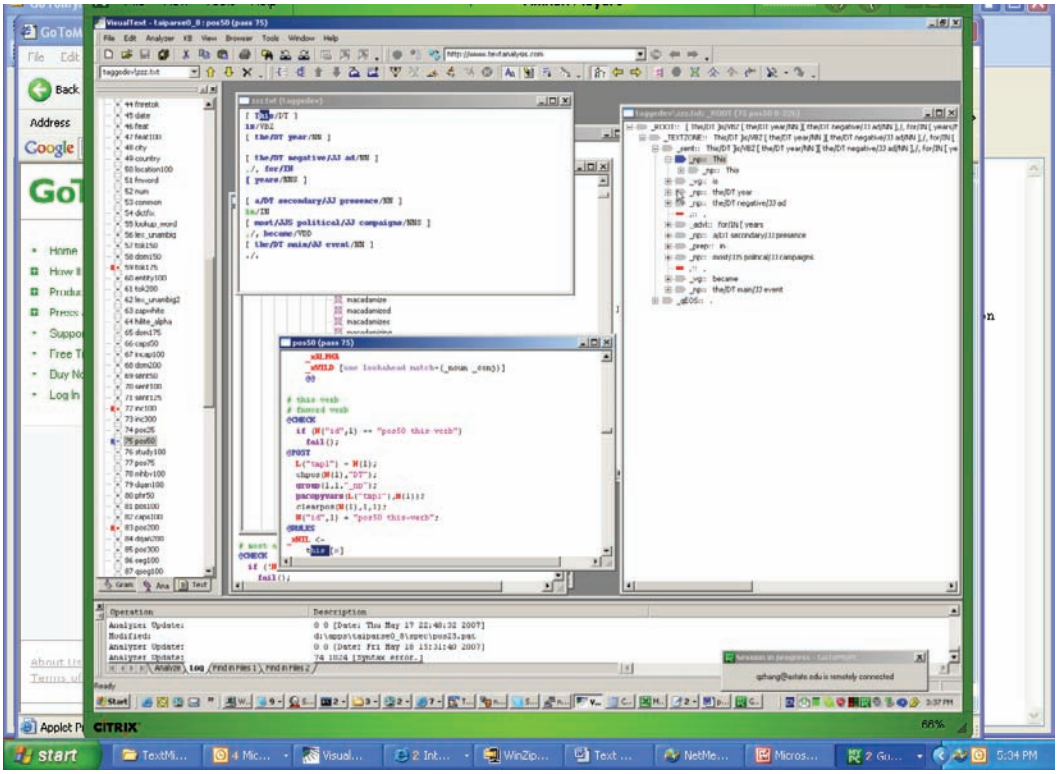
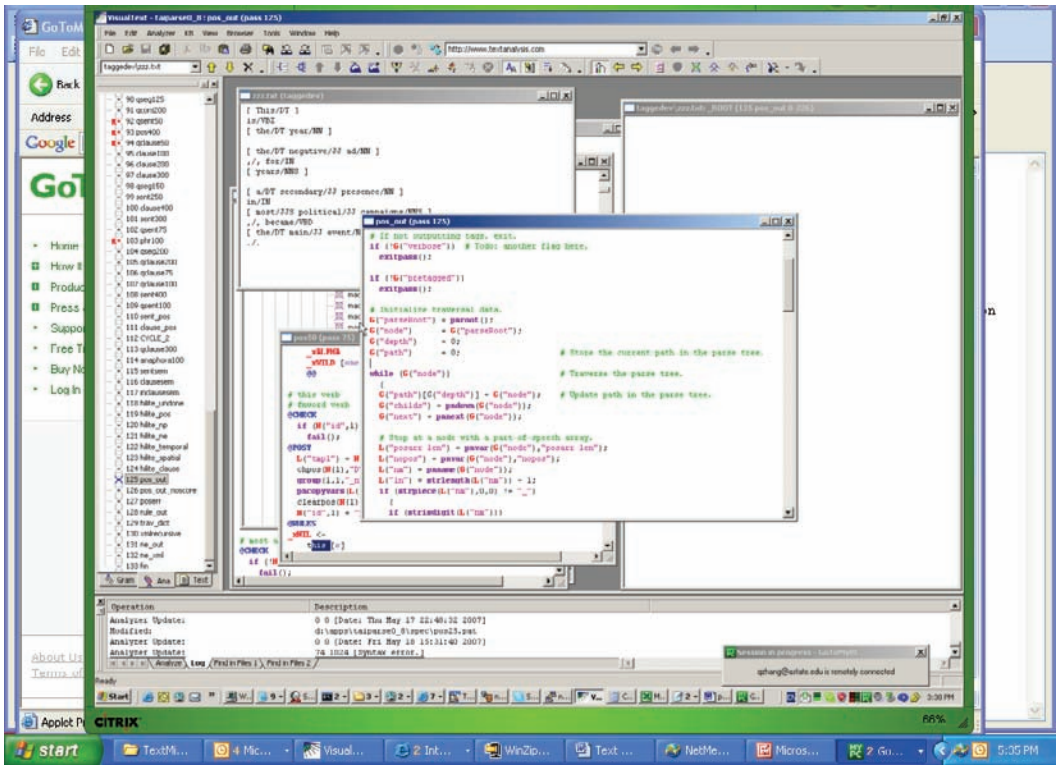


Figure 8. Parsing trees in Visual Text



According to Megaputer Intelligence Inc. (2007), PolyAnalyst "provides simple means for creating, importing, and managing taxonomies, and carries out automated categorization of text records against existing taxonomies." Megaputer Intelligence Inc. (2007) provides examples of applications to executives, customer support specialists, and analysts. According to Megaputer Intelligence Inc. (2007), "executives are able to make better business decisions upon viewing a concise report on the distribution of tracked issues during the latest observation period."

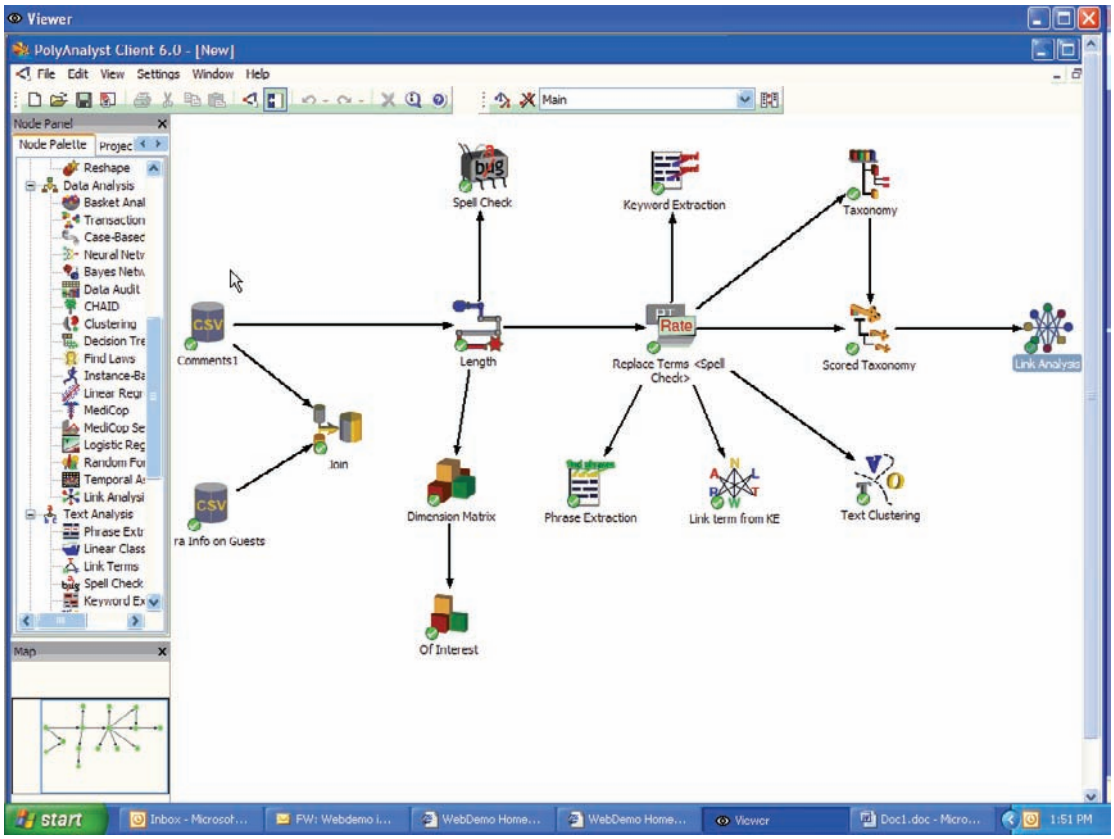
This chapter provides several figures of actual screen shots of Megaputer PolyAnalyst version 6.0 for text mining. These are Figure 9 for workspace of text mining of Megaputer PolyAnalyst, Figure 10 for key word extraction window for the word "breakfast" from customer written comments, and Figure 11 for initialization of link term report. Megaputer PolyAnalyst can also provide screen shots with drill-down text analysis and histogram plot of text analysis.

6. WordStat

WordStat is developed by Provalis Research. It is a text analysis software module run on a base product of SimStat or QDA Miner. It can be used to study textual information such as interviews, answers to open-ended questions, journal articles, electronic communications, and so on. WordStat may also be used for categorizing text automatically using a dictionary approach, or for developing and validating new categorization dictionaries or taxonomies.

WordStat incorporates many data analysis and graphical tools that can be used to explore relationships between document contents and

Figure 9. Workspace for text mining in Megaputer PolyAnalyst



information amassed in categorical or numeric variables. Hierarchical clustering and multidimensional scaling analysis can be used to identify relationships among categories and document similarity. Correspondence analysis and plots can be used to explore relationships between keywords and different groups.

An input file (e.g., excel file) can be imported into the software for analysis. An important preliminary to WordStat analysis is to create a categorization dictionary (which needs domain knowledge). WordStat analysis consists of many tabulations or cross-tabulations of different categories. Figure 12 shows a screen shot of a categorization dictionary window. Correspondence analysis results and 3-D plots by WordStat are illustrated in Figure 13.

7. SPSS Clementine

Text mining for Clementine is text mining software developed by SPSS Inc. It can be used to extract concepts and relationships from textual data. It can also be used to convert an unstructured format to a structured one for creating predictive models. Text mining for Clementine can be accessed directly from the interface of SPSS Inc.'s leading data mining software "Clementine".

Text mining for Clementine can process many types of unstructured data, including texts in MS office files, survey text responses, call center notes, Web forms, and Web logs, blogs, Web feeds, streams, and so on. It uses a natural language processing (NLP) linguistic extraction process. It has a graphical interface and is easy to

Figure 10. Keyword extraction window of text analysis in Megaputer PolyAnalyst

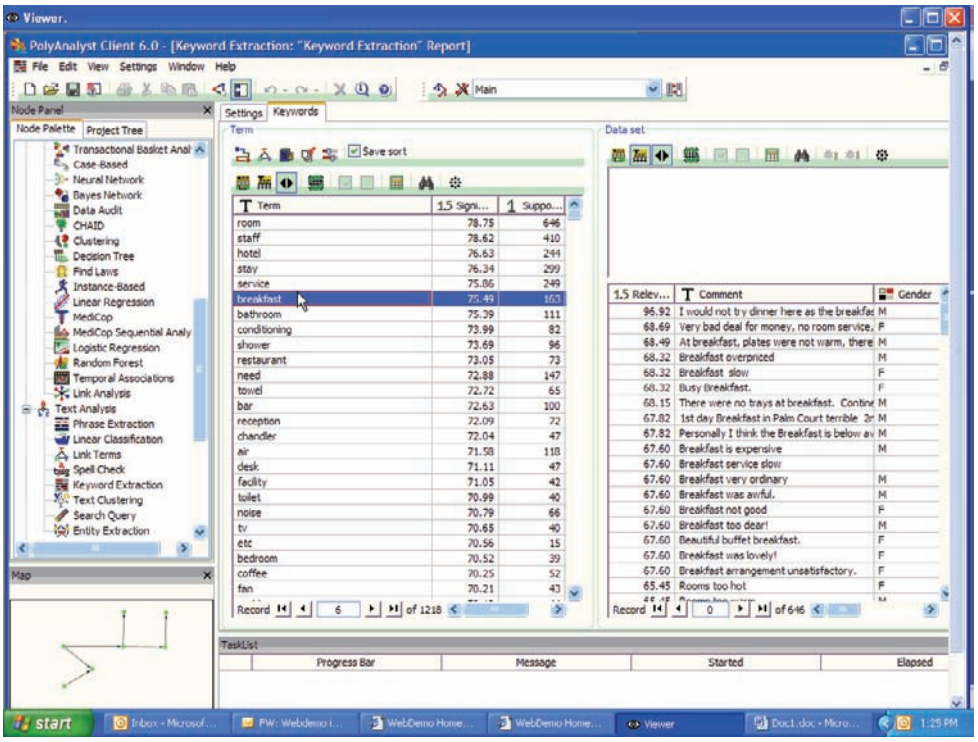


Figure 11. Initialization of link term report in Megaputer PolyAnalyst

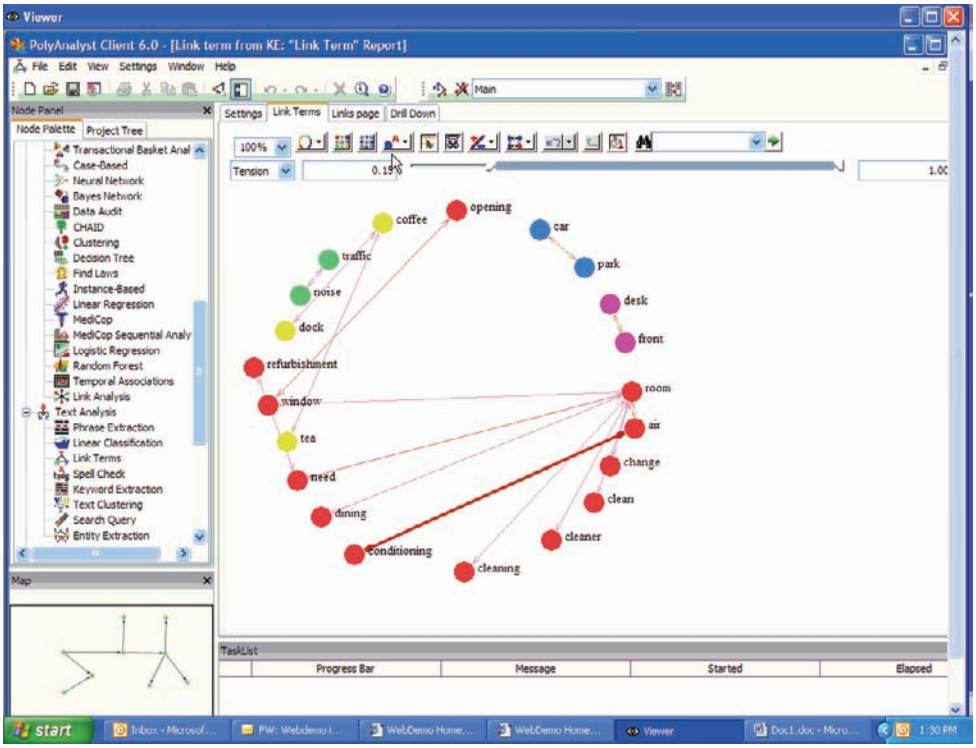


Figure 12. Screen shot of categorization dictionary window using WordStat (source: http://www.provalisresearch.com/WordStat/WordStatFlashDemo.html)

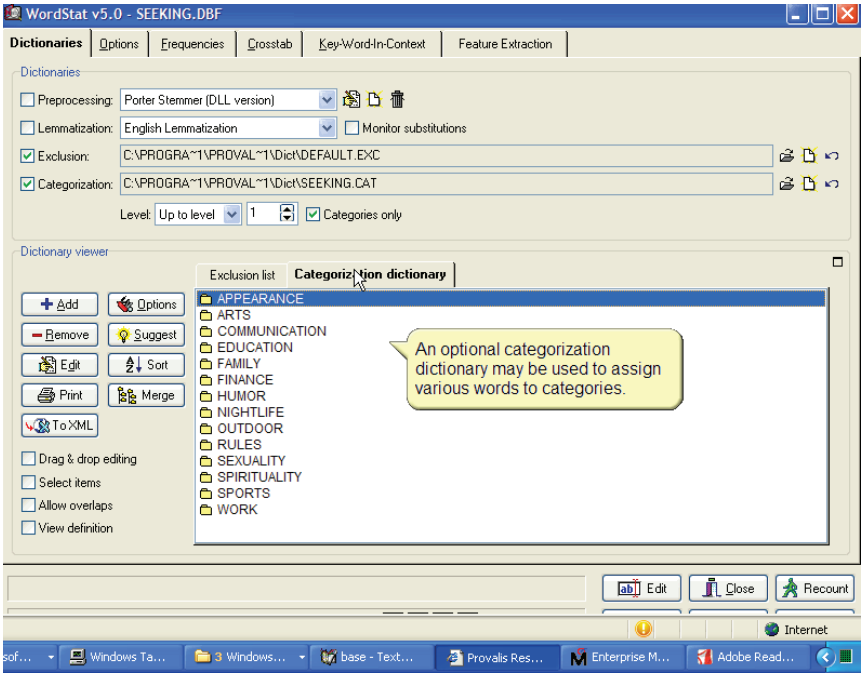


Figure 13. 3-D View of correspondence analysis results using WordStat (source: http://www.provalisresearch.com/WordStat/WordStatFlashDemo.html)

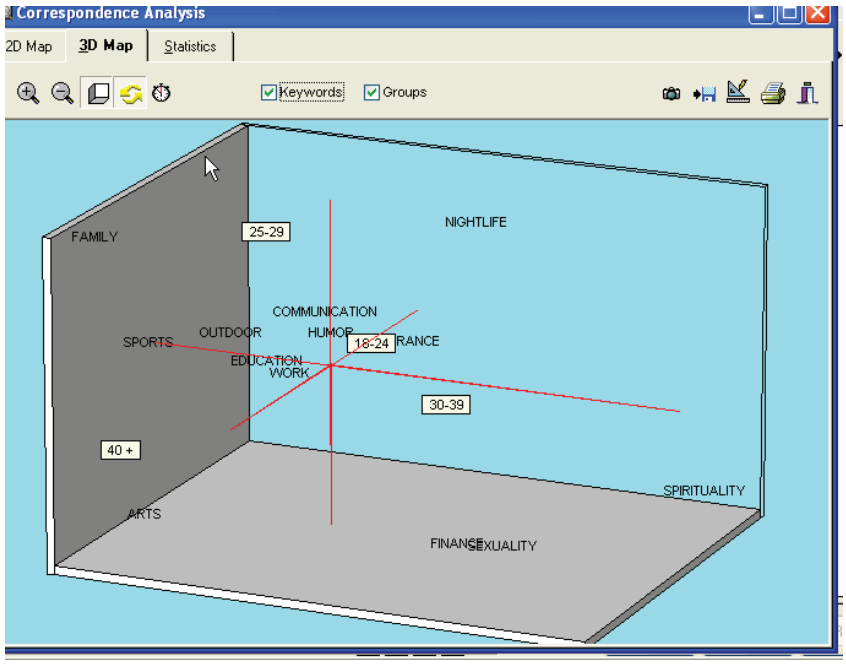
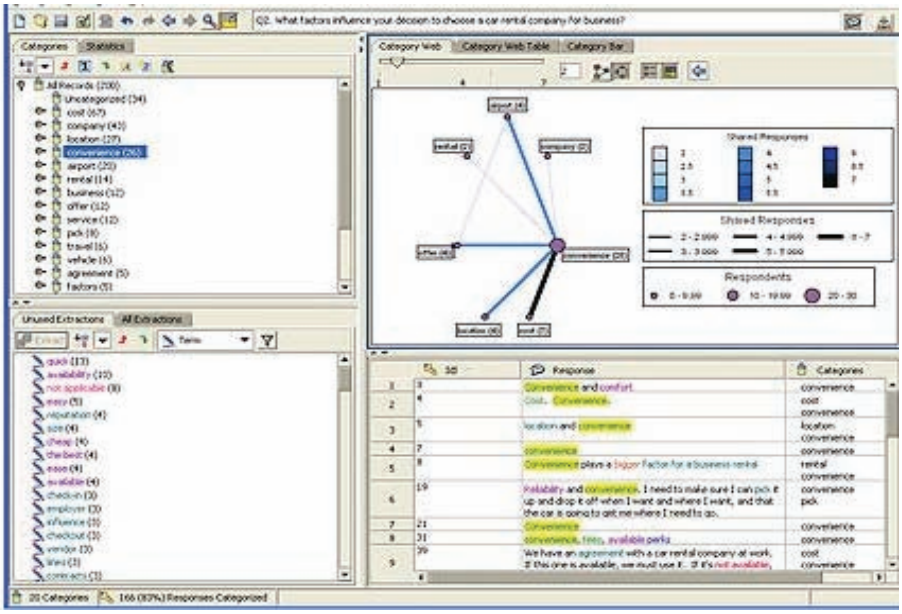


Figure 14. Screen shot of SPSS Clementine for text analysis (Source: http://www.spss.com/textanalysis_surveys/demo.htm)



use. Dictionaries can be customized for specific domain areas by using the built-in Resource Editor. It allows extracting text from **multiple languages** such as Dutch, English, French, German, Italian, Portuguese, or Spanish. It can also process text translated into English from 14 languages including Arabic, Chinese, Persian, Japanese, and Russian. Figure 14 shows a screen shot of category analysis using SPSS Clementine.

FUTURE TRENDS/CONCLUSION

This chapter has shown that software for text mining is a new and expanding area of technology that numerous vendors are in competition with each other in providing both unique and common features. Users needing to use text mining are fortunate to have such resources available for needs that were unthinkable a decade ago. Future trends are that text mining software will continue to grow in dimensionalities of features and available software. The applications of software for text mining will be extremely diverse ranging from uses in customer survey responses to drill-downs in medical records or credit or bank reports.

A future direction of this work is to pursue the area of Web mining software and contrast with that of text mining. Web mining is according to the TMRG (Text Mining Research Group) Web site defined to be "the slightly more general case of looking for patterns in hypertext and often applies graph theoretical approaches to detect and utilize the structure of Web sites." Web mining entails mining of content, structure, and usage. Web content mining entails both Web page content mining and search result mining. Web structure mining uses interconnections between Web pages to give weight to pages. Web usage mining is the application that uses data mining to analyze and discover interesting patterns of user's usage of data on the Web.

Some of the popular software that could be selected for Web mining for future comparison include Megaputer WebAnalyst, SAS Web Analytics, SPSS Web Mining for Clementine, WebLog Expert 2.0, and Visitator.

ACKNOWLEDGMENT

The authors would like to acknowledge funding for support of this research from the Summer Faculty Research Grant as awarded to both authors from the College of Business at Arkansas State University. The authors would also like to acknowledge their gratefulness to those at SAS Inc. for the Mini-Grant provided for SAS Text Miner, TextAI and Megaputer Inc. for their extremely helpful technical support.

REFERENCES

Allan, J., Kumar, V., and Thompson, P. (2000). *Institute for Mathematics and Its Applications (IMA). IMA Hot Topics Workshop: Text Mining.* April 17-18, 2000, University of Minnesota, Minneapolis, MN.

Amir, A., Aumann, Y., Feldman, R. and Fresko, M., (2005). Maximal association rules: a tool for mining associations in text. *Journal of Intelligent Information Systems*, *25*(3), 333–345.

Ananiadou, S. and McNaught, J. (eds) (2006). *Text Mining for Biology and Biomedicine,* Artech House Publishers, Boston, MA. ISBN 1-58053-984-X.

Ananiadou, S., Chruszcz, J., Keane, J., McNight, J., Watry, P. (2005). *Ariadne, 42.* Retrieved January 2005, from http://www.ariadne.ac.uk/issue42/ananiadou

Baker, C. and Witte, R., (2006). Mutation mining - A prospector's tale. *Information Systems Frontier, 8,* 47–57.

Cerrito, P.B., Badia, A., and Cox, J., (2003). The Application of Text Mining Software to Examine Coded Information. *Proceedings of SIAM International Conference on Data Mining,* San Francisco, CA, May 1-3.

Cody, W., Kreulen, J., Krishna, V., and Spangler, W., (2002). The integration of business intelligence and knowledge management. *IBM Systems Journal, 41*(4), *697-713.*

Fan, W. P. (2006). *Text Mining, Web Mining, Information Retrieval and Extraction from the WWW References.* Retrieved from http://filebox.vt.edu/users/wfan/text_mining.html

Fan, W., Gordon, M., and Pathak, P., (2005). Genetic programming-based discovery of ranking functions for effective Web search. *Journal of Management Information Systems*, *21*(4), 37-56.

Firestone, J., (2005). Mining for information gold. *Information Management Journal*, 47-52.

Grobelnik, M. and Mladenic, D. (n.d.) *Text-Garden — Text-Mining Software Tools.* Retrieved from http://kt.ijs.si/Dunja/textgarden/

Guernsey, L. (2003). Digging for Nuggets of Wisdom. *The New York Times,* October 16, 2003.

Hayes, J. H., Dekhtyar, Sundaram, S., (2005). Text Mining for Software Engineering: How Analyst Feedback Impacts Final Results. International Conference on Software Engineering: *Proceedings of the 2005 International Workshop on Mining Software Repositories,* St Louis, MO, May.

Hearst, M. A. (1999). Untangling Text Data Mining, School of Information Management & Systems, University of California at Berkeley. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL),* June 20-26.

Hearst, M. A. (2003). *What is Data Mining?* Retrieved from http://www.ischool.berkeley.edu/~hearstr/text_mining.html

Hersh, W., (2005). Evaluation of biomedical text-mining systems: lessons learned from information retrieval. *Briefings in Bioinformatics*, *6*(4), 344-356.

Hirsch, L., Saeedi, M., and Hirsch, R. (2005). Evolving text classification rules with genetic programming. *Applied Artificial Intelligence*, *19,* 659–676

Jin, X., Zhou. Y., and Mobasher, B. (2004). Web usage mining based on probabilistic latent semantic analysis, Conference on Knowledge Discovery in Data. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 197-205.

KDNuggets (2005). *Polls: Text Mining Tool*s. Retrieved January, 2005 from http://www.kdnuggets,com/polls/2005/text_mining_tools.htm

KDNuggets (2007). *Text Analysis, Text Mining, and Information Retrieval Software*. Retrieved from http:///www.kdnuggets.com/software/text.html

Kloptchenko, A., Eklund, T., Karlsson, J., Back, B. and Vanhar, H., (2004). Combining data and text mining techniques for analysing financial reports. *Intelligent Systems in Accounting, Finance and Management, 12(1),* 29-41.

Kostoff, R., Rio, J., Humenik, J., Garcia, E., and Ramirez, A., (2001). Citation mining: Integrating text mining and bibliometrics for research user. *Journal of the American Society for Information Science and Technology*, *52*(13), 1148-1156.

Leon, D.A., (2007). Using text mining software to identify drug, compound, and disease relationships in the literature. In *Proceedings of 233rd ACS National Meeting*, Chicago, IL, March 25-29. Retrieved from http://acsinf.org/docs/meetings/233nm/abs/CINF/073.htm

Liang, J.W.(2003). Introduction to Text and Web Mining, Seminar at North Carolina Technical University. Retrieved from http://www.database.cis.nctu.edu.tw/seminars/2003F/TWM/slides/p.ppt

Lieberman, H.(2000). *Text Mining in Real Time, Talk Abstract*. Retrieved from http://www.ima.umn.edu/reactive/abstract/lieberman1.html

Lynn, A. (2004). *Mellon grant to fund project to develop data-mining software for libraries*. Retrieved from http://www.news.uiuc.edu/news/04/1025mellon.html

Mack, R., Mukherjea, S., Soffer, A., and Uramoto, N., et al., (2004). Text analytics for life science using the unstructured information management. *IBM Systems Journal*, *43*(3), 490-515.

McCallum, A. (1998). *Bow: A Toolkit for statistical language modeling, Text Retreival, Classification and Clustering.* Retrieved from http;//www.cs.cmu.edu/~mccallum/bow/

Megaputer Intelligence Inc. (2000). *Tutorial: TextAnalyst Introduction*. Retrieved from http://www.megaputer.com/products/ta/tutorial/textanalyst_tutorial_1.html

Megaputer Intelligence Inc. (2007). *Data Mining, Text Mining, and Web Mining Software*. Retrieved from http:///www.megaputer.com

Megaputer Intelligence Inc. (2007). *Text OLAP*. Retrieved from http://www.megaputer.com/products/pa/algorithms/text_oplap.php3

Megaputer Intelligence Inc. (2007). *TextAnalyst*. Retrieved from http://www.megaputer.com/products/ta/index.php3

Megaputer Intelligence Inc. (2007). *WebAnalyst, Benefits of Web Data Mining*. Retrieved from http://www.megaputer.com/products/wa/benefits/php3

Megaputer Intelligence Inc. (2007). *WebAnalyst, Introduction to Web Data Mining*. Retrieved http://www.megaputer.com/products/wa/intro.php3

Metz. C. (2003). Software: Text mining. *PC Magazine*. Retrieved from http://www.pcmag.com/print_article2/0,1217.a=43573,00.asp

Miller, T. M. (2005). *Data and Text Mining: A Business Applications Approach*. Pearson Prentice Hall, Upper Saddle River, NJ.

Rajman, M. and Besann, R. (1997). Text Mining: Natural Language Techniques and Text mining Applications. *Proceedings of the 7th IFIP2.6 Working Conference on Database Semantics (DS-7)*, Leysin, Switerland, October 1997.

Robb, D., (2004). Text Mining tools take on unstructured data. *Computerworld*, June 21

Romero, C. and Ventura, S., (2007). Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications*, 33, 135–146.

Saravanan, M., Reghuraj, C., and Raman, S. (2003). Summarization and categorization of text data in high-level data cleaning for information retrieval. *Applied Artificial Intelligence*, 17, 461–474.

SAS Raises Bar on Data, Text Mining (2005). *GRID Today*. Retrieved August 29, 2005 from www.gridtoday.com/grid/460386.html

Scherf, M., Epple, A., and Werner, T. (2005). The next generation of literature analysis: integration of genomic analysis into text mining. *Briefings in Bioinformatics*, *6*(3), 287-297.

Seewald, A., Holzbaur, C., and Widmer, G., (2006). Evaluation of term utility functions for very short multidocument summaries. *Applied Artificial Intelligence*, *20*, 57–77.

Segall, R.S. and Zhang, Q. (2009). Comparing four-selected data mining software. *Encyclopedia of Data Warehousing and Mining*, Chapter XLV, Edited by Jon Wang. IGI Global, Inc., 269-277.

Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 20-23, 2000, Boston, MA.

Slynko, Y. and Ananyam, S. (2007). *WebAnalyst Server – universal platform for intelligent e-business*. Retrieved from http://www.megaputer.com/tech/wp/wm.php3

Spasic, I., Ananiadou, S., McNaught, J., and Kumar, A., (2005). Text mining and ontologies in biomedicine: making sense of raw text. *Briefings in Bioinformatics*, *6*(3), 239-251.

Srinivasan, P. (2004). Text mining: Generating hypotheses from MEDLINE. *Journal of the American Society for Information Science and Technology*, *55*(5), 396-413.

Survey of Text mining: Clustering, Classification, and Retrieval, Springer-Verlag.

Text mining (2007). Held in conjunction with Seventh SIAM International Conference on Data Mining, Minneapolis, MN, April 28, 2007.

Text mining Research Group at the University of Waikato (2002). *Text mining, Computer Science Department*. University of Waikato, Hamilton, New Zealand. Retrieved from http://www.cs.waikato.ac/nz/~nzdl/textmining

The Lemur Project (2007). *The Lemur Toolkit for Language Modeling and Information Retrieval*. Retrieved from http://www.lemurproject.org/index.php?version=print

Trumbach, C., (2006). Addressing the information needs of technology managers: making derived information usable. *Technology Analysis & Strategic Management*, *18*(2), 221–243.

Turmo, J., Ageno, A., and Catala, N., (2006). Adaptive Information Extraction. *ACM Computing Surveys*, *38*(2), 1-47.

Uramoto, N., Matsuzawa, H, Nagano, T., Murakami, A., Takeuchi, H., and Takeda, K., (2004). A text-mining system for knowledge discovery from biomedical documents. *IBM Systems Journal*, *43*(3), 516-533.

Uramoto, N., Matsuzawa, H., Nagano, T., Murakami, A., Takeuchi, and Ta, K, (2004). *Unstructured Information Management*, *43*(3).

Visa, A., Toivonen, J., Vanharanta, H., and Back, B., (2002). Contents matching defined by prototypes: Methodology verification with books of the bible. *Journal of Management Information Systems, 18*(4), 87-100.

Weiss, S. M., Indurkhya, N., Zhang,T, and Damerau, F. (2005). *Text mining: Predictive Methods for Analyzing Unstructured Information*, Springer Press

Woodfield, Terry (2004). *Mining Textual Data Using SAS Text Miner for SAS9 Course Notes*, SAS Institute, Inc., Cary, NC.

Wu, Y., Li, Q., Bot, R., and Chen, X., (2006). Finding Nuggets in Documents: A Machine Learning Approach, *Journal of the American Society For Information Science And Technology, 57*(6), 740–752.

Yang, H. and Lee, C., (2005). Automatic Category Theme Identification and Hierarchy Generation for Chinese Text Categorization, *Journal of Intelligent Information Systems, 25*(1), 47–67.

KEY TERMS

Compare Suite: AKS Labs software that compares texts by keywords, highlights common and unique keywords.

Megaputer TextAnalyst: Software that offers semantic analysis of free-form texts, summarization, clustering, navigation, and natural language retrieval.

Natural Language Processing: Also know as computational linguistics that is, using natural language, e.g., English, to do query or search**.**

SAS Text Miner: Software by SAS Inc. that provides a suite of text processing and analytical tools.

SPSS Mining for Clementine: Enables you to extract key concepts, sentiments, and relationships from call center notes, blogs, e-mails and other unstructured data, and convert it to structured format for predictive modeling.

Text Mining: Discovery by computer of new, previously unknown information by automatically extracting information from different written resources.

Visual Text: Software manufactured by TextAI that uses a comprehensive GUI development environment (www.textanalysis.com)

WordStat: Analysis module for textual information such as responses to open-ended questions, interviews, etc.

This work was previously published in Handbook of Research on Text and Web Mining Technologies, edited by M. Song & Y. Wu, pp. 766-784, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 3.5
A Software Tool for Reading DICOM Directory Files

Ricardo Villegas
Universidad de Carabobo, Venezuela

Guillermo Montilla
Universidad de Carabobo, Venezuela

Hyxia Villegas
Universidad de Carabobo, Venezuela

ABSTRACT

DICOMDIR directory files are useful in medical software applications because they allow organized access to images and information sets that come from radiological studies that are stored in conformance with the digital imaging and communication in medicine (DICOM) standard. During the medical application software development, specialized programming libraries are commonly used in order to solve the requirements of computation and scientific visualization. However, these libraries do not provide suitable tools for reading DICOMDIR files, making necessary the implementation of a flexible tool for reading these files, which can be also easily integrated into applications under development. To solve this problem, this work introduces an object-oriented design and an open-source implementation for such reading tool. It produces an output data tree containing the information of the DICOM images and their related radiological studies, which can be browsed easily in a structured way through navigation interfaces coupled to it.

INTRODUCTION

The digital imaging and communications in medicine (DICOM) standard (National Electrical Manufacturers Association (NEMA0, 2004a; Revet, 1997) was published in 1993. Its main goal was to establish norms for handling, storing, and interchanging medical images and associated digital information within open systems. Also it was to facilitate the interoperability among acquisition

equipments and other medical devices, as well as their integration within specialized information systems in the medical and health care area.

Since then, the appearance and use of computer-assisted medical applications have increased, as a result of the accelerated technological development and the standardization process of medical information representation and handling, which generated a greater demand of development tools for those applications.

These applications range from health care information systems and picture archiving and communication systems (PACS) () solutions, to technological support systems for medical procedures, such as image-based diagnosis and surgical planning, which previously depended on the knowledge and expertise of the physicians.

In such applications, the handling of images coming from different acquisition modalities is essential. These images generated from radiological studies and stored according to the specifications of parts 10, 11 and 12 of the DICOM standard (NEMA, 2004e, f,& g) must be retrieved from storage media as a bidimensional display or in tridimensional reconstructions and other special processes, such as fusion and segmentation of images. The use of DICOMDIR directory files is almost mandatory for searching, accessing, and browsing medical images because they index the files belonging to the patient on whom the studies were performed, thus making it easier to access to those images and their associated medical information.

During the medical application software development, the use of programming interfaces (APIs) or class libraries is frequent in order to solve the computation and visualization needs, as well as for providing DICOM support to the applications. In that sense, there exist numerous public domain applications that can be used by radiologists and other specialists for reading and displaying DICOM images files and even for reading DICOMDIR index files, which cannot be integrated into applications under development because of their proprietary code.

Companies, such as Lead Technologies, ETIAM, Merge, Laurel Bridge, and DeJarnette, have commercial software development kits (SDKs) that provide complete implementations of the DICOM standard, but the acquisition costs for these SDKs are high. Open-source libraries are an alternative choice for integrating DICOM support into applications. Regarding this matter, libraries, such as visualization tool kit (VTK) (), insight segmentation and registration tool kit (ITK), DICOM tool kit (DCMTK), and virtual vision machine (VVM), allow the reading of DICOM images, but they do not provide mechanisms for reading DICOMDIR files. Like in the DCMTK case, there are other libraries that provide tools for a basic and low-level access to the information contained in the files. However, they have disadvantages, such as troublesome information retrieving process and reading tools, which are difficult to integrate into the applications.

Due to the lack of an adequate tool for reading and handling DICOMDIR files in a structured and simple way, which could be also easily coupled to browsing interfaces and attached to medical application under development, we introduce in this article the design and implementation of a DICOMDIR files reader. This tool has been successfully integrated into an application for neurosurgery preoperative planning (Montilla, Bosnjak, Jara, & Villegas, 2005), but it also can be attached to any other software under development that requires the handling of DICOM images and DICOMDIR directory files.

The next sections include the revision of related works, the essential theoretical background that frames this work within the DICOM standard context; the description of the methodology used for the implementation of the tool; and, finally, the discussion and conclusions obtained from the integration and test of the implemented reader into a medical application.

ANTECEDENTS AND RELATED WORKS

The creation of the American College of Radiology (ACR)-NEMA committee in 1983 was the product of earlier attempts by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to establish a normative for exchanging, transmitting, and storing medical images and their associated information. The version 1.0 of the standard was published by this joint committee in 1985, under the document ACR-NEMA No. 300-1985, followed in 1988 by the document ACR-NEMA No. 300-1988 of version 2.0. Previous to these normatives, medical images were stored in files by the acquisition devices under their own proprietary formats and transferred through point-to-point communication or by removable storage media. Versions 1.0 and 2.0 established a standardized terminology and information structure, as well as hardware interfaces, software commands sets, and consistent data formats.

The most recent version, known as DICOM 3.0 (NEMA, 2004a) was published in 1993, and it was structured in parts, or documents, to facilitate its support and extension. In the last version, objects for the representation of patients, studies, reports, and other data sets were added, as well as unique identifiers for these objects, enabling the transmission of information through communication networks using the TCP/IP protocol. The DICOM 3.0 standard facilitates the transfer of images and related information within open systems containing different medical equipments and the integration with medical information systems and applications.

DICOM parts 3, 6, 10, and 12 (NEMA, 2004b, d, 4e, & g) were of particular interest for our tool design. They define and describe the DICOMDIR directory object and other information objects; attributes and representation values of DICOM information model entities; and specifications for files formats and information storage in physical media.

We have not found formal research papers related to the design and implementation of open-source tools for reading DICOMDIR files and their integration within medical applications. Nevertheless, there exist documented development libraries that include support for this kind of file. On the other hand, regarding the complexity of searching and decoding information contained in DICOM files and the fact that the tool had to be integrated into a medical application developed with C++ language, we decided to search for open-source development libraries, based upon C++ and with DICOM support in order to use them as a base for the tool development.

Just a few development libraries, besides the expensive commercial ones, enable the reading and analysis of DICOMDIR files. Due to their structures or features only three open-source libraries, based upon C++ language, were deemed appropriate to be used as reference for the tool design and implementation; the remaining APIs were found either to have a complicated structure or were based upon Java language.

GDCM (GDCM, 2005) is an API supported by Centre de Recherche et d'Applications en Traitement de l'Image et du Signal (CREATIS), which provides a fairly complete support for reading DICOMDIR files and a simple access to the information extracted from them. However, it implements only part 5 of the DICOM standard; therefore, it would be of little use as a base library for medical applications that require the implementation of other features defined by the standard.

Dicomlib library (DicomLib, 2005) provides a fuller implementation of the DICOM standard, and it also features the reading of DICOMDIR files. Although this library tries to ease the huge intrinsic complexity in the use of the DICOM standard, its access and presentation of the DICOMDIR compiled information is not the best one to be integrated into the applications.

Finally, DICOM tool kit (DCMTK) (DCMTK, 2005) from Oldenburger Forschungs und Entwicklungsinstitut für Informatik-Werkzeuge und Systeme (OFFIS) is another complete library, having several years of evolution and continous use in medical applications development. Although DCMTK provides support for the creation, modification, and opening of DICOMDIR files, it does not offer structured and simple access to the information gathered from the files. However, we selected DCMTK as the base library for the reader development due to its robustness, flexibility, and ability to handle of DICOM images.

Thus, starting from the basic functions for information searches and decoding that provides DCMTK, it was possible to develop the tool for reading the information contained in DICOMDIR files and attach this tool to medical applications that require organized access to DICOM image sets from medical studies.

THEORETICAL BACKGROUND

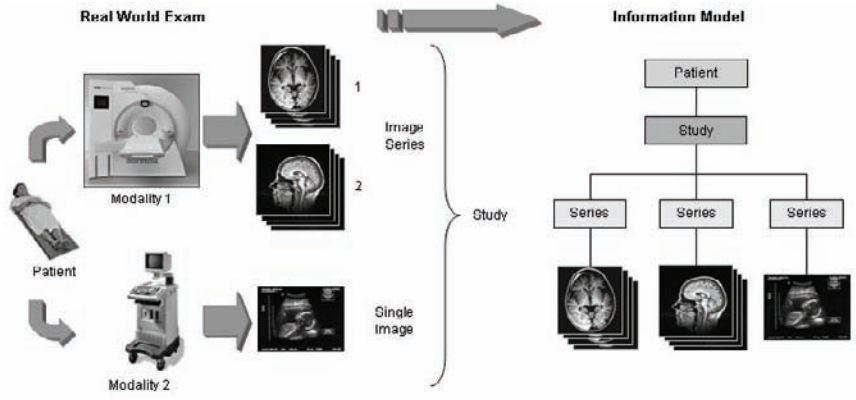
A lot has been written about the DICOM standard ever since it was published, including revisions and extensions. The scope of DICOM is so wide that the researcher certainly gets overwhelmed by the amount and complexity of the information contained in the standard documents. In our case, as in any other case of software development that involves the handling of information within the DICOM scope, the revision of basic fundamentals for real-world information representation according to the DICOM standard was necessary.

Within DICOM's structure, the term "information" refers to medical images coming from different acquisition modalities, signals, curves, look-up tables, and structured reports, as well as to other information gathered during patient visits to the healthcare specialist and the studies derived from them. This information and its generating agents are represented by the standard through models.

DICOM Application and Information Models

DICOM structures and organizes medical data and information through models that emulate the real-world hypothetical situation, where a patient visits a health care specialist, who later orders a set of radiological studies as shown in Figure 1.

Figure 1. Correspondence between the radiological exam environment and the DICOM Information Model



An application model is defined as an entity/relationship diagram (see Figure 2) that relates real-world objects inside the standard scope. Its diagram derives from the way hospitals' radiology departments handle images from one or more acquisition modalities. Those images are ordered in a series, according to some spatial or temporal relationship, and then stored in a folder for each patient.

Although the application model contains entities for representing several DICOM objects, such as exams results, medical reports, and study procedures, in this work only the patient, study, series, and image entities were considered because they are directly associated to data contained within the images. This consideration produces the simplified version of the DICOM application model shown in Figure 2, where the dashed region contains the entities of interest.

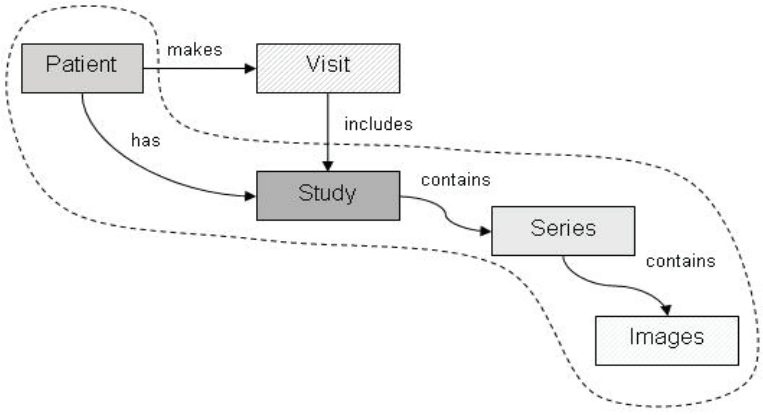
The DICOM information model derives from the application model and its entities are known as information object description (IOD). In an abstract way, an IOD describes real-world objects that share the same properties (NEMA, 2004b). This abstraction keeps a close relationship to the object-oriented design and programming paradigm.

Information model entities are featured by attributes whose types, multiplicities, and contents change depending upon the entity to which they belong. Attributes or data elements are defined in part 5 of the standard (NEMA, 2004c), and they are cataloged in part 6 of the data dictionary (NEMA, 2004d). An attribute is classified according to its presence in obligatory (types 1 and 2), conditional (types 1C and 2C), and optional (type 3).

Attributes are identified in the data dictionary through a tag composed from an ordered 16-bit number duple (*gggg,eeee*) expressed in hexadecimal form. These numbers represent the group and the element number within that group. The standard attributes have an even group number, different from 0000, 0002, 0004, and 0006, whereas private attributes, not contained in the data dictionary, have an odd group number, different from 0001, 0003, 0005, 0007, and FFFF. All aforementioned group numbers are reserved by the standard.

In addition to the tag, there are other data fields that also belong to the attributes structure, such as value representation (VR), value multiplicity (VM), length, and contained value. The VR describes, through a 2-byte character string,

Figure 2. Simplified DICOM application model showing information entities and relationships among them



the type and format of the data contained in the attribute value field, such as integer or floating numbers, dates, string characters, and sequences. The VM specifies the cardinality or number of values that are codified in the value field. The length contains the attribute's value size in bytes. Finally, the value field stores the attribute data, according to its respective presence type.

For each IOD there are defined operation sets and named services that are executed on the information objects. When an entity needs to perform an operation over an IOD, it must request the proper service to another entity, which behaves as a server. Each object-defined service establishes a service-object pair (SOP), and the whole service set that is applicable to a particular IOD is a named SOP class.

The *media storage service class* and the *queryrRetrieve service class* are examples of SOP classes. The first one comprises the M-READ, M-WRITE, and M-DELETE services set, applied to the reading, writing, and deleting of files with image IODs coming from acquisition modalities. The second class groups the C-FIND, C-MOVE, and C-GET services that can be requested for querying and transferring information from IODs associated with different entities.

An SOP instance is the occurrence of an IOD that is operated within a communication context, through a specific service set. For example, DICOMDIR directory files are SOP products from requesting the information writing service from a directory IOD to a physical storage media.

The simplified model from our research considers only four entities and their corresponding IODs. The patient entity contains data of the patient for whom radiological studies were performed as described by the study entity. The series entity models information resulting from radiological studies, such as images or signals, and keeps some kind of spatial or temporal relationship among them. The image entity represents the images coming from some of the existing acquisition modalities, for example, computed tomography (CT), magnetic image resonance (MRI), or ultra sound (US).

DICOM File Format

The DICOM file format, described in part 10 of the standard (NEMA, 2004e), defines the way data representing a SOP instance is stored in a physical storage media. The data is encapsulated as a stream of bytes, preceded by a header with metainformation required for identifying the SOP instance and class.

The header has an organized sequence of components, named file ID, which organizes files hierarchically. An ID has up to eight components, where each one is a string of one to eight characters separated by backslashes. The file ID generally corresponds to a directory path and to a filename, for example, SUBDIR1\SUBDIR2\SUBDIR3\ABCDEFGH.

Located after the header is the data set associated with the information model entity that is stored in the file. Depending upon the entity nature, this stream of bytes could represent some of the following objects: images, curves, signals, overlay annotations, lookup tables for transforming images pixel values according to acquisition modalities or values of interest, presentation images descriptions, structure reports, or raw data.

Within the context and scope of our research, we considered only DICOM files containing images associated with studies performed on patients. Therefore, the stored data describes the image plane and the pixels features, as well as values for mapping the image to color or gray scales, overlay planes, and other specific features.

DICOM files are gathered in collections sharing a common name space, such as storage volumes or directory trees, and having unique file identifiers within it. The file collection is an abstraction of a container where files can be created or read. Each collection must be accompanied by an index file with a DICOMDIR identifier, corresponding to a

DICOM directory object instance. Part 12 of the standard (NEMA, 2004g) describes the way the DICOM file information is encoded inside the physical storage media. It depends upon the file system used by the computer system for the files creation and interchange, as well as the physical media used for it.

DICOMDIR File Format

The DICOM standard defines a special object class as a named basic directory object, whose purpose is to serve as an organizing index for DICOM files stored in a physical media. The instance of a DICOM directory class object is a file with a unique filename and ID named DICOMDIR. The formal definition of the DICOMDIR object and its content are in part 3 annex F of the DICOM standard (NEMA, 2004b), whereas its structure complies with the DICOM files format specified in section 7 of part 10 (NEMA, 2004e).

There are registers in the DICOMDIR file with the information associated to objects stored in a DICOM files set, and it does not make reference to files that do not belong to the set. Each register contains a field identifying the represented information type, such as patient, study, series, and image, besides a group of specific fields with attributes extracted from the stored SOP instance. The registers are hierarchically sorted, and they are linked among themselves in the same hierarchy level and to the next lower level in the hierarchy (Figure 3).

The preceding information is generally present in the file, such as indicated in the standard, but some data fields can be optional, such as those related to the complementary information of the patient (birth date, sex, and age) and the studies (referring physician, institution, protocols, and diagnoses), as well as image descriptive information, for example, dimensions, samples by pixel, photometric interpretation, and gray-level window.

There should be a unique DICOMDIR file for each DICOM files set contained in a storage media. The DICOMDIR file location is related to the storage media directory organization, and it is commonly found at the root directory. DICOMDIR files help to make fast queries and searches throughout media contained images, without the need for reading whole file sets. Otherwise, searching and browsing images and information

Figure 3. Structure of a DICOMDIR file and its representation in a physical storage media

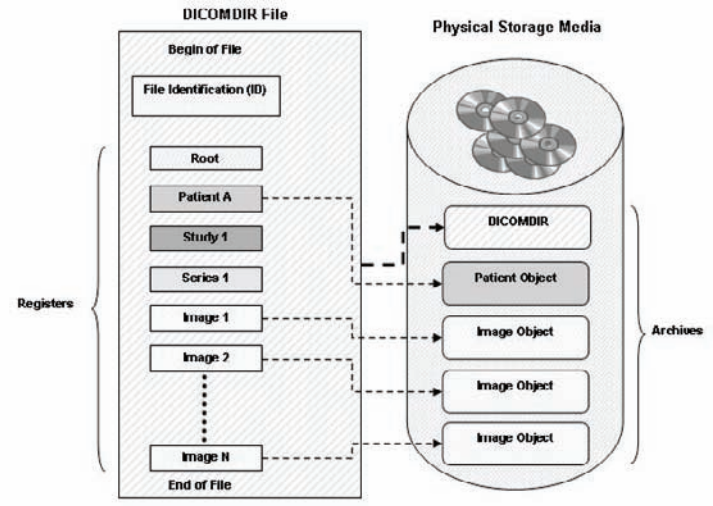
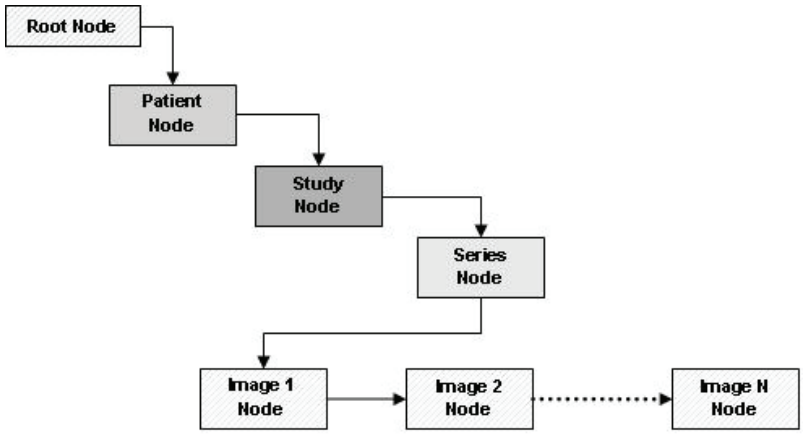


Figure 4. Structure of the data tree structure built by the reading tool



within file sets becomes an intensive, tedious, and difficult task.

DICOMDIR FILE READER IMPLEMENTATION

Description

The implemented tool enables DICOMDIR files to be opened in order to obtain the most relevant information from the DICOM-file collection they index. This information is organized in a hierarchical data structure, which can be easily consulted, and when coupled to a suitable graphic interface, permits the interactive browsing of the information related to the collection. The implementation was made using C++ language, based upon an object-oriented design that allows new DICOM data fields to be added to the reader. Some DCMTK library classes and methods (DCMTK, 2005) were used to facilitate the searching and decoding of the data contained within DICOMDIR files, thus avoiding the inherent complexity in the handling of the information stored under DICOM standard specifications.

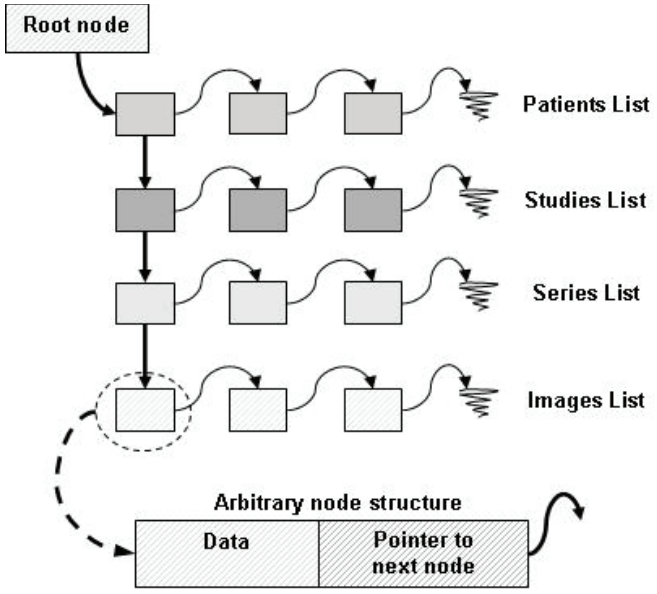
Data Structures

The references to SOP instances contained in DICOMDIR files are linked according to the entities' hierarchy of PATIENT-STUDY-SERIES-IMAGE, which is implicitly established in the DICOM information model, thus establishing a natural correspondence between the hierarchy and a tree data structure. This tree has heterogeneous content nodes that correspond to the DICOM simplified application model (Figure 2), so there are five node types: root, patient, study, series, and image. The entities' relationship cardinality sets the offspring multiplicity, that is, a patient could be object from several studies, whereas each one of them could have several image series.

During the reading of a directory file, all of the file's hierarchical links are traveled, and the related collection file information is gathered for filling the nodes' specific fields, thus building the data tree structure. At the end of the reading process, the tree has the relevant collection information, having a structure similar to that shown in Figure 4. This data structure could be browsed in order to consult the information without the need of accessing the whole file set again.

An approach was considered for the tree structure implementation where the nodes behave

Figure 5. Data tree implementation using linked node lists as data containers



simultaneously as structural elements and data containers. In this way, the nodes establish the tree hierarchical structure and each one of them also stores the information associated to the corresponding entity. Hierarchy levels are made up of linked node lists, and each one of these lists have a children's list, corresponding to elements from the next lower hierarchy level (Figure 5). This approach facilitates the travelling of the tree and its coupling to browsing interfaces.

A class hierarchy was defined for enabling the nodes polymorphic handling through virtual methods. In this hierarchy, the base class, DICOMDIRNode, defines the basic structure for the other nodes, as well as the virtual methods for accessing and travelling the tree. All subclasses deriving from it, that is, DICOMDIRRootNode, DICOMDIRPatientNode, DICOMDIRStudyNode, DICOMDIRSeriesNode, and DICOMDIRImageNode, redefine their contents and behavior according to the corresponding DICOM application model entity to which they belong.

Collected DICOM Data Fields

DICOM files contain rather large data-element groups that are cataloged in the DICOM data dictionary (NEMA, 2004d). However, not all of these elements are useful in the scope of software applications that handle medical images, and for that reason, a selection of relevant elements was made in order to limit the number of data fields gathered during the DICOMDIR files reading process. Table 1 shows the collected data fields with their tags, value representations, multiplicities, presence attributes, and corresponding DCMTK classes and C++ standard types.

One selection criteria for the data fields is based upon the information provided by the image processing. Among the selected fields are the image dimensions, the samples per pixel number, and the gray-level window. Another criterion is the general information that guides the user during the search and selection of images, for example, patient's data, description, and modality of the study. Complementary fields that do not provide

Table 1. DICOM attributes compiled during DICOMDIR file reading process. Attributes are grouped by node type, showing main DICOM data fields and corresponding DCMTK/ C++ data types for each one of them

Node type	Tag	Field name	VR	VM	Presence	DCMTK data type	C++ data type
PATIENT	(0010,0010)	Patient name	PN	1	Obligatory	DcmPersonName	char[ ]
	(0010,0020)	Patient ID	LO	1	Obligatory	DcmLongString	char[ ]
	(0010,0030)	Birth date	DA	1	Optional	DcmDate	char[ ]
	(0010,0040)	Sex	CS	1	Optional	DcmCodeString	char[ ]
	(0010,1010)	Age	AS	1	Optional	DcmAgeString	char[ ]
STUDY	(0020,0010)	Study ID	SH	1	Obligatory	DcmShortString	char[ ]
	(0008,0020)	Study date	DA	1	Obligatory	DcmDate	char[ ]
	(0008,0030)	Study time	TM	1	Obligatory	DcmTime	char[ ]
	(0008,1030)	Description	LO	1	Obligatory	DcmLongString	char[ ]
	(0018,1030)	Protocol name	LO	1	Optional	DcmLongString	char[ ]
	(0008,0090)	Referring physician	PN	1	Optional	DcmPersonName	char[ ]
	(0008,0080)	Institution	LO	1	Optional	DcmLongString	char[ ]
	(0008,1080)	Diagnoses	LO	1..N	Optional	DcmLongString	char[ ]
SERIE	(0020,0011)	Serie number	IS	1	Obligatory	DcmIntegerString	signed long
	(0008,0060)	Modality	CS	1	Obligatory	DcmCodeString	char[ ]
	(0018,0015)	Body part	CS	1	Optional	DcmCodeString	char[ ]
	(0008,0021)	Serie date	DA	1	Optional	DcmDate	char[ ]
	(0008,0031)	Serie time	TM	1	Optional	DcmTime	char[ ]
	(0008,103E)	Description	LO	1	Optional	DcmLongString	char[ ]
	(0008,1050)	Performing physician	PN	1..N	Optional	DcmPersonName	char[ ]

continued on following page

Table 1. continued

Node type	Tag	Field name	VR	VM	Presence	DCMTK data type	C++ data type
IMAGE	(0020,0013)	Image number	IS	1	Obligatory	DcmIntegerString	signed long
	(0008,0008)	Image type	CS	1..N	Optional	DcmCodeString	char[ ]
	(0008,0023)	Image date	DA	1	Optional	DcmDate	char[ ]
	(0008,0033)	Image time	TM	1	Optional	DcmTime	char[ ]
	(0028,0010)	Rows number	US	1	Optional	Uint16	unsigned short
	(0028,0011)	Columns number	US	1	Optional	Uint16	unsigned short
	(0028,0100)	Bits allocated	US	1	Optional	Uint16	unsigned short
	(0028,0101)	Bits stored	US	1	Optional	Uint16	unsigned short
	(0028,0102)	High bit	US	1	Optional	Uint16	unsigned short
	(0028,0002)	Samples per píxel	US	1	Optional	Uint16	unsigned short
	(0028,0103)	Pixel representation	US	1	Optional	Uint16	unsigned short
	(0028,0004)	Photometric interpretation	CS	1	Optional	DcmCodeString	char[ ]
	(0018,0050)	Slice thickness	DS	1	Optional	DcmDecimalString	float
	(0028,0030)	Pixel spacing	DS	2	Optional	DcmDecimalString	float
	(0028,1050)	Window center	DS	1..N	Optional	DcmDecimalString	signed long
	(0028,1051)	Window width	DS	1..N	Optional	DcmDecimalString	unsigned long
	(0028,1053)	Rescale slope	DS	1	Optional	DcmDecimalString	float
	(0028,1052)	Rescale Intersection	DS	1	Optional	DcmDecimalString	float
	(0004,1500)	Referenced file ID	CS	1..8	Optional	DcmCodeString	char[ ]

Algorithm 1.

```
Open DICOMDIR file for reading
Assign DICOMDIR file root register to NODE variable
With NODE, do recursively /* Begin of recursive block */
{
  Assign NODE first child to Next_Register variable
  While (Next_Register <> NULL)
  {
        Select according to Next_Register type
        {
        Case PATIENT:
                    Get patient node related information
                    Create a new DICOMDIR_PatientNode node object
                    Fill node members with gathered information
                    Insert node in the data tree
                    Assign Next_Register to NODE
                    Call recursive block with new NODE value
Complete missing patient information extracted from related image file
        Case STUDY:
                    Get study node related information
                    Create a new DICOMDIR_StudyNode node object
                    Fill node members with gathered information
                    Insert node in the data tree
                    Assign Next_Register to NODE
                    Call recursive block with new NODE value
Complete missing study information extracted from related image file
        Case SERIES:
                    Get series node related information
                    Create a new DICOMDIR_SerieNode node object
                    Fill node members with gathered information
```

significant information for the image processing or to the user applications were not used for the gathering process, for example, address, occupation, and medical antecedents of the patient, and the technical information from the image acquisition devices.

Reading Process

The reader uses Algorithm 1 for the DICOMDIR information retrieving process.

It can be observed that in this algorithm starting from the parent nodes, each tree node and its children's nodes are recursively travelled, level by level, until reaching the deepest tree level. Only nodes from PATIENT, STUDY, SERIES, and IMAGE types are taken into account, though it is possible to extend the consideration to other types of nodes. For each visited node, the selected fields are compiled according to the node type (see Table 1). However, as explained in the DICOMDIR File Format section, not all the fields have an obligatory presence inside the DICOMDIR file, therefore the missing information must be completed from the corresponding DICOM image file, once its file identifier is known.

Some methods from the DCMTK library are used to travel through the DICOMDIR structure and retrieve its data. These methods can handle data elements encoded either in *big Endian* or *little Endian* byte ordering, or with any kind of representation values and cardinalities. The retrieving methods are invoked by the data element tag each time a node is visited, whereas the travelling methods are used during recursive calls to the tool's main reading method.

RESULTS AND DISSCUSSION

The DICOMDIR reader was integrated for its test and validation into software for neurosurgery and brachytherapy planning, developed with the VVM library (Montilla, Bosnjak, & Villegas, 2003). A graphical interface was coupled to the reader in order to enable the navigation of the information in a simple way. By just opening the associated DICOMDIR file, users will be able to have the interface display the patients' study images and their related medical information. Control widgets provided by the interface were used for interactively browsing and selecting the images.

The reading process was verified with DICOMDIR files associated to several image sets, coming from studies of CT and MRI performed on different patients. It could be proved that the reader recollected the information specified by the class definitions for each type of node, enabling display of its structure through the interface. Two examples from the tests are shown in Figure 6. In both examples, the studies were made on the patient's head but with two different modalities. The CT study has an eight-image series, whereas the MRI study contains 14 images.

It was observed that access to the information collected from the studies can be made in an organized and fast way, making transparent the navigation process of data structures and improving the efficiency in the use of the software. The tool is able to be integrated with other applications under development that require the handling of DICOM images and DICOMDIR files.As is usual in software projects implying code reusability, benefits from using our tool are going to be directly reflected in ease and speed of development.

CONCLUSION

The design used for the DICOMDIR file reader allows the medical application programmer to effortlessly incorporate the feature for the handling of this kind of file in software under development. Also, it avoids exhaustively understanding the DICOM standard and DCMTK classes and methods, which can be really hard and bothersome, enabling the programmer to focus on the integration of the

tool into the application, as well as on the development of a navigation interface, according to the application's needs. Because the reading tool has open-source code, it can be reused and modified at will by the programmers. In addition, the design used for the tool development enables inclusion of new data fields to the presently used entities, as well as to add other information model entities to the data tree structure.

By integrating the tool into a medical application that handles DICOM image sets, it was proved it facilitates the browsing and searching of the information contained in the image sets, accelerating the fulfillment of these tasks and improving the efficiency and performance of the application. Open-source programming tools of this type also facilitate the development of medical applications and help to reduce software costs, thus making it more accessible to health institutions, physicians, and patients, particularly in regions where investment in health care solutions is either limited or not a priority issue.

FUTURE WORKS

We are considering the future implementation of tools for converting images from other formats to the DICOM format and for the creation of DICOMDIR files from nonindexed studies files. A tool for anonymization of DICOM fields also could be useful in protecting patient confidentiality during the sharing of clinical data among research teams. We hope that the implementation and subsequent use of this tool set will represent an incremental increase in the efficiency, quality, and speed of development of medical informatics software, producing applications that will be more complete and flexible at same time.

ACKNOWLEDGMENT

The authors want to express their gratitude to Dr. Luis Iván Jara, a neurosurgeon appointed to the Hospital Metropolitano del Norte (Valencia-Venezuela), for providing us the DICOM image sets used for the functional verification and evaluation of the reading tool. Financial support for this research was granted by Scientific and Humanistic Development Council of the University of Carabobo, Venezuela, under project CDCH-UC No. 0461-06.

REFERENCES

Digital imaging and communications in medicine took kit (DCMTK). (2005). *DICOM toolkit software documentation*. Oldenburger Forschungs und Entwicklungsinstitut für Informatik-Werkzeuge und Systeme (OFFIS). Retrieved on September 29, 2005, from http://dicom.offis.de/dcmtk.php.en

DicomLib. (2005). *DICOM software documentation.* Sunnybrook and Women's College Health Sciences Center Imaging Research Group. Retrieved on September 29, 2005, from http://dicomlib.swri.ca/dicomlib/html/index.htm

GDCM. (2005). *Grass roots DICOM software documentation.* Centre de Recherche et d'Applications en Traitement de l'Image et du Signal (CREATIS). Retrieved on September 29, 2005, from http://www.creatis.insa-lyon.fr/Public/Gdcm

Montilla, G., Bosnjak, A., Jara, L. I., & Villegas, H. (2005). Computer assisted planning using dependent texture mapping and multiple rendering projections in medical applications. *Proceedings from 3rd European Medical & Biological Engineering Conference and Ifmbe European Conference on Biomedical Engineering. Ifmbe* (pp. 4420-4425). Praga, Czech Republic: .

Montilla, G., Bosnjak, A., & Villegas, H. (2003). Visualización de mundos virtuales en la medicina. bioingeniería en Iberoamérica: Avances y desarrollos. In C. Müller-Karger & M. Cerrolaza (Eds.), *Centro internacional de métodos numéricos en ingeniería CIMNE* (pp. 519-545). Barcelona: .

National Electrical Manufacturers Association (NEMA). (2004a). *Digital imaging and communications in medicine. Part 1: Introduction and overview* (NEMA Standards Pub., PS3.1).

National Electrical Manufacturers Association (NEMA). (2004b). *Digital imaging and communications in medicine. Part 3: Information object definitions* (NEMA Standards Pub., PS3.3).

National Electrical Manufacturers Association (NEMA). (2004c). *Digital imaging and communications in medicine. Part 5: Data structures and encoding* (NEMA Standards Pub., PS3.5).

National Electrical Manufacturers Association (NEMA). (2004d). *Digital imaging and communications in medicine. Part 6: Data dictionary* (NEMA Standards Pub., PS3.6).

National Electrical Manufacturers Association (NEMA). (2004e). *Digital imaging and communications in medicine. Part 10: Media storage and file format for media interchange* (NEMA Standards Pub., PS3.10).

National Electrical Manufacturers Association (NEMA). (2004f). *Digital imaging and communications in medicine. Part 11: Media storage application profiles* (NEMA Standards Pub., PS3.11).

National Electrical Manufacturers Association (NEMA). (2004g). *Digital imaging and communications in medicine. Part 12: Media formats and physical media for media interchange* (NEMA Standards Pub., PS3.12).

Revet, B. (1997). *DICOM cookbook for implementations in modalities* (Tech. Rep.). Philips Medical Systems.

APPENDIX

Glossary of Terms

American College of Radiology(ACR)/ National Electrical Manufacturers Association (NEMA)(ACR): This joint committee is responsible for the development and maintenance of the DICOM standard.

Acquisition modalities: These imaging techniques and devices provide radiological images from patients' anatomy. Most commonly used modalities are CT (computed tomography), MRI (magnetic resonance imaging) and US (ultra sound).

Application programming interfaces (APIs): A set of specialized functions, class libraries, and tools used by software programmers to facilitate the application development process.

Attribute: A property of an information object represented as a data element.

Big Endian: An encoding scheme where multiple byte values are encoded with the most significant byte first, followed by the remaining bytes in decreasing order of significance.

Browsing interfaces: These mechanisms and widgets provide a way to navigate information within software applications.

Data element: This is a single atomic information unit that is related to a real-world object attribute and is defined by an entry in the DICOM data dictionary.

Digital imaging and communications in medicine (DICOM): This standard establishes norms for handling, storing, and interchanging medical images and associated digital information.

DICOM application model: An entity/relationship diagram used to model the relationships existing between real-world objects within the DICOM standard's scope.

DICOM data dictionary: This is a catalog of DICOM data elements that describes the semantics and contents of each one of them.

DICOM information model: An entity/relationship diagram used to model the relationships between the information-object definitions representing classes of real-world objects defined by the DICOM application model.

DICOMDIR file: This is a unique and mandatory file that accompanies a file set and indexes these files.

Entity/relationship diagram: This is a graphical representation of a set of objects and the relationships existing among them.

Image-based diagnosis: Techniques used by health care specialists to diagnose patients' diseases through analysis of radiological imaging studies.

APPENDIX CONTINUED

I**nformation object description (IOD):** This is an abstract definition for real-world objects that share the same properties.

Little Endian: An encoding scheme where multiple byte values are encoded with the least significant byte first, followed by the remaining bytes in increasing order of significance.

Open source: Programming paradigm based upon software engineering principles that pursues software code reutilization to facilitate the development of applications.

Proprietary code: Programming paradigm that establishes that the software applications code is protected and is not available either for modification or reutilization.

Software development kits (SDKs): See APIs.

Service: This is an operation that can be requested for acting over an information object.

Service-object pair (SOP): This is a relationship that is established between an information object and an operation or the service applicable over it.

SOP class: This is the whole set of services applicable over an information object.

Surgical planning: This describes the preoperative procedure where the surgeon determines surgical protocols and approachg trajectories to be followed during the intraoperative stage of the surgery.

Value multiplicity (VM): This is the DICOM data field that specifies the cardinality or number of values existing for a data element.

Value Representation (VR): This is the DICOM data field that specifies the data type and format of values existing for a data element.

This work was previously published in International Journal of Healthcare Information Systems and Informatics, Vol. 2, Issue 1, edited by J. Tan, pp. 54-70, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 3.6
Tools for the Study of the Usual Data Sources found in Libre Software Projects

Gregorio Robles
Universidad Rey Juan Carlos, Spain

Jesús M. González-Barahona
Universidad Rey Juan Carlos, Spain

Daniel Izquierdo-Cortazar
Universidad Rey Juan Carlos, Spain

Benjamin E. Erlandson
Universidad Rey Juan Carlos, Spain

ABSTRACT

Due to the open nature of Free/Libre/Open Source software projects, researchers have gained access to a rich set of development-related information. Although this information is publicly available on the Internet, obtaining and analyzing it in a convenient way is not an easy task and many considerations have to be taken into account. In this paper we present the most important data sources that can be found in libre software projects and that are studied by the research community: source code, source code management systems, mailing lists and bug tracking systems. We will give advice for the problems that can be found when retrieving and preparing the data sources for a posterior analysis, as well as provide information about the tools that support these tasks.

INTRODUCTION

The fact that communication and organization are heavily tied in libre software[1] projects to the use of telematic means and that these interactions are, in general, stored and publicly offered over the Internet makes the number of data sources where development information can be extracted from

grow beyond source code. In addition, the ability of having memory (as data from activities in the past can be obtained) offers the possibility of performing longitudinal analysis as well. Research groups from all around the world have already taken benefit from the availability of such a rich amount of data sources in the last years. Nonetheless, the access, retrieval and fact extraction is by no means a simple task and many considerations have to be considered to successfully mine the data sources.

This chapter is probably the first attempt to have a detailed description of the most common data sources that can generally be found for libre software projects on the Internet and the data that can be found in them. In addition, we present some available tools that might help researchers obtaining and partially analyzing the described data sources. These data sources comprise **source code**, **source code management** (in the following, SCM), **mailing lists archives**, and **bug tracking system** (in the following, BTS).

Mining and analyzing these data sources offer an ample amount of possibilities that surpass or complement other data-acquiring methodologies such as surveys, interviews or experiments. The amount of data that can be obtained, in a detailed way and in many cases for the whole lifetime of a software project, gives a precise description of the history of a project (Bauer and Pizka, 2003). In this sense, we have access to the activities (the what), the points in time (the when), the actors (the who) and sometimes even the reason (the why) (Hahsler and Koch, 2005). Compared to surveys, mining these data sources allows to access data for thousands of developers and a wide range of software projects. Most of these efforts can be considered as non-intrusive, as researchers can analyze the projects without interacting with developers. But even in a small environment, e.g., when evaluating the impact of software tools in a small team (Atkins et al., 2002), the use of data from one or more of these sources provides additional insight. Furthermore, mining software repositories has many advantages compared to conducting experiments as real-world software projects are taken into consideration (Mockus and Votta, 2000, Graves and Mockus, 1998).

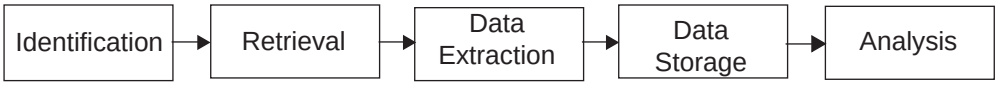
The structure of this chapter is as follows: the next section handles the identification of the data sources as well as its retrieval process. Next, various analysis on source code are introduced (hierarchy, file discrimination, analysis of *traditional* source code files, analysis of the rest of files (such as documentation, multimedia, etc.), and authorship). The fourth section presents how SCM systems can be mined, putting special attention on the CVSAnalY tool and some details to be considered when performing analyses on CVS. The fifth section presents the most common format in which mailing lists are stored (MBOX), while the sixth one is devoted to present the data to be found in a BTS. Finally, the reader can find a short summary of the chapter in the last section.

IDENTIFICATION OF DATA SOURCES AND RETRIEVAL

There are some steps before the analysis of data from libre software projects can be started that should be considered: identification and retrieval. It should be noted that there may be several ways of accessing the data, depending on the projects. This is because of the use of the several development-supporting tools that projects use and of having different usage conventions (for instance, the use of tags, comments, among others, may differ from one project to another). The complexity and feasibility of both activities depend on the data source and on the project. Figure 1 gives a diagram that shows the steps that have to be accomplished for any source considered in our study.

In general terms, the identification of the data source depends mostly on its significance for the software development of a project. Hence, identifying the source code, the SCM system, the mailing lists or the BTS is in no way problematic

Figure 1. Whole process: from identification of the data sources to analysis of the data



as it lies in the interest of the projects that feedback is provided by users in an easy and fast way. In these cases, the biggest drawback is the lack of historical data. Sometimes we only have a partial set of the data, and in the worst cases nothing at all. This situation is common for software releases, where finding historical versions of the software is sometimes not possible. Other situations where this might happen is when a development tool has not been used in the early stages of development. This is the case of many projects that start using a SCM system once the project has gained certain momentum. Having only partial data can also be the result of a migration from one tool to another, losing in the way some information if not all. When researching libre software projects, these considerations have to be taken into account.

But there exist other data sources for libre software projects that are not so obvious and hence their identification is not that straightforward. For instance, organizational information that is embedded into some format and that is beyond the use of standard tools as SCM systems, mailing lists and BTS. In general, such type of information is project-dependent and can be only obtained for one project or a small number of them. This is the case for packaging systems such as the .deb format used in Debian and Debian-based distributions or the .rpm Red Hat package system in use in Red Hat and other distributions. But beyond this, we can find project-related information in other places such as the Debian Popularity Contest (Robles et al., 2006b) or the Debian Developer database (Robles et al., 2005). Other data sources may also be considered; for instance, in KDE there is a file that is used to list all the ones who have write access to their SCM repository. Another example is given in a study by Tuomi (Tuomi, 2004) in which the credits file, a text file listing all important contributors to the project, of the Linux kernel are studied in detail. Identification of the data source requires in such cases specific knowledge on the project and is difficult if not impossible to be generalized.

Once the data source has been identified, it has to be retrieved to a local machine in order to be analyzed (see Figure 1). Although this process may not seem to be very difficult at first, previous experiences have shown that some considerations and good practices should be followed in this step as reported by Howison et al. in the retrieval of information from the web pages hosted at SourceForge (Howison and Crowston, 2004). For instance, the analysis of the credits file, which can be found together with the sources in many projects, has to deal with the complexity that there is no standardized way of naming the authors, so projects follow their own conventions.

In the next sections we will enter into detail in the process of data extraction and data storage once the data have been properly retrieved from the information source to a local machine.

SOURCE CODE

We should begin with the concept of release. It is important due to the fact that it points out the main milestone happened during the life of a project. It usually has a common nomenclature which is akin to "MM.mm.bb". Where "MM" means the number of the major release, "mm" means the number of minor releases and "bb" connotes some bug fixes and small improvements.

As software development projects, source code is the central point of all interactions, being

a primary way of communication and playing a major signaling and coordination role. According to (Lanzara and Morner, 2003), source code "is transient knowledge: it reflects what has been programmed and developed up to that point, resuming past development and knowledge and pointing to future experiments and future knowledge.".

The study of the source code, as the main product of the software development process, is a matter that has been done for over thirty years now. But not only *traditional* source code (i.e., programmed in a programming language) can be taken into account, but also all the other elements that make the software, such as documentation, translation, user interface and other files (Robles et al., 2006a).

The analysis usually starts with a source code base that is stored in a directory (or alternatively in a compressed directory, usually in tar.gz or tar.bz2 format common in the libre software world). After decompressing the tarball, if needed, the hierarchical structure of the source code tree is identified and stored.

Then, files can be grouped into several categories depending on type (as will be described below) which allows for a more specific analysis. This means, for instance that source code files in a programming language can be analyzed differently than images or documentation files. On the other hand, the discrimination for files with source code can be finer, identifying the programming language and offering the possibility of using alternative metrics depending on it. As a consequence, object oriented metrics could be applied to files containing Java code, but would not be required for files that are written in assembler language.

The whole process can be observed in Figure 2: after (possibly) decompressing, the directory and file hierarchy is obtained, then files are discriminated by their type and finally analyzed, if possible taking into consideration the file type that has been identified in the previous step. In the following subsections the different steps are described more in detail.

Hierarchical Structure

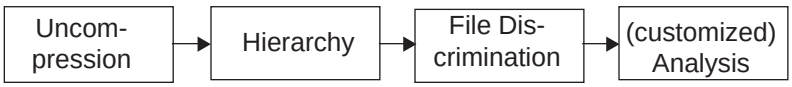
The structure of directories and files of a software program (and how it changes over time) has already been the focus of some research studies (Capiluppi, 2004, Capiluppi et al., 2004). The idea is that the technical architecture and probably therefore the organization of the development team is mapped by the tree hierarchy of directories. So, from a directory hierarchy, we could infer the organizational structure of a libre software development project.

File Discrimination

File discrimination is a technique that is used to specifically analyze files on behalf of their content (Robles et al., 2006a). The most common way of discriminating files is by using heuristics, which may vary in their accuracy as well as in the granularity of their results.

A first set of heuristics may determine the type of a file by considering its extension. File extensions are non-mandatory, but usually conventions exist so that the identification of the content of a file can be made easier and to enable the automation of administrative tasks.

Figure 2. Process of source code analysis



Hence, a first step for file discrimination consists of having a list of extensions that links to the content of the file. In this context, the *.pl* extension is indicative for a file that contains programming instructions while a *.png* can be considered as an image file. Of course, this can be done at several granularity levels, meaning that a *.c* file is a file that with high probability contains programming language, being that the programming language C code. Table 1 shows an excerpt of the list of file extensions that can be used.

The file types that can be considered are documentation, images, internationalization (i18n) and localization (l10n), user interface (ui), multimedia and code files. For the latter type, a more detailed analysis and discrimination between source code that is part of the software application (code) from the one that helps in the building process (generally Makefiles, configure.in, among others) and from documentation files that are tightly bound to the development and building process (such as README, TODO or HACKING) can be made.

A second step in the process of file discrimination includes inspection of the content of the files both to check if the identification made by means of matching file extensions is correct and to identify files that have no extension or whose extension is not included in the previous list.

Table 1. (Incomplete) set of matches performed to identify the different file types

File type	Extension/file name matching
documentation	*.html *.txt *.ps *.tex *.sgml
images	*.png *.jpg *.jpeg *.bmp *.gif
i18n	*.po *.pot *.mo *.charset
ui	*.desktop *.ui *.xpm *.theme
multimedia	*.mp3 *.ogg *.wav. *.au *.mid
code	*.c *.h *.cc *.pl *.java *.s *.ada
build	configure.* makefile.* *.make
devel-doc	readme* changelog* todo* hacking*

In this case, heuristics are generally content-specific and may go more in depth depending on the detail of discrimination we are looking for. One of the most common ways to improve file discrimination by looking at the file content is to analyze the first line. There exists some convention in source code files that denotes that the programming language that they contain. For instance, in the case of a file written in the Python, *Bourne again* shell or Perl programming language (examples can be found in Table 2), the first line could contain respectively the following information[2].

In the case of programming languages, further information can be gained from the structure of the code, by the identification of specific keywords or other elements such as specific comments. For text files (especially the ones that are based on mark-up languages), tags and other specific elements may help in the identification process. Finally, other algorithms can be taken into account, as the information returned by the UNIX *file* command on the file type (which also identifies some of the binary formats, especially useful in the case of images).

Some of the previous discrimination techniques are already in use in some tools, most notably in SLOCCount (see (Wheeler, 2001, Robles et al., 2006b)). As SLOCCount counts the number of lines of code it is only concerned with identifying source code files and identifying the programming language in which they are written, not considering all other file types that we have taken into consideration in this work (documentation, translations, and other).

Analysis of Source Code Files

The analysis of source code files is one of the most known tasks. There exist an ample number of measures that can be and have been extracted directly from the source code, among other its length (in lines of code or source lines of code), complexity measures (as the popular

Table 2. Examples of first line indicating that the file is written in Python, Shell or Perl respectively

```
#! /usr/bin/python
#! /usr/bin/sh
#! /usr/bin/perl
```

ones proposed by Halstead (Halstead, 1977) and McCabe (McCabe, 1976)) or even composite metrics such as the Maintainability Index (Oman and Hagemeister, 1992).

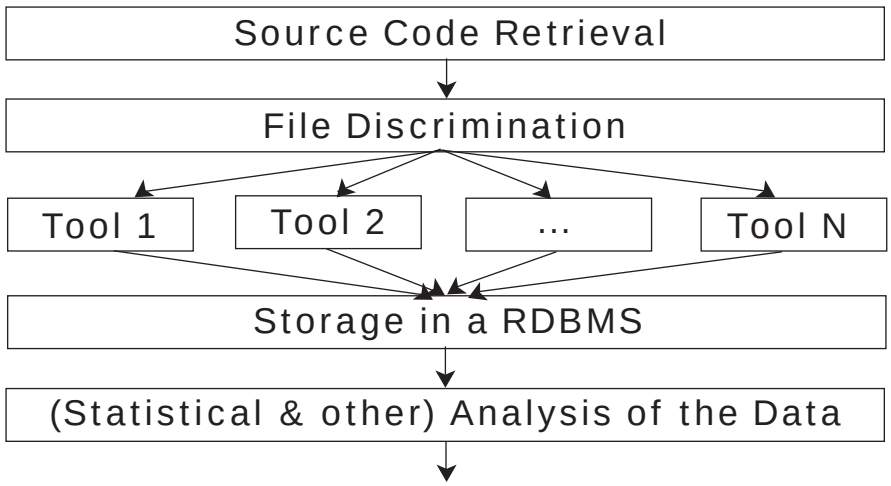
The availability of a certain range of tools for this purpose makes the conception of a tool that integrates all of them a primary task. The goals of the integration is to make it possible to extract all the metrics and facts from source code files by using several tools in a simple and most uniform way. The tools used to measure the code should be, if possible, used as *black boxes*, so that the integration tool does not need to know or adapt its inner functioning. In addition, the integration tool should handle the input to and the output from the measurement tools to ease its use.

That is precisely what can be done with GlueTheos[3], a tool designed and implemented by the authors of this chapter: a system with an architecture that allows the data retrieval and analysis of public software development data repositories. The structure of the GlueTheos tool is presented in Figure 3, and consists of a module for downloading (if required, with a periodical pattern) the sources to be analyzed, to examine the content of the sources on a file basis, to run the tools depending on the file type, to identify the results and store them properly in a relational database system and finally to provide results.

The current version can access *CVS*, *Subversion* and *git*. File discrimination allows to run the tools specifically on the files where this makes sense. Hence, if we had a tool that returns object oriented measures from Java files it would make no sense to run it on a shell script. This step then optimizes the analysis to be performed.

The next step is the heart of GlueTheos and consists of running the different tools on the source code and retrieving the data that these tools return. GlueTheos has been designed in a way in which it does not require to adapt the tools it integrates, hence facing the complexity of the various ways of calling them and the various ways of obtaining their results. Both calling and returning have been solved following an object-oriented approach, so that for any tool only the differences have to be implemented.

Figure 3. Architecture of the GlueTheos tool



The calling procedure requires information such as the way a tool has to be called (mainly the path to the executable), the input that the tool requires (usually a file or a directory) and the type of output that the tool returns (again, usually a file or a directory).

The returning methods depend on the type of output that the analysis tools provide. If it is a file, the number of returning elements has to be given and the special character that is used to separate them (usually a tabulator, a white space or a comma). In general, the path that gives the filename of the file that has been analyzed is also returned, so its position has to be specified.

After retrieving and storing the data from external tools, GlueTheos has to consider only the data in the database to obtain statistical and other results from the data set. This includes some procedures to enhance the database structure in order to normalize the fields or to obtain intermediate tables with statistical information that is of common use.

Analysis of Other Files

Besides source code written in a programming language, we identify other artifacts that compose the sources of libre software projects. (Robles et al., 2006a) shows the many possibilities that arise from the study of those files, but other references to this issue may be found in related literature. Some authors have focused on the analysis of the change log files (Capiluppi et al., 2003) as they usually follow a common pattern in libre software projects, although sometimes this pattern is slightly different from the standardized way used in GNU projects[4].

Translation files may be used to keep track of the amount of translation work that has been accomplished to the moment and hence have a quantitative manner of knowing the support of that software in a given language.

Regarding licenses, in addition of a reference to the licensing terms that can be found at the top of the code files, usually projects have a text file which includes the full text of the license. The filename may give enough evidence for the type of license that a project uses, but other ways can also be considered. One that we have been trying with is the use of a locality-sensitive hash like nilsimsa (Chang and Mockus, 2008). This type of hashes return codes with small changes for inputs that differ only slightly. As intellectual property issues have become a recent area of interest among industry, some approaches (and tools, such as FOSSology) have been presented that target these problems (Gobeille, 2008).

Finally, the amount of documentation for a software system could be a good topic for empirical research. In this sense, the *doceval*[5] tool offers a way of assessing and partially evaluating the documentation that can be found in the sources of libre software projects (Robles et al., 2006c).

Authorship Analysis

Usually, source code files contain copyright and license information in their first lines (Spinellis, 2003). So, for instance, the notice in the apps/units.c file of the GIMP project shown in Table 3 clearly states that the copyright holders are Spencer Kimball and Peter Mattis and that the license in use is the GNU General Public License.

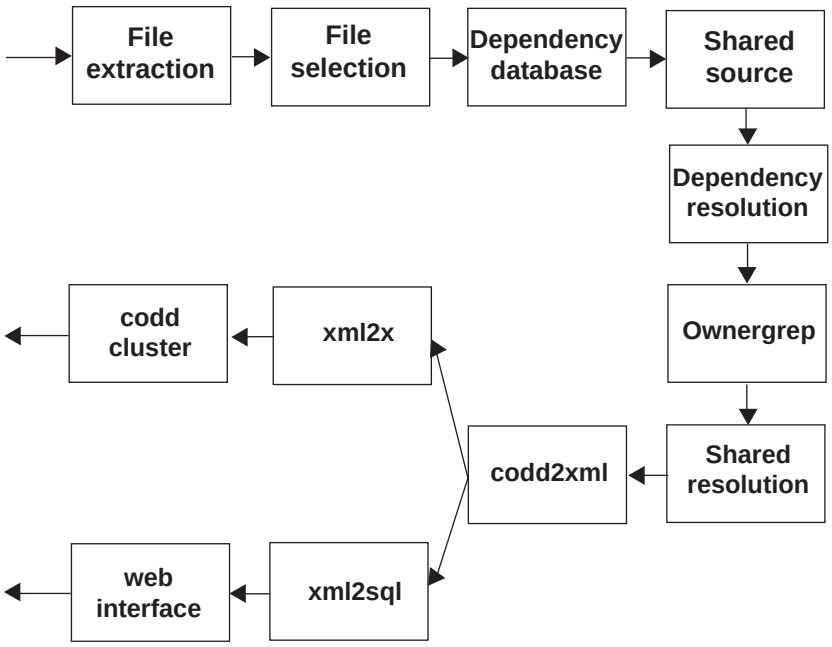
CODD[6] is a tool that searches for authorship information in source code by tracking copyright notices and other information in the headings of files (Ghosh and Prakash, 2000, Ghosh et al., 2002). It assigns the length (in bytes) of each file to the corresponding authors. The process that CODD follows to obtain these results are shown in Figure 4.

File extraction is composed of the *init* subroutine which takes the source code package (or packages) that are given through the command line by the user, decompresses them if necessary, and tries to identify recursively the files that the package contains.

Table 3. Excerpt of a copyright statement found in the GIMP project

```
/* The GIMP – an image manipulation program
 * Copyright (C) 1995 Spencer Kimball and Peter Mattis
 *
 * This program is free software; you can redistribute it and/or modify
[...]
```

Figure 4. Process of the CODD tool



During the file selection all source code files, documentation, interfaces and not-resolved implementations are taken together with their size in bytes, their MD5 sum and their relative route in the package and stored in a *codd*[7]. Files are selected by means of their extension, so for instance the *.c* file extension is categorized into source code files (usually they correspond to C files). CODD stores the *.h* files that have a *.c* in the same package as interfaces (the algorithm that is used here depends partially on the programming language that is being analyzed). Calls to an interface in source code files (for instance *.c* files for C) that do not have their corresponding interface in the same package (a *.h* for C) will be classified in the non-resolved implementations category, that in a future step will be handled for dependency resolution.

In a third step two databases are created in order to find shared source code and dependencies. In the first one, named *codefile_signatures*, all the MD5 sums of the files are stored. The second

one contains all the interfaces that were found in the previous step. MD5 is a type of hash that allows to know if two files are equal; if they are they will have the same MD5 hash value. MD5 hashes are very interesting when the source code file is exactly the same, but a single modification (i.e., when it is committed into the SCM of the new project the RCT-type id changes) makes it impossible to recognize it as a shared file.

In order to find shared source code, CODD runs another time through all codds and looks if the source code files appear more than once in the database (really it looks if the name and MD5 sum appear more than once). If this occurs, the file is located in at least two different packages. A similar process is used to resolve dependencies. CODD will search for not-resolved implementations in the codds and compare their MD5 sums with the ones that are stored in the interfaces database. A list with all the packages where this interface is implemented will be inserted as well.

The owner grep block is the one that is responsible for looking for authorship contributions. It runs again through all source code and documentation files and scans authorship attribution by means of certain heuristics. Mainly the heuristics look for several patterns: email addresses [a], copyright notices [b] and software control versioning ids [c]. Information about the authors is stored in the *credits* section of the codds. The regular expressions that have been used are following:

[a] Email grep: [\d\w_\=\.\%]+?\@[\d\w\._\-]+?\.\w+?)(?=[\s:>\n\r\)]|$

[b] Copyright grep: .*copyright (?:\(c\))?[\d\,\-\s\:]+(?:by\s+)?([?\d]*)

[c] Id grep: (?:Id|Header).*?\d\d\:\d\d\:\d\d (\S+?) \S+?

Next, the resolution of shared source code is done. In the *shared* source code section of the codds we still have files and a list of packages that contain these files. As these files can only be assigned to a single package (in order to avoid double counting the contribution of an author), CODD looks for its author (running again the *ownergrep* algorithm) in the file and assigns it to the package in which the author is the main contributor.

The last blocks of Figure 4 show that the codds can be then transformed to an intermediate and independent format (as for instance XML and SQL).

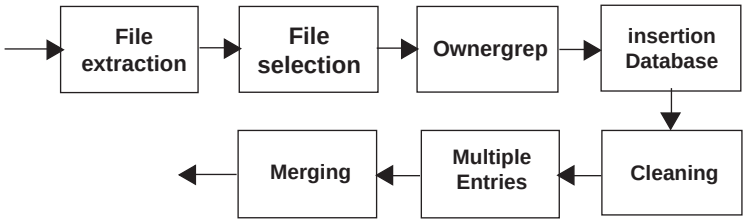
CODD is a very powerful tool, but it has some weaknesses. The most important one is that it lacks a way of merging the various ways in which an author may appear. So, authors may appear several times with different names or e-mail addresses. For instance, we have found that some developers have up to 15 e-mail addresses. In the case of companies, the same may happen; so, IBM or the MIT appear in several ways (up to ten times!) with slightly different wordings (Robles et al., 2007).

Cleaning of the data should also be enhanced. The heuristics that are used in CODD have proved to be very powerful, but cannot avoid that developers use different conventions to assign copyright. Most of these problems could be solved by a set of more powerful heuristics.

As *CODD* raises some limitations regarding authorship identification, the authors decided 2004 to create a new tool from scratch based on the heuristics given by CODD. This tool has been called *pyTernity*[8]. The architecture of pyTernity is identical to the one described for CODD as it can be seen from Figure 5, although it lacks of all the procedures for identifying dependencies among files.

The most innovative elements are the ones that consider data cleaning and the identification of multiple entries. For the former, entries in database are removed from elements that make them different; this goes from additional white-spaces to the avoidance of dots. Some heuristics have been set up for this, although they have been complemented with a database of frequent changes. Cleaning includes splitting up an entry

Figure 5. Process of the pyTernity tool



when it is due to two or more authors. So, the entry "Spencer Kimball and Peter Mattis" will result in two, one for Spencer Kimball and another one for Peter Mattis. If this is the case, both names appear as authors of the file and get attributed half of its length (in bytes or lines of code).

The latter part comprises the identification of multiple entries. Developers may appear in several ways, making results very unsatisfactory. The first efforts in this sense went into the construction of a large database where the various entries identified for a given developer were merged into a unique one. This has proved to enhance results in a prominent way. Other, more complex, routines may be used for extracting names from e-mails, with procedures from the machine learning world as for instance applying *named entity recognition* (Minkov et al., 2005).

Once cleaning has been performed and multiple entries have been identified, pyTernity merges the entries in the database so that authors appear only once in a file. This procedure implies to add all the contributions by an author, so it adds the lengths of each entry (in bytes or lines of code).

SCM SYSTEM META-DATA

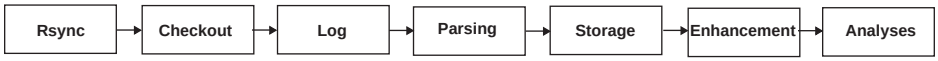
Generally speaking, most libre software projects use a SCM system to manage file versions during the development process. They allow to track changes and past states of a software project. Thus, obtaining the current and any past state of the code is made possible by the use of a SCM system. This allows to make source code analyses as we have presented them in the previous section in a longitudinal manner and to extract facts on the evolution of a software project.

But beyond this, SCM systems store a set of meta-data of the changes. These meta-data can be tracked and analyzed. This information is usually related to the interactions that occur among developers and the SCM systems. In general the information is only related to actions that comprehend write access while reading (downloading the sources) or obtaining other information (diffs, among others) cannot be tracked in that way. For instance, along with a change, valuable information is recorded, like the date of change, the full path where the change occurred, user who committed or the comment written by the committer[9].

Here, we present a tool that analyzes the interactions that occur between developers and the most used SCM systems used in libre software projects at the current time, CVS, Subversion, *git* and *Bazaar*. This tool, which has been labeled CVSAnalY, is based on the analysis of the SCM system log entries and implements all the theoretical details that will be presented in this section (Robles et al., 2004). Another tool, called SoftChange, has been used for similar purposes by German *et al.* (Germán and Hindle, 2005).

In CVSAnalY any interaction -also called commit- a committer does with the central SCM system repository is logged with following data

Figure 6. Process of the CVSAnalY tool



associated (some aforementioned): committer name, date, file, revision number, lines added, lines removed and an explanatory comment introduced by the committer. There is some file-specific information that can also be extracted, as for instance if the file has been removed[10]. On the other hand, the human-inserted comment can also be parsed in order to see if the commit corresponds to an external contribution or even to an automated script.

Basically CVSAnalY consists of three main steps, preprocessing, insertion into database and post-processing, but they can be subdivided into several more as it has been done in Figure 6. In the following subsections the inner functioning of CVSAnalY will be presented, focusing on details of its use with CVS. Its use with other SCM systems should be similar.

Preprocessing: Retrieval and Parsing

Preprocessing includes downloading the sources from the repository of the project in study. Once this is done, the logs are retrieved and parsed to transform the information contained in log format into a more structured format (SQL for databases or XML for data exchange).

Besides the information for every commit, other data obtained from the parsing requires some attention. Although committers seldom change their username, sometimes this happens, so the various usernames have to be merged into a unique one. For instance, in the KDE project committers usually get an account prior to a *kde.org* e-mail address. If a developer is afterwards assigned an e-mail address the username of e-mail and SCM system have to be identical for organizational and practical reasons. If the username in the e-mail address is different from the CVS username, CVSAnalY syncs with the former one and the actions done with both usernames are considered as done by a unique developer.

The following is a CVS log excerpt for the AUTHORS file of the KDevelop project[11]. It gives the last three revisions (from revision 1.47 to 1.49) done during the last months of the year 2003 until mid-2004. Log messages from other SCM systems, such as Subversion, *git* or *Bazaar* look similar.

```
[...]
RCS file: /mirrors/kde//kdevelop/AUTHORS,v
Working file: /mirrors/kde//kdevelop/AU-
THORS
head: 1.49
branch:
locks: strict
access list:
keyword substitution: kv
total revisions: 103;   selected revisions: 103
description:
----------------------------
revision 1.49
date: 2004/06/21 18:57:13; author: rgruber; state:
Exp; lines: +4 -0
Added self
----------------------------
revision 1.48
date: 2004/02/24 14:42:59; author: dagerbo; state:
Exp; lines: +5 -1
Added self :)
----------------------------
revision 1.47
```

date: 2004/02/15 22:40:33; author: aclu; state: Exp; lines: +3 -3
Some more credits update.
[...]

While being parsed each file is also matched for its type. Usually this is done by looking at its extension, although other common filenames (for instance README or TODO) are also looked for. The goal of this separation is to identify different contributor groups that work on the software, so besides source code files the following file types are also considered: documentation (including web pages), images, translation (generally internationalization and localization), user interface and sound files. Files that do not match any extension or particular filename are accounted as unknown. This discrimination follows the criteria that have been presented in section 3.2, although it lacks the possibility of looking at the content of the files as we only consider filenames (because this is the only information that appears in the CVS logs).

CVS also has some peculiarities when introducing contents for the first time (this action is called initial check-in). The initial version (with version number 1.1.1.1) is not considered in our computation as it is the same as the second one (which has version number 1.1). The number of aggregated and removed lines in CVS are computed from this initial version. This means that the first commit (the initial check-in) logs as if 0 lines were added. This does not correspond to reality. In order to obtain the actual number of LOCs in the first version we count the LOCs by means of the UNIX *wc* tool[12] of the latest version, subtracting the added lines and adding the removed lines of all the other commits.

Comments attached to commits are usually forwarded to a mailing list so that developers keep track of the latest changes in CVS. Some projects have established some conventions so that certain commits do not produce a message to the mailing list. This happens when those commits are supposed to not require any notification to the rest of the development team. A good example of the pertinent use of *silent* commits comes from the existence of bots that do several tasks automatically.

In any case, such conventions are not limited to non-human bots, as human committers usually may also use them. In a large community -as it is the case for the ones we are researching- we can argue that *silent* commits can be considered as not contributory (i.e., changes to the head of the files, for instance a change in the license or the year in the copyright notice, or moving many files from one location to another). Therefore, we have set a flag for such commits in order to compute them separately or leave them out completely in our analysis.

For instance, the developers of the KDE project mark such commits with the comment *CVS_SILENT* as it can be seen from following log excerpt extracted from the kdevelop_scripting.desktop file of the KDevelop CVS module. In this case it is due to a change to a *desktop* file, a file type that is related to the user interface. Being this change not considered interesting for other developers to know about, the author of this commit decided to make this commit *silently*.

[...]
RCS file: /mirrors/kde//kdevelop/kdevelop_scripting.desktop,v
Working file: /mirrors/kde//kdevelop/kdevelop_scripting.desktop
head: 1.24
branch:
locks: strict
access list:
keyword substitution: kv
total revisions: 30; selected revisions: 30
description:
----
revision 1.24
date: 2005/03/28 03:29:25; author: scripty; state: Exp; lines: +2 -2

CVS_SILENT made messages (.desktop file)

[...]

Write access to the SCM system is not given to anyone. Usually this privilege is granted only to those contributors who have reached a compromise with the project and the project's goals. But external contributions -commonly called patches, that may contain bug fixes as well as implementation of new functionality- from people outside the ones who have write access (committers) are always welcome.

It is a widely accepted practice to mark an external contribution with an authorship attribution when committing it. Thus, we have constructed certain heuristics to find and mark commits due to such contributions. The heuristics we have set up are based on the appearance of two circumstances: patch (or patches in its plural form) together with a preposition (from, by, of, and other) or an e-mail address or an indication that the code had been attached to a bug fix in the BTS. The regular expressions that have been used are following:

[a] patch(es)?\s?.* from [f] patch(es)?\s?.* by
[b] patch(es)?\s.*@ [g] @.* patch(es)?
[c] ?s.* patch(es)? [h] s? .* patch(es)?
[d] patch(es)? of [i] <.* [Aa][Tt] .*>
[e] attached to #

As an example, the following slightly modified excerpt taken from the kdevelop.m4.in file from the KDevelop module in the KDE CVS repository shows a patch applied by a committer with username "dymo" that was submitted originally by Willem Boschman:

[...]
----
revision 1.39
date: 2004/06/11 17:07:57; author: dymo; state: Exp; lines: +3 -3
Applied patch from Willem Boschman -
fix builddir != srcdir configuration problem.

[...]

All these efforts have in common that they perform text-based analysis of the comments attached by committers to the changes they perform. The range of possibilities in this sense is very ample. For instance, Mockus *et al.*, and later on in an enhanced manner Amor *et al.*, have tried to identify the reasons for changes (classifying changes as adaptative, perfective or corrective) in the software using text-analysis techniques (Mockus and Votta, 2000, Amor et al., 2006).

Data Treatment and Storage

Once the logs have been parsed and transformed into a more structured format, some summarizing and database optimization information is computed and data is stored into a database.

Usually the output of the previous parsing consists of a single database table with an entry per commit. This means that information is stored in a raw form, the table containing possibly millions of entries depending on the size and age of a project. Information is hence in a raw format and in an inconvenient way if we consider getting statistical information for developers and projects from it.

A first step in this direction is to make use of normalization techniques for the data. In this sense, committers are assigned a unique numerical identification and if further granularity is needed, procedures have been implemented to do the same at the directory and file level. For the sake of optimization this has been introduced during the parsing phase so additional queries do not have to be performed. The next step is to gather statistical information on both committers and modules. These additional tables will give detail on the interactions by contributors or to modules,

which is one of the most frequent information that is asked.

Additional information that makes longitudinal analyses possible is the evolution of contributions by developers and to modules. Hence, the same statistical queries that have been obtained for committers and modules for the summarizing tables can be obtained in a monthly or weekly basis since the date the repository was set up.

On the other hand, unfortunately CVS does not keep track of which files have been committed at the same time. The absence of this concept in CVS may bring some distortion into our analysis. We have therefore implemented the sliding window algorithm proposed by German (Germán, 2004) and Zimmermann et al. (Zimmermann et al., 2005) that identifies atomic commits (also known as *modification requests* or transactions) by grouping commits from the CVS logs that have been done (almost) simultaneously. This algorithm considers that commits performed by the same committer in a given time interval (usually in the range of seconds to minutes) can be considered as an atomic commit. If the time window is fixed, the amount of time that is considered from the first commit to the last one is a constant value. For a sliding time window, the time interval is not constant; the time window is restarted for every new commit that belongs to the same transaction until no new commit occurs in the (new) time slot (Zimmermann et al., 2005).

The post-process is composed of several scripts that interact with the database, statistically analyze its information, compute several inequality and concentration indexes and generate graphs for the evolution over time for a couple of interesting parameters (commits, committers, LOCs...). Results are shown through a publicly accessible web interface that allows easy inspection of the whole repository (general results), a single module or by committers[13]. Therefore results are available for remote analysis and interpretation by project participants and other stakeholders.

MAILING LISTS ARCHIVES (AND FORUMS)

Mailing lists and forums are the key elements for information dissemination and project organization in libre software projects. Without almost any exception, libre software projects provide one or more mailing lists. Depending on the project, many mailing lists may exist for several target audiences. So, for instance, SourceForge recommends to open three mailing lists: a technical one for developers, another one to give support to users and and a third one that is used for announcing new releases.

Mailing lists are programs that forward e-mail messages they receive to a list of subscribed e-mail addresses. More sophisticated mailing list managers have plenty of functionality which allows for easy subscription, unsubscription, storage of the messages that have been sent (known as the archives), and avoidance of spam, among others.

Forums are web-based programs that allow visitors to interact in a similar manner as in an e-mail thread with the difference that in this case all the process goes through HTML forms and that results are visible on the web.

Both mailing lists and forums are based on similar concepts and their differences lie in their implementation and the need for different clients to participate in them. Mailing lists require the use of an e-mail client, while forums can be accessed through web browsers. As their concept is the same, there exist some software programs that transform mailing lists messages to a forum-like interface and vice-versa. Because of that, in this chapter we will only focus on mailing lists, specifically on one of the most used mailing lists managers called GNU Mailman[14] and the RFC 822 (also known as MBOX) format in which it generally stores and publishes the archives.

The RFC 822 Standard Format

As mentioned above, generally all mailing list managers offer the possibility of storing all posts (the archives) and making them publicly available through a web interfaces. This offers the possibility for newcomers to go through the history and to gain knowledge on technical as well as organizational details of a project.

The archives are also offered in text files following the MBOX format. MBOX is a format used traditionally in UNIX environments for the local storage of e-mail messages. It is a plain text file that contains an arbitrary number of messages. Each message is composed of a special line followed by an e-mail message in the RFC-822 standard format. The special line that allows to differentiate messages consists of the keyword "From" followed by a blank space, the poster's e-mail address, another blank space and finally the date the message was sent. The RFC-822 format can be divided into two parts: (a) headers, that contain information for the delivery of the message and (b) the content, which is the information to be delivered to the receiver; the standard only allows lines of text, so filtering has to be implemented if an image or other information is attached.

Mailing lists in MBOX format can be analyzed by means of the MailingListStats, or *mlstats* for short, tool[15]. Given an URL of the archives of the mailing lists, *mlstats* outputs the information extracted from the headers and the content of the message in database format for further processing and analysis.

Below is an excerpt of a post sent to a mailing list that has been stored following the RFC-822 standard. Is is an automatic message sent April 30 2005 to the GNOME CVS mailing list. This list keeps track of all the commits that are done to the CVS system of the GNOME project. This assures that subscribers are aware of the latest changes in the CVS. The content of the message, the description of the modification that had been performed, has been omitted in the excerpt.

From gnomecvs@container.gnome.org Sat Apr 30 20:16:38 2005
Return-Path: <gnomecvs@container.gnome.org>
X-Original-To: cvs-commits-list@mail.gnome.org
Delivered-To: cvs-commits-list@mail.gnome.org
To: cvs-commits-list@mail.gnome.org
X-CVS-Module: marlin
Message-Id: <20050501001636.0C5EA165E4A@container.gnome.org>
Date: Sat, 30 Apr 2005 20:16:36 -0400 (EDT)
From: gnomecvs@container.gnome.org (Gnome CVS User)
X-Virus-Scanned: by amavisd-new at gnome.org
Cc:
Subject: GNOME CVS: marlin iain
X-BeenThere: cvs-commits-list@gnome.org
X-Mailman-Version: 2.1.5
Precedence: list
Reply-To: gnome-hackers@gnome.org
List-Id: CVS Logs <cvs-commits-list.gnome.org>
List-Unsubscribe: <http://mail.gnome.org/mailman/listinfo/cvs-commits-list>,
<mailto:cvs-commits-list-request@gnome.org?subject=unsubscribe>
List-Archive: </archives>
List-Post: <mailto:cvs-commits-list@gnome.org>
List-Help: <mailto:cvs-commits-list-request@gnome.org?subject=help>
List-Subscribe: <http://mail.gnome.org/mailman/listinfo/cvs-commits-list>,
<mailto:cvs-commits-list-request@gnome.org?subject=subscribe>
X-List-Received-Date: Sun, 01 May 2005 00:16:38 -0000
[Here comes the body of the post which has been omitted in this excerpt]

From the message excerpt above, we can see some of the headers that are described in the standard. The most important ones are following: *From* (e-mail address, sometimes also real name, of the sender), *Sender* (address of the responsible entity for the last transmission), *Reply-To* (address the author wants to be replied), *To* (address(es) of the receiver(s)), *Cc* (e-mail address(es) of the receiver(s) that should receive a copy), *Bcc* (addressee(s) with *carbon* copy), *Subject* (usually contains a brief description of the topic), *Received* (contains address of the intermediate machine that has transferred the message), *Date* (when the message was sent given by the sender machine), *Message-ID* (unique identifier of this message), *In-reply-to* (Identifier of the parent message to which the current one is a response), and *References* (identifications (message-IDs) of all the other messages that are part of the conversation thread).

In addition to the data that can be found in the headers, some other information could be obtained from analyzing the content of the messages. In this regard, Weißgerber *et al.* analyze the type of patches first sent to mailing lists and later on integrated into the source code tree of the project (Weißgerber et al., 2008).

BUG-TRACKING SYSTEMS

BTS are used in libre software projects to manage the incoming error and feature enhancement reports from users and co-developers. The use of BTS is relatively extended and the most known tool in this area is BugZilla[16], a BTS developed by the Mozilla project that has been adopted by other large projects as well. Hence BugZilla is the system we study in this chapter, although conceptually all other systems should work similarly.

BugZilla allows to manage all bug reports and feature requests by means of a publicly available web interface. Besides the reports, it also offers the possibility of adding comments so that developers may ask for further information about the error or other end-users may comment it. Beyond BugZilla, other tools exist with similar features, as for instance GNATS (the one used in the FreeBSD project). SourceForge and other web platforms that support software development have implemented their own BTS for the projects they host.

Data Description

BugZilla stores in its database specific information for each bug report. The fields that can be usually found are following[17]:

- **Bugid:** Unique identifier for any bug report.
- **Description:** Textual description of the error report.
- **Opened:** Date the report was sent.
- **Status:** Status of the report. It can take one of the following status: new, assigned (to a developer to fix it), reopened (when it has been wrongly labeled as resolved), needinfo (developers require more information), verified, closed, resolved and unconfirmed.
- **Resolution:** Action to be performed on the bug. It can take following status: obsolete (will not be fixed as it is a bug to a previous, already solved issue), invalid (not a valid bug), incomplete (the bug has not been completely fixed), notgnome (the bug is not of GNOME, but of a component of another project, as for instance X window system or the Linux kernel), notabug (the issue is not really a bug), wontfix (the developers consider not to correct this error for any reason) and fixed (the error has been corrected).
- **Assigned:** Name and/or e-mail address of the developer in charge of fixing this bug.
- **Priority:** Urgency of the error. It can take following values: immediate, urgent, high, normal and low. Usually this field is modified by the bugmaster as users do not have

sufficient knowledge on the software to know the correct value.

- **Severity:** How this error affects the use and development of the software. Possible values are (from high severity to lower one): blocker, critical, major, normal, minor, trivial and enhancement.
- **Reporter:** Name and e-mail address of the bug reporter.
- **Product:** Software that contains the bug. Usually this is given at the tarball level.
- **Version:** Version number of the product. If no version was introduced, *unspecified* is given. Also, for enhancements the option *unversioned enhancement* may be chosen.
- **Component:** Minor component of the product.
- **Platform:** Operating system or architecture where the error appeared.

Usually all fields (besides the automatic ones like *bugid*, the opening date or its status) are filled out the first time by the reporter. Larger projects usually have some professional or volunteer staff that review the entries in order to adjust the information (Villa, 2003, Villa, 2005). This is especially important for fields like priority or severity as end-users hardly have no knowledge or experience on how to evaluate these fields.

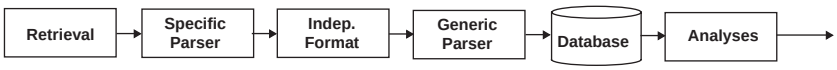
Data Acquisition and Further Processing

For the analysis of the data stored in a BTS, we have created a preliminary tool that is specifically devoted to extract data from BugZilla. The architecture of the BugZilla Analyzing Tool is described in Figure 7. Although the retrieval of the data could theoretically be simplified by obtaining the database of the BugZilla system from the project administrators, we thought that retrieving the data directly from the web interface would be more in accordance with the non-intrusive policy that all other tools described in this chapter follow.

We had to deal with several problems while retrieving the BugZilla data. After crawling for all web pages (one per bug) and storing them locally, we had to transform the HTML data into an intermediate log-type format, as not all fields were given for all bugs due probably to a transition from a previous system. Probably also because of this, there may have been some information loss and some ids could not be tracked. Other problems that we found, were the existence of wrong date entries for some bugs and comments. As the bug report ids are sequential, we could fix these entries when we found out that the date was wrong. We applied the same solution to comments with erroneous dates, as comments are also posted sequentially and cannot be introduced before the bug report has been submitted.

In recent versions of BugZilla, it is possible to obtain the data in XML format which simplifies in a great manner the data extraction[18]. When writing this chapter, the use of the XML interface was not as common as the author would wish, so retrieving the data from parsing web pages was the unique non-intrusive manner at that time. In any case, the BugZilla analyzing tool has been designed in such a way that only by removing some parts (specifically the specific HTML-parser which parses into the independent format) and by modifying the generic parser we could reuse

Figure 7. Architecture of the BugZilla analyzer



the rest of the modules without major changes using the XML query format. This is also valid for other BTS, as GNATS.

One of the issues of BTS is that in general the most relevant information in a bug report is included in natural language (usually in English) in the *Description* field . Bettenburg *et al.* have proposed a tool that extracts structural information such as source code (i.e., patches), listings, etc. from it (Bettenburg et al., 2008).

SUMMARY

Libre software projects offer a vast amount of information about their development process and the resulting product. Although this information is publicly available over the Internet, researchers should take into consideration the many *hidden* problems that may occur when obtaining and properly analyzing these data. In this chapter we have given some insight into the most data sources in research, its problems, how to circumvent them and, if possible, have provided and introduced tools that may help when researchers mine them.

Table 4 summarizes the data sources described in this chapter and the tools that can help researchers in their analysis. Regarding source code, there exist an ample amount of tools that have been used for years in corporate software engineering environments. The ones presented here have a specific target on libre software, as they address issues such as licensing of the files (FOSSology), the identification of the authors – or copyright holders – (CODD and pyTernity) or the presence of absence of documentation (*doceval*).

SCM systems are widely used in libre software projects and provide much information about the software development. The analysis of the logs of these systems, by means of tools like CVSAnalY, gives insight on the dynamics of the projects. The combination of source code and information from the SCM offers a wide range of analysis, especially concerning patterns over time. In this sense, GlueTheos and SoftChange, retrieve the sources for various points in time and extract some metrics. The final result allows to analyse how software projects have evolved over time in respect to measurements related to source code (growth, complexity, etc.), human resources (number of developers, inequality of contributions, etc.) and activity (number of commits, number of patches, etc.).

Finally, mailing lists and forums are usually the main communication channels used in libre software projects. In this chapter, we have discussed a tool, mlstats, that analyzes the information that can be found in the header of the mail messages.

Table 4. Summary of data sources, tools and purpose

Data Source	Tool	Purpose
Source Code	FOSSology	Licensing
Source Code	CODD/pyTernity	Authorship (copyright hlder) analysis
Source Code	doceval	Assessment and partial evaluation of documentation
SCM	CVSAnalY	SCM log messages analysis (evolutionary analysis)
Source Code & SCM	GlueTheos	Evolutionary analysis
Source Code & SCM	SoftChange	Evolutionary analysis
Mailing lists & forums	MailingListStat	Analysis of MBOX headers

Despite the possibilities that the vast amount of publicly available information from libre software projects offer, there are a number of problems, threats and challenges that researchers have to consider when using these data for their activities.

The first major problem comes from incomplete data sets. The cause for this is due to the fact that projects may have switched development-supporting systems and have not migrated old contents into the new system.

A threat to the use of SCM data comes from the necessity of having an account to perform changes. The original developers are in these scenarios not the ones who commit the code into the repository (committers). The validity of some research may be affected by the policies of projects, as for example the inequality of contributions may be artificially skewed towards those who have permission to do the changes in the SCM.

There is a big challenge in merging information from various sources. For instance, correlating bugs in the BTS to commits in SCM and to code in the source code is a tricky task that requires complex methods. In addition, these methods may have to be changed from project to project as the dynamics may be different among them (i.e., in some projects, committers indicate the bug report number in the commit message, while in others patches are not handled via a BTS but through a mailing list)

Finally, as mining libre software projects has become popular among scientists, many projects have suffered from an overflow of data gathering petitions, both automatically by means of tools or directly from humans in the sense of invitations to participate in surveys. In the specific case of tools, sometimes retrieving data has caused the slow down, or denial of service, of servers where the infrastructure of the project is installed, resulting in the tool being banned.

Some of the aforementioned issues are being addressed by the FLOSSMetrics[19] and the FLOSSMole (Conklin et al., 2005) projects, that have as objective to construct, publish and analyse a large scale database with information and metrics about libre software development coming from several thousands of software projects. If such initiatives succeed, researchers wanting to study the libre software phenomenon will have an ample amount of data ready to be analyzed, avoiding many of the tasks (identifications, acquisition, extraction and storage) and the threats discussed in this chapter.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful comments and suggestions. This work has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033547), QUALOSS (FP6-IST-5-033547) and QUALIPSO (FP6-IST-034763) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172) and by SEDEPECA.

REFERENCES

Amor, J. J., Robles, G., and González-Barahona, J. M. (2006). Discriminating development activities in versioning systems: A case study. In *Proceedings PROMISE 2006: 2nd. International Workshop on Predictor Models.*

Atkins, D. L., Ball, T., Graves, T. L., and Mockus, A. (2002). Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering,* 28(7):625–637.

Bauer, A. and Pizka, M. (2003). The contribution of free software to software evolution. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE),* Helsinki, Finland. IEEE Computer Society.

Bettenburg, N., Premraj, R., Zimmermann, T., and Kim, S. (2008). Extracting structural information from bug reports. In *MSR '08: Proceedings of the 2005 Working Conference on Mining software repositories.*

Capiluppi, A. (2004). Improving comprehension and cooperation through code structure. In *Proceedings of the 4th Workshop on Open Source Software Engineering, 26th International Conference on Software Engineering*, Edinburg, Scotland, UK.

Capiluppi, A., Lago, P., and Morisio, M. (2003). Evidences in the evolution of OS projects through changelog analyses. In *Proceedings of the 3rd International Workshop on Open Source Software Engineering*, Orlando, Florida, USA.

Capiluppi, A., Morisio, M., and Ramil, J. F. (2004). Structural evolution of an Open Source system: a case study. In *Proceedings of the 12th International Workshop on Program Comprehension*, pages 172–183, Bari, Italy.

Chang, H.-F. and Mockus, A. (2008). Evaluation of source code copy detection methods on FreeBSD. In *MSR '08: Proceedings of the 5th Working Conference on Mining software repositories.*

Conklin, M., Howison, J., and Crowston, K. (2005). Collaboration using OSSmole: A repository of FLOSS data and analyses. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 126–130, St. Louis, Missouri, USA.

Germán, D. M. (2004). Mining CVS repositories, the softChange experience. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburgh, UK.

Germán, D. M. and Hindle, A. (2005). Visualizing the evolution of software using softChange. *Journal of Software Engineering Knowledge Engineering.* Accepted for publication, under revisions.

Ghosh, R. A. and Prakash, V. V. (2000). The orbiten free software survey. *First Monday*, 5(7).

Ghosh, R. A., Robles, G., and Glott, R. (2002). Software source code survey (free/libre and open source software: Survey and study). Technical report, International Institute of Infonomics. University of Maastricht, The Netherlands.

Gobeille, R. (2008). The fossology project. In *MSR '08: Proceedings of the 5th Working Conference on Mining software repositories.*

Graves, T. L. and Mockus, A. (1998). Inferring change effort from configuration management databases. In *5th IEEE International Software Metrics Symposium*, pages 267–, Bethesda, Maryland, USA.

Hahsler, M. and Koch, S. (2005). Discussion of a large-scale open source data collection methodology. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, Hawaii, USA.

Halstead, M. H. (1977). *Elements of Software Science*. Elsevier, New York, USA.

Howison, J. and Crowston, K. (2004). The perils and pitfalls of mining SourceForge. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 7–11, Edinburg, Scotland, UK.

Lanzara, G. F. and Morner, M. (2003). The knowledge ecology of open-source software projects. In *Proceedings of the 19th EGOS (European Group of Organizational Studies) Colloquim*.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320.

Minkov, E., Wang, R., and Cohen, W. (2005). Extracting personal names from emails: Applying named entity recognition to informal text. In *Proceedings of the Human Language Technology Conference. Conference on Empirical Methods*

in Natural Language Processing, Vancouver, B.C., Canada.

Mockus, A. and Votta, L. G. (2000). Identifying reasons for software changes using historic databases. In *Proc Intl Conf Softw Maintenance*, pages 120–130.

Oman, P. and Hagemeister, J. (1992). Metrics for assessing a software system's maintainability. In *International Conference on Software Maintenance*, pages 337–344, Los Alamitos, CA. IEEE Computer Society Press.

Robles, G., Dueñas, S., and González-Barahona, J. M. (2007). Corporate involvement of libre software: Study of presence in debian code over time. In *OSS*, pages 121–132.

Robles, G., González-Barahona, J. M., and Guervós, J. J. M. (2006a). Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248.

Robles, G., Gonzalez-Barahona, J. M., Michlmayr, M., and Amor, J. J. (2006b). Mining large software compilations over time: Another perspective of software evolution. In *Third International Workshop on Mining Software Repositories*, pages 3–9, Shanghai, China.

Robles, G., González-Barahona, J. M., and Michlmayr, M. (2005). Evolution of volunteer participation in libre software projects: evidence from Debian. In *1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy.

Robles, G., Koch, S., and González-Barahona, J. M. (2004). Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In *Proc 2nd Workshop on Remote Analysis and Measurement of Software Systems*, pages 51–56, Edinburg, UK.

Robles, G., Prieto-Martínez, J. L., and González-Barahona, J. M. (2006c). Assessing and evaluating documentation in libre software projects. In *Proceedings of the Workshop on Evaluation Frameworks for Open Source Software (EFOSS 2006)*.

Spinellis, D. (2003). *Code Reading: The Open Source Perspective*. Addison Wesley Professional.

Tuomi, I. (2004). Evolution of the Linux Credits file: Methodological challenges and reference data for Open Source research. *First Monday*, 9(6). http://www.firstmonday.dk/issues/issue9_6/ghosh/

Villa, L. (2003). How gnome learned to stop worrying and love the bug. In *Otawa Linux Symposium*, Otawa.

Villa, L. (2005). Why everyone needs a bugmaster. In *linux.conf.au*, Canberra.

Weißgerber, P., Neu, D., and Diehl, S. (2008). Small patches get in! In *MSR '08: Proceedings of the 2005 Working Conference on Mining software repositories*.

Wheeler, D. A. (2001). More than a gigabuck: Estimating GNU/Linux's size. http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.

Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445.

ENDNOTES

[1] In this chapter the term "libre software" is used to refer to any software licensed under terms that are compliant with the definition of "free software" by the Free Software Foundation, and the definition of "open source software" by the Open Source Initiative, thus avoiding the controversy between those two terms.

[2] The location of the binaries may depend from system to system, although the standard location for them is the /usr/bin directory.

[3] GlueTheos is named after its purpose to glue different tools together in an easy way. Hence, this program is the *god*, *theos* in Greek, of gluing some already existing tools together. It can be retrieved from http://tools.libresoft.es/gluetheos.

[4] In the GNU coding standards, some conventions for change log files are given, see http://www.gnu.org/prep/standards/html_node/Change-Logs.html

[5] *doceval* can be obtained from https://forja.rediris.es/projects/csl-doceval/.

[6] The most current version of CODD may be found at http://libresoft.es/Tools/CODD.

[7] CODD uses as intermediate storage a file for each source package which are called the *codd* files.

[8] The most current version of pyTernity may be found at http://tools.libresoft.es/pyternity.

[9] A committer is a person who has write access to the repository and does a commit -an interaction- with it at a given time.

[10] In a SCM system there is actually no file deletion. In the case of CVS, files that are not required anymore are stored in the *Attic* and may be called back anytime in future.

[11] KDevelop is an IDE (Integrated Development Environment) for KDE. More information can be obtained from http://kdevelop.org/.

[12] *wc* is a standard UNIX tool to count lines of files, among others.

[13] See http://libresoft.es/Results

[14] The MailMan's project web site can be found at following URL: http://www.gnu.org/software/mailman/.

[15] http://forge.morfeo-project.org/frs/?group_id=33

[16] http://www.bugzilla.org/

[17] The ones shown next are the ones that can be found for the GNOME BugZilla system. BugZilla can be adapted and modified, so the fields may (and will) change from project to project.

[18] For instance, bug #55,000 from the KDE BTS, which can be accessed through the web interface at http://bugs.kde.org/show_bug.cgi?id=55000 may also be obtained in XML at following URL: http://bugs.kde.org/xml.cgi?id=55000.

[19] http://flossmetrics.org

This work was previously published in International Journal of Open Source Software & Processes, Vol. 1, Issue 1, edited by S. Koch, pp. 24-45, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 3.7
Software Platforms for Mobile Programming

Khoo Wei Ju
Malaysia University of Science and Technology, Malaysia

K. Daniel Wong
Malaysia University of Science and Technology, Malaysia

INTRODUCTION

Java 2 Micro Edition (J2ME), .NET Compact Framework (.NET CF), and Active Server Pages .NET (ASP.NET) Mobile Controls are commonly used alternatives in mobile programming. They provide an environment for applications to run on mobile devices. However, they are different in many ways, such as supported mobile devices, architecture, and development. Hence, it is important for mobile application developers to understand the differences between them in order to choose the one that meets their requirement. Therefore, in this article we will discuss the general architecture of J2ME, .NET CF and ASP.NET Mobile Controls and compare the three alternatives.

BACKGROUND AND INTRODUCTION

Since the mid-1990s, the growth of wireless communications has led to the mushrooming of mobile devices in the market. Initially, the mobile devices were mainly cell phones with limited programmability. However, many analysts and company executives were worried that mobile phone sales would eventually slow down, prompting research and development into software suitable for cell phones (Grice & Charny, 2001). Hence, now, there is a rise of programmable mobile devices. Furthermore, programmable mobile devices these days include not just cell phones but smartphones, PDAs, and pocket PCs. There are three well-known alternatives in mobile programming for general-purpose applications: J2ME, .NET CF, and ASP.NET Mobile Controls.

J2ME is a version of Java that provides an application environment running on consumer devices and embedded devices. It targets machines with as little as 128KB of RAM (Tauber, 2001). J2ME consists of Java virtual machines (JVMs) and a set of standard Java application program interfaces (APIs) defined through the Java community process (JCP). J2ME can be used with different configurations and profiles, which provide specific information to a group of related devices. Configurations support the Java core

APIs. Profiles are built on top of configurations to support device-specific features like networking and user interfaces. The J2ME is available in two main configurations: connected limited device configuration (CLDC) and connected device configuration (CDC). Figure 1 shows the hierarchical structure of J2ME.

.NET CF is a lightweight version of Microsoft's .NET framework. It provides an environment for executing client-side code and eXtensible Markup Language (XML) Web services to smart devices. It is compatible with C# and Visual Basic.NET (VB.NET), and it supports (.NET Compact Framework Team, 2005):

- Windows mobile (2000, 2002, 2003)-based pocket PC,
- Windows mobile-based smartphones, and
- embedded systems running Windows CE .NET 4.1 and later.

.NET CF consists of two main components: the development environment and the runtime environment. The development environment, known as smart device extensions (SDEs), is a Visual Studio .NET (VS.NET) 2003 project type that allows .NET CF applications to be developed rapidly by simply dragging appropriate controls into the application. The runtime environment is the common language runtime (CLR). The size of the CLR and relevant class libraries is smaller than 2MB, which is suitable for mobile devices. The architecture of .NET CF is shown in Figure 2.

Active server pages (ASPs) is Microsoft's server-side scripting technology. An active server page has an .asp extension, and it mixes HyperText

Figure 1. Hierarchical structure of J2ME

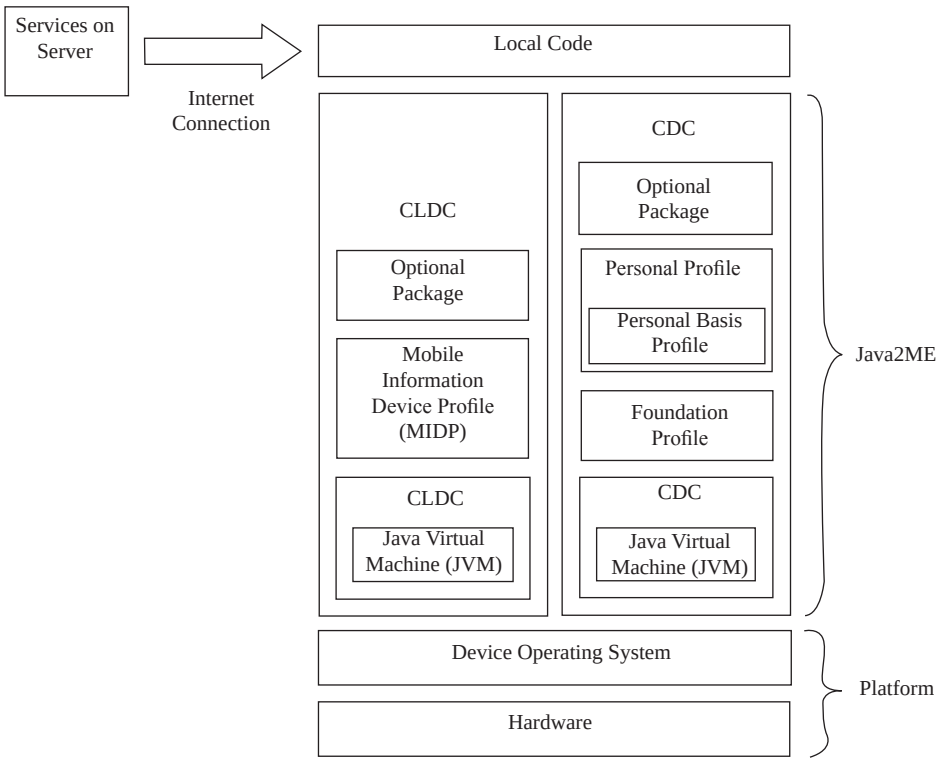
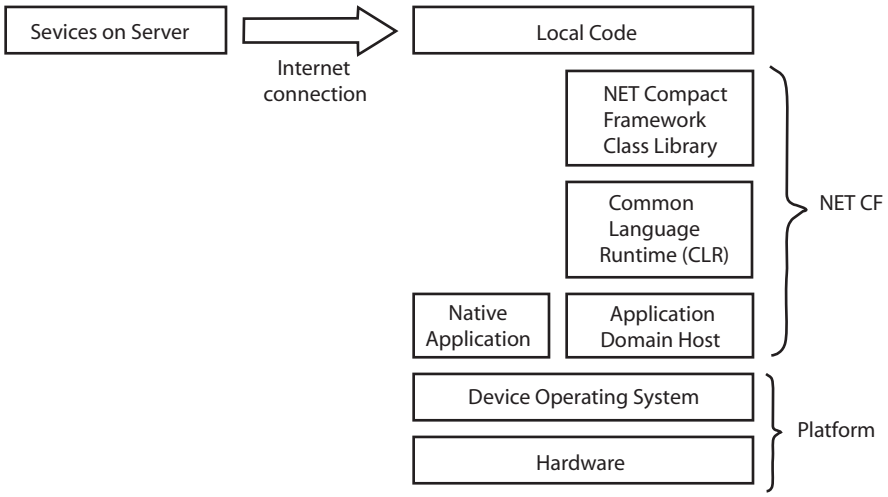


Figure 2. .NET compact framework



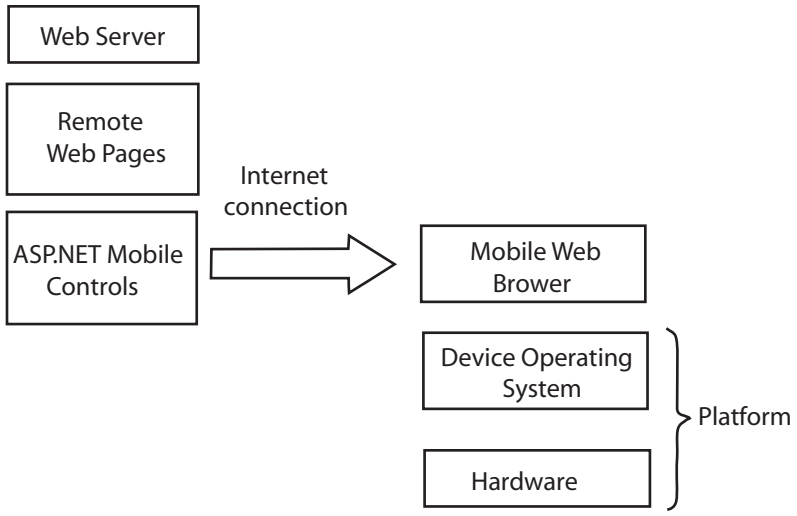
Markup Language (HTML) and scripting code that can be written in VBScript or JavaScript. ASP is distributed with Microsoft's Internet information services (IIS) Web server, so most hosts using IIS will also offer ASP for dynamic Web programming. ASP.NET is the version of ASP that works with Microsoft's .NET Framework.

ASP.NET Mobile Controls was previously known as Microsoft mobile Internet toolkit (MMIT). It was renamed as ASP.NET Mobile Controls to reinforce the concept that it is a collection of ASP.NET controls designed for mobile applications. It extends the ASP.NET server-side technology to allow developers to develop applications for a variety of mobile devices. Executing on the IIS Web server, ASP.NET Mobile Controls allows Web applications to be accessed by almost any Internet-enabled mobile device. During runtime, it will automatically detect the device running the application. The application is then transformed into a form suitable for that device. This frees the developer to concentrate on the application logic and leaves the user interface rendering to the runtime (Lee, 2002a). Furthermore, it allows developers to visually drag and drop controls on forms aimed at mobile devices using VS.NET. The rest of the work, such as writing the proper markup language (e.g., Wireless Markup Language (WML), wireless application protocol (WAP)), is handled by the toolkit. The application development environment for ASP.NET Mobile Controls should be familiar to most ASP.NET programmers. Figure 3 shows the architecture of ASP.NET Mobile Controls.

FEATURE COMPARISON

J2ME, .NET CF, and ASP.NET Mobile Controls cannot be easily compared feature-to-feature because the analysis must include non-technology aspects such as market acceptance, development and testing tools, reach, standardization, and platform coherence. Besides, the final releases of J2ME's mobile information device profile (MIDP), personal basis profile, and personal profile are still in production (Sun, 2005). On the other hand, .NET CF is in the final stages of its beta tests. Nevertheless, a feature comparison, although limited, should still be useful.

Figure 3. ASP.NET mobile controls



Flexibility of Machine Control and Scope of Applications

Virtual Machines, Pointers, Native Features

In the .NET Compact Framework, the common language runtime (CLR) environment executes .NET's Microsoft Intermediate Language code. The CLR also offers support services, such as code verification, memory, and code security. The managed code is always translated into native machine code rather than interpreted. CLR supports interfaces and pointers. As for security policy, .NET CF grants full trust to all code (Microsoft, 2005b). The standard frameworks cover only a limited set of commonly used mobile device features. Other features are accessible via native methods. Besides, it is believed that .NET CF has better support for native methods than J2ME because Microsoft controls both .NET CF and the Windows operating system (Yuan, 2002).

With J2ME, Java source code is compiled into machine-independent byte code. The byte code is then interpreted by the Java virtual machine (JVM) during runtime. J2ME employs different versions of the JVM based on the needs of a particular situation. The configuration specifications define the characteristics of the J2ME virtual machines. In most cases, features of the JVM are removed to accommodate the needs of a configuration. The CDC runs on a C-virtual machine (CVM) that is fully compliant with the Java virtual machine specification. The CDC profile accommodates devices with as little as 512kB of memory, although it is really designed for platforms with about 2 MB of available memory (White & Hemphill, 2002). Sun provides a reference implementation of the CLDC specification that is based on the KVM, a small footprint of JVM that satisfies the CLDC requirements. However, products need not be based on KVM—any virtual machine that has the features required by the specification and can work within the resource restrictions of the CLDC environment can be used (Topley, 2002). Although JVM supports interfaces, it does not support pointers because it can result in unsafe code. The Java native interface (JNI), which allows access to native methods, can be used but only by CDC. For CLDC, the native features must be built into the runtime.

Consumer Applications, Multimedia, Gaming

.NET CF supports direct draw on canvas, double buffering, and device button remapping through its rich Windows Forms User Interface library. It also supports multimedia playback by using the native methods from Windows Media Player on Pocket PC (Yuan, 2002).

In J2ME, the mobile information device profile (MIDP) 2.0 for CLDC includes animation and game controls in the javax.microedition. lcdui.game package. Multimedia playback is supported via the Java media framework (JMF) on the CDC or the multimedia optional package for the CLDC. Many game developers prefer J2ME, because it is supported by a wider range of mobile platforms.

Development Support

Programming Languages

ASP.NET supports any language supported by the .NET Framework, including C, C++, C#, Visual Basic, and even Java. However, .NET CF currently supports only two major .NET languages: C# and VB.NET (Microsoft, 2005a). C# and VB.NET are standardized by EMCA and ISO/IEC. Hence, Microsoft has long been criticized for tightly controlling its technologies. However, the support of multiple standardized languages allow developers flexibility in programming in .NET CF.

J2ME only supports Java. Anyone can propose a Java specification request (JSR) to the Java community process (JCP) for a new platform extension. Unlike with the tightly controlled development of .NET compact framework and ASP.NET, it may appear that under a more free process like the JCP, developers have to spend much time understanding the features to make use of all extensions in the language. However, J2ME APIs undergo rigorous standardization processes to ensure wide industry support and minimum learning for developers.

Platforms

.NET CF supports high-end PDAs such as Windows pocket PCs, Windows smartphones, and embedded devices running on the Windows CE .NET platform (Microsoft, 2005a). Windows devices consist of only a small part of today's mobile device population.

With J2ME, most of the cell phone devices (Motorola iDEN, Nokia Symbian OS, and Qualcomm Brew platforms) and low-end PDAs (Palm OS and Real-Time OS platforms) have built-in Java support because Java allows developers to be productive across many mobile platforms.

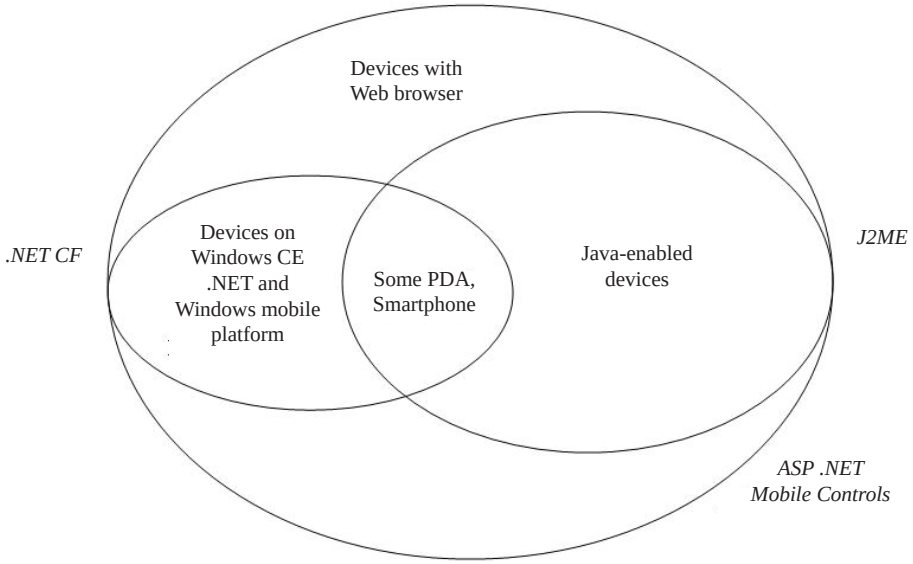
Because it is server-side based, placing minimal requirements on the client, ASP.NET is supported by the widest range of mobile devices, including all devices that support .NET CF, all devices that support J2ME, and more. However, each of these devices may present the output of the ASP.NET controls differently due to their different limitations and capabilities.

Development Tools

Regarding .NET CF and ASP.NET, Visual Studio .NET provides similar design interfaces for both mobile and non-mobile applications. It supports Web services integration and relational database access, and VS.NET is tightly integrated with Visio Enterprise Network Tools edition, which can generate C# or VB.NET code from UML (Unified Modeling Language) diagrams. Furthermore, VS.NET supports debugging on both emulators and real devices. However, VS.NET is not free.

Sun's J2ME Wireless ToolKit is a widely used MIDP development tool. Furthermore, command-line tools and vendor-specific toolkits are readily available. All major Java integrated development environments (IDEs) have J2ME modules or plug-

Figure 4. Devices supported by .NET CF, J2ME, and ASP.NET mobile controls



ins. A big challenge for all J2ME IDEs is vendor software development kit (SDK) integration. Every device vendor provides SDKs for their device emulators and proprietary J2ME extensions. The unified emulator interface (UEI) is designed to standardize the interfaces between IDEs and device SDKs. However, the UEI is available only through a Sun licensing program.

Miscellaneous

Specification Process

When a new technology emerges, Microsoft has the veto power to make decisions and make it available on .NET CF and/or ASP.NET. This saves time and effort. On the other hand, this also means that developers have no say on the specification process.

The Java community process (JCP) decides the new J2ME standard APIs, whereas Sun has veto power on only Java language specifications. The JCP develops all current J2ME configurations, profiles, and optional packages, so the specification process is arguably very lengthy and inefficient. However, some developers like this because more people are allowed to contribute and decide.

Gateways

There are technical difficulties in using .NET CF in mobile gateways because it was not designed to run lightweight application servers required in mobile gateways. Although Microsoft mobile information server (MIS) is a powerful gateway, messaging, and synchronization server, .NET CF lacks built-in APIs to interact with Microsoft MIS (Yuan, 2002).

For J2ME, the primary mobile service gateway product is from IBM. The Oracle9i wireless application server and Oracle J2ME SDKs provide gateway integration points for mobile devices to many other Oracle or third-party application servers (Yuan, 2002).

Additional Comparisons Between .NET CF and ASP.NET Mobile Controls

Server-Client Side

.NET CF uses client-side technology. Code is executed on the mobile device using just-in-time (JIT) compilation and native execution. ASP.NET uses server-side technology. Code is executed on the server, producing markup-language-based output such as HTML to be interpreted by a Web browser.

Web Server

For .NET CF, a Web server is not needed because code is executed directly on the device. For ASP.NET, a Web server (such as Microsoft IIS 5.0 or 6.0) that supports ASP.NET is required. The ASP.NET HTTP runtime is used to handle and process requests via a set of ASP.NET server controls.

Device Support

Only devices that have .NET CF runtime can execute programs written on .NET CF. On the other hand, since ASP.NET is server-side based, less processing is required on the client side compared to .NET CF. However, each of these devices may present the output of the ASP.NET controls differently due to their different limitations and capabilities.

Connectivity

For .NET CF, standalone applications can be created and installed on portable devices such as the pocket PC, pocket PC phone, and smartphone. By having the applications downloaded into such devices, the devices can either be connected or not connected. The XML or SQL Server 2000 Windows CE editions are used for local storage when working off-line. With ASP.NET Mobile Controls, an HTTP connection is required to request an ASP.NET page that uses the Mobile Controls.

FUTURE TRENDS

In recent years, a strange trend can be seen in the design of mobile devices; they are getting bigger and bigger. They are gradually taking on more

Table 1. Comparison summary between .NET CF, J2ME (CDC and CLDC) and ASP.NET Mobile Controls

	.NET Compact Framework	**J2ME Connected Device Configuration**	**J2ME Connected Limited Device Configuration**	**ASP.NET Mobile Controls**
Virtual Machine	Common Language Runtime (CLR)	Java Virtual Machine (JVM)		Common Language Runtime (CLR)
Portable Code	Intermediate Language (IL)	Byte code		Intermediate Language (IL)
Just-In-Time (JIT) Compiling	Yes	Yes		Yes
Garbage Collection	Yes	Yes		Yes
Portability	No	Yes		Yes

continued on the following page

Table 1. continued

Cross-Language Integration	Yes	No		Yes
Standardized	EMCA, ISO/IEC	Yes		EMCA, ISO/IEC
Server-Client Side	Client side	Client side		Server side
Web Server	Not needed	Not needed		Needed
Device Support	Pocket PC, smartphone, Windows CE	General-purpose Java phone, smartphone, and PDA		Device independent
Connectivity	Standalone	Standalone		Connected
Market Focus	Enterprise	Enterprise	Consumer and enterprise	Consumer and enterprise
Language Support	VB.NET, C#	Java	Java	VB.NET, C#, C++, C, Java
Platforms	Pocket PC, Windows CE	Major mobile platforms except Palm OS	All mobile platforms	All mobile platforms
API Compatibility	Subset of .NET	Subset of J2SE plus standard optional packages	Partial compatibility with CDC with additional standard optional packages	Subset of .NET
Native APIs	Platform Invoke	JNI; device and OS specific	-	-
Coding and Development Tools	Smart Device Programming (SDP), Microsoft Visual Studio .NET	Command line, vendor SDKs, CodeWarrior, and WebSphere	Command line, vendor SDKs, all major Java IDEs	Microsoft Visual Studio .NET
Specification Process	Single company	Community	Community	Single company
Service Gateway	-	Run gateways as OSGi servlets; run gateway clients via vendor-specific SDKs	Run gateway clients via vendor-specific SDKs	-
Security Model	Simplified.NET model	Full Java security manager	Limited Java 2 model supplemented by OTA specification	Simplified.NET model
Client Installation	ActiveSync, Internet Explorer download	Sync, download	Formal OTA specification	
Lifecycle Management	-	OSGi for gateway apps, J2EE Client Provisioning Specification for generic clients	Included in OTA spec, works with J2EE Client Provisioning Specification	-
User Interface	Rich subset of Windows Forms	Rich subset of AWT (Abstract Windowing Toolkit), vendor-specific UI libraries	PDA Profile subset of AWT, vendor-specific UI libraries	Rich subset of Windows Forms

continued on the following page

Table 1. continued

Mobile Database	SQL Server CE, Sybase iAnywhere Solutions(coming soon)	IBM DB2 Everyplace, iAnywhere Solutions, PointBase, Oracle9i Lite	Vendor-Specific relational implementation over RMS, Oracle SODA	SQL Server CE
Database Synchronization	Vendor specific	Vendor specific	Vendor specific	Vendor specific
XML API	Built into ADO.NET and other standard APIs	Third-party tools	Third-party tools	Built into ADO.NET and other standard APIs
E-Mail and PIM (Personal Information Manager)	Platform Invoke—Outlook APIs	PDA optional packages	PDA optional packages	-
Short Message Service (SMS)/Multimedia Messaging (MMS)	Platform Invoke—SMS/MMS	Wireless Messaging API (WMA)/WMA 2.0	Wireless Messaging API (WMA)/WMA 2.0	Simple Mail Transport Protocol (SMTP) and third party
Instant Messenger	Platform Invoke—Microsoft Network (MSN) and other IM client APIs	Third-party APIs for most IM clients including Jabber and Jxta	Third-party APIs for most IM clients including Jabber and Jxta	-
Enterprise Messaging	Platform Invoke—Microsoft Message Queuing (MSMQ)	Proprietary JMS (Java Message Service) APIs	JMS via third-party toolkits (e.g., WebSphere MQ Everyplace, iBus Mobile)	-
Cryptography	Third-party APIs	JCE (Java Cryptography Extension) and third-party libraries	Third-party libraries	Third-party APIs
Multimedia	Platform Invoke—Windows Media Player APIs	Subset of Java Media Framework (JMF)	Built into MIDP plus J2ME multimedia APIs	-
Game	Included Windows Forms UI	Direct draw on Canvas	GameCanvas support in MIDP	-
Location API	APIs provided by carriers	Location API	Location API	-

of the features of regular computers. We say this is a strange trend because mobile devices were originally meant to be stripped down, barebones devices with only the most useful features for mobile usage. Nevertheless, this trend is getting encouraging responses from users because the mobile devices are able to hold all the files they need to carry around. Due to the encouraging response from the users and the advances in technology, it is predicted that the trend will continue.

Besides, the sales of traditional PDAs have declined in the past few years (ETForecasts, 2003). This is because the PDA market is gradually being taken over by the smartphone, also known as the PDA phone. Users favor phones with computer features, such as storage capacity and clearer display (Kewney, 2005).

Currently, non-Microsoft operating systems like Symbian dominate the smartphone operating system market. As the sales and variety of smartphones are increasing worldwide, assuming Windows mobile platform keeps the same percentage of the market, the usage of Windows mobile platform will grow as well. Besides, given

the past success of Microsoft to expand into new related markets, it is quite likely that they could increase their market share over the next few years, and there is much room for them to grow. Since Windows mobile platform will support only .NET CF (Java is not included), the growth of Windows mobile platform will directly lead to the increase in the use of .NET CF in mobile programming.

Furthermore, the current versions of .NET CF grant full trust to all code. However, the upcoming versions of .NET CF will offer a subset of the policy-driven, evidence-based code access security of the full .NET framework (Microsoft, 2005b). This is a good feature from the security point of view; however, it may be less convenient for developers compared to the current version.

Hopefully in the future, both Microsoft and Java applications can coexist in the same mobile device.

CONCLUSION

.NET CF and J2ME are both excellent platforms for developing smart clients for mobile commerce applications. J2ME has already gained a lot of industry support as the most favorable platform for developing mobile applications, and there are over 500,000 skilled Java developers around the world (Wishart, 2002). J2ME implements a modular design and is portable across a variety of devices. The platform provides balanced support for both enterprise and consumer applications. J2ME vendors offer excellent selections of mobile databases and gateway application server products.

On the other hand, .NET has the advantage over J2ME where it provides a single development platform and common coding practices based around Visual Studio. There are more than 1.5 million skilled developers worldwide (Wishart, 2002), and Microsoft has the largest tools and third-party developers program. The .NET CF platform focuses on enterprise applications with rich user interface, database, and XML Web services support. Hence, .NET CF is suitable for cash-rich customers with controlled mobile environments. However, .NET CF runs only on Windows-powered high-end PDAs. As a young platform, it currently lacks support for gateway servers and choices for mobile databases.

For the near future, the choice between .NET CF and J2ME is not so much a question of the desired platform features (both are excellent in this respect) as the targeted devices. In the short run, J2ME is supported by more devices than .NET CF. In the long run, most experts expect both platforms to coexist in all market sectors. Developers must choose the right tools and make them all work in heterogeneous environments. For example, J2ME clients would need to work with .NET backend servers and vice versa. So it would ultimately not come down to a choice between J2ME or .NET CF.

Both .NET CF and J2ME have advantages over ASP.NET Mobile Controls in supporting code that will execute on the device, and that can run in disconnected, connected, or occasionally connected modes. Therefore, for most enterprise mobile solutions, .NET CF and J2ME are more appropriate.

On the other hand, if browser-based applications are required, Microsoft ASP.NET Mobile Controls can be used to develop mobile Web applications that adapt their page rendering for a range of devices, such as micro-browsers on PDAs, smartphones, and WAP phones. ASP.NET Mobile Controls allows the developers to target the users they need to target, without worrying about the device they are using.

ACKNOWLEDGMENT

The assistance of Lisa Tang in reviewing, and commenting on, a draft of this article is gratefully acknowledged.

REFERENCES

ETForecasts. (2003, June 16). *Smartphones have started to impact PDA sales.* Retrieved December 13, 2005, from http://www.etforecasts.com/pr/pr0603.htm

Faridi, M. (2003). *Beginning compact framework*. Retrieved from http://www.ilmservice.com/twincitiesnet/presentations/BeginningCF.NET.ppt

Grice, C., & Charny, B. (2001, February 2). *Wireless jungle still waiting for its king.* Retrieved from http://news.com.com/2100-1033-252009.html

Jagers, B. (2003). *Comparing file transfer and encryption performance of Java and .NET.* Retrieved from http://www.lore.ua.ac.be/Publications/pdf/Jagers2004.pdf

Kewney, G. (2005, February 8). *Landscape phones mark the resurgence of the PDA smartphone*. Retrieved December 13, 2005, from http://www.newswireless.net/index.cfm/article/1918

Lee, W. (2002a, December 2). *Developing mobile applications using the Microsoft Mobile Internet Toolkit.* Retrieved from http://www.devx.com/wireless/Article/10148

Lee, W. (2002b, November 18). *Announcing .NET Framework 1.1.* Retrieved from http://www.ondotnet.com/pub/a/dotnet/2002/11/18/everett.html

Leghari, N. (2003, December 17). *Tools and platforms: Choices for a mobile application developer*. Retrieved from http://weblogs.asp.net/nleghari/articles/mobiledeveloper.aspx

Microsoft. (2005a). *.NET Compact Framework*. Retrieved from http://msdn2.microsoft.com/en-us/library/f44bbwa1(en-us,vs.80).aspx

Microsoft. (2005b). *Security in the .NET Compact Framework*. Retrieved from http://msdn2.microsoft.com/en-us/library/13s3wxyw.aspx

Milroy, S. (2003, March 6). *.NET Compact Framework overview.* Retrieved from http://www.windowsitpro.com/Articles/Index.cfm?ArticleID=38314&DisplayTab=Article

NET Compact Framework Team. (2005, January 6). *.NET Compact Framework FAQ.* Retrieved December 6, 2005, from http://msdn.microsoft.com/smartclient/community/cffaq/default.aspx

Sun. (2005). *Java 2 Platform Micro Edition (J2ME).* Retrieved December 8, 2005, from http://java.sun.com/j2me/index.jsp

Tauber, D. A. (2001, August 3). *What's J2ME?* Retrieved from http://www.onjava.com/pub/a/onjava/2001/03/08/J2ME.html

Topley, K. (2002). *J2ME in a nutshell*. O'Reilly & Associates.

White, J.P., & Hemphill, D.A. (2002). *Java 2 Micro Edition*. Manning Publications.

Wishart, A. (2002, April 29). *Mobile development environments: .NET Contra J2ME*. Retrieved from http://www.datalogforeningen.dk/fa/fa-20020221.html

Yuan, M.J. (2002, February 21). *Let the mobile games begin, Part 1. A comparison of the philosophies, approaches, and features of J2ME and the upcoming .NET CF.* Retrieved from http://www.javaworld.com/javaworld/jw-02-2003/jw-0221-wireless.html

Yuan, M.J. (2003, May 16). *Let the mobile games begin, part 2 J2ME and .NET Compact Framework in action*. Retrieved from http://www.javaworld.com/javaworld/jw-05-2003/jw-0516-wireless.html

KEY TERMS

Active Server Page (ASP): Microsoft's server-side technology that allows scripting language for dynamically generated Web.

Cell Phone (Mobile Phone): Electronic telecommunications device that is able to move over

a wide area, connected using wireless radio wave transmission technology.

Personal Digital Assistant (PDA): Mobile device that serves as personal organizer. The basic features of a PDA include phone book, address book, task list, memo pad, clock, and calculator software.

Pocket PC: Operating platform for handheld devices introduced by Microsoft, based on the Windows CE operating system.

Smartphone: Mobile device that integrates the functionality of a mobile phone and PDA by adding telephone functions to a PDA or including the PDA capabilities on a mobile phone.

Web Service: Software system designed to support interoperability among machines over a network using a standardized interface.

Windows CE: A simplified version of the Windows operating system designed to run on handheld-size computers.

Windows Mobile: Operating system that replaced the Windows CE operating system on mobile devices. It includes a suite of basic applications for mobile devices based on the Microsoft Win32 API.

This work was previously published in Encyclopedia of Mobile Computing and Commerce, edited by D. Taniar, pp. 912-920, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 3.8
Present and Future of Software Graphics Architectures for Interactive Digital Television

Pablo Cesar
CWI: Centrum voor Wiskunde en Informatica, The Netherlands

Keith Baker
Philips Applied Technologies, The Netherlands

Dick Bulterman
CWI: Centrum voor Wiskunde en Informatica, The Netherlands

Luiz Fernando Gomes Soares
PUC-RIO, Brazil

Samuel Cruz-Lara
LORIA-INRIA Lorraine, France

Annelies Kaptein
Stoneroos, The Netherlands

ABSTRACT

This chapter aims to define a research agenda regarding the software graphics architecture for interactive digital television (iDTV). It is important to note that by iDTV we do not refer to the provision of a return path, but rather to the potential impact the user has over the television (both video stream and applications) content. We can differentiate three major topics to be included in the agenda: (1) to define a suitable declarative environment for television receivers, (2) to research television input (as multiple input devices) and output (multiple display devices) capabilities, and (3) to rethink the models of television distribution and post-distribution (e.g., peer-to-peer [P2P] networks and optical storage technologies). This

chapter elaborates on these topics.

INTRODUCTION

Digital television receivers are starting to show a reasonable level of maturity and their market penetration is becoming significant (e.g., Italy, Finland, UK, and Korea). A number of multimedia home platform (MHP) (European Telecommunications Standards Institute [ETSI], 2003, 2005) compliant receivers exist and regional standardization initiatives have joined forces by creating the Globally Executable MHP (GEM) standard (ETSI, 2004).

Still, a number of questions regarding the graphics engine (that is, the low-level software presentation control engine) and the interactive capabilities of next-generation receivers arise from both the research community and the industry. Some of the research topics include:

- Definition of a suitable declarative environment for digital television receivers, such as the synchronized multimedia integration language (SMIL) and the World Wide Web Consortium (W3C) recommendation (Bulterman & Rutledge, 2004).
- Integration of other standards, such as Moving Picture Experts Group (MPEG)-4, and Multimedia and Hypermedia information coding Expert Group (MHEG) in current standardization efforts.
- Interaction/visualization using other devices than the remote control/television set (e.g., mobile devices, tablet augmenters).
- Definition of new distribution and post-distribution models, such as P2P and optical storage devices, apart from the typical broadcast model.

This chapter is structured as follows. First, section 2 discusses the state of the art in terms of the broadcast environment, the receiver middleware, and services. Then, based on the state of the art, section 3 identifies relevant research topics in the area. Next, section 4 proposes a research agenda, and section 5 concludes the chapter.

STATE OF THE ART

Jensen (2005) has written an interesting study that categorises and defines iDTV services. He differentiates three different iDTV forms:

- **Enhanced:** Enhanced information that is sent via the broadcast channel (e.g., banners)
- **Personalized:** Automatic selection of programs by the receiver (recommendations) and personal digital recording (PDR) capabilities such as play/pause
- **Complete Interactive:** Return channel provision

He points out that, currently, only "low-technology discount solutions," referring to Nielsen usability evaluation methods, are provided. The most important discount solution today is SMS mobile phone return channel, which can evolve in the future to multimedia messaging service (MMS) solutions.

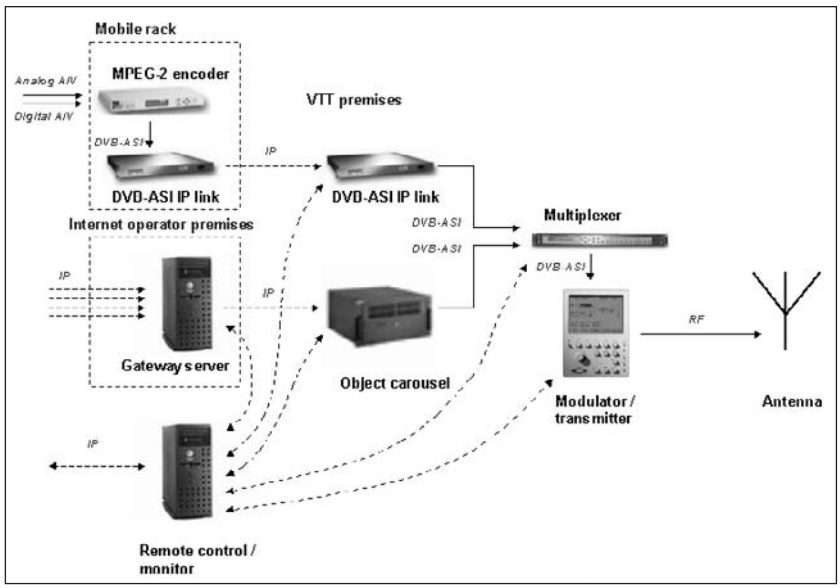
The following subsections describe the state of the art in terms of broadcast environment, software middleware, and services.

Broadcast Environment

Figure 1 depicts a typical example of a terrestrial digital television broadcast system.[1] It is composed of the following components: MPEG2 encoder, digital video broadcasting (DVB) asynchronous serial interface (ASI) internet protocol (IP) link pair, gateway server, remote control/monitor unit, object carousel, multiplexer, modulator/transmitter, and antenna.

First, the audiovisual stream is encoded with

Figure 1. Broadcast system (Cesar, 2005)



the MPEG-2 encoder. Because people in several locations might encode the audiovisual content, the encoder is stored in a mobile rack and connected to the broadcast system by using the DVB-ASI IP links. The object carousel contains application code and data. It can be uploaded using the gateway server. Next, the multiplexer generates the final MPEG-2 transport stream by combining the audiovisual content (output of the DVB-ASI IP link) and the applications (output of the object carousel). Finally, the modulator/transmitter feeds the multiplexed MPEG-2 transport stream to the antenna. The remote control/monitor can be used to remotely monitor the whole system.

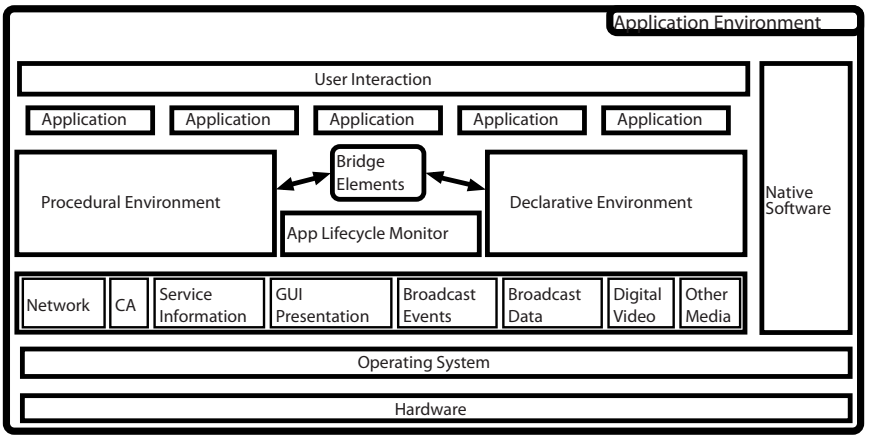
Software Middleware

Third-party development of services requires the digital television receiver to incorporate an interoperable middleware component. Three major regional initiatives exist: MHP[2] in Europe,[3] Advanced Common Application Platform (ACAP[4]) in the USA, and Association of Radio Industries and Business (ARIB[5]) in Japan.

These standardized solutions are composed of a procedural and a declarative environment, as depicted in Figure 2. The two environments do not have to be separated; bridge functionality might link them. In addition to the environments, native applications, proprietary formats, and service-specific software and content can be supported.

The procedural component includes a Java Virtual Machine (JVM) and a set of Java Application Programming Interfaces (APIs), while the declarative corresponds to an extensible markup language (XML) user agent. The declarative environment of the three mentioned standards is based on extensible hypertext markup language (XHTML), cascading style sheets (CSS), and

Figure 2. Application environment of interactive digital television receivers (Cesar, 2005)



television-specific extensions to the Document Object Model (DOM) (Cesar, 2005; International Telecommunication Union [ITU]-T, 2001, 2003, 2004; Morris & Smith-Chaigneau, 2005).

In the case of MHP, the procedural environment is called DVB-Java (DVB-J) and the declarative one is named DVB-HTML. Since the regional standards (MHP, ACAP, and ARIB) share a common approach, DVB has collaborated with the American and Japanese standardization bodies in defining a worldwide application environment: GEM. GEM has been ratified by the ITU in the J.200 (ITU-T, 2001) and J.202 (ITU-T, 2003) recommendations. GEM defines a procedural environment, DVB-J, which includes two Java APIs for graphics: Java Media Framework (JMF) and Home Audio/Video Interoperability (HAVi) (Cesar, 2005; Morris & Smith-Chaigneau, 2005).

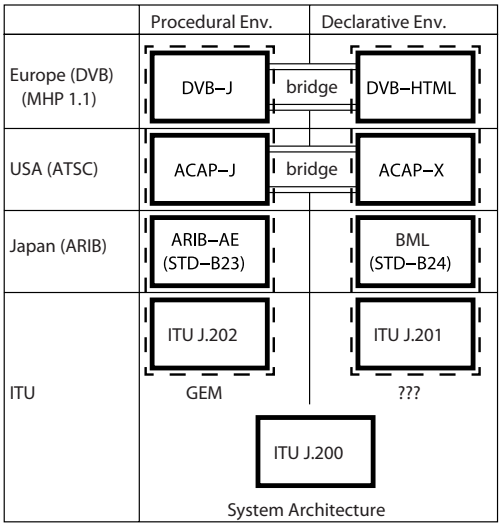
GEM defines the television display as three overlapping planes, ordered from bottom to top: background, video, and graphics. The broadcast video is normally hardware decoded and demultiplexed, and then rendered in the video layer by JMF. Alternatively, the applications are developed using HAVi and rendered in the graphics layer, requiring an alpha channel for composition purposes. HAVi extends Personal Java's *java.awt* package, including, for example, remote control events and a television-specific set of widgets.

Still, one of the challenges ahead is the definition of a common declarative environment. In order to clarify the goals of the standardization activities, their relationships are depicted in Figure 3.

Apart from conventional digital television receivers such as set-top boxes, companies are producing the next generation of optical storage devices. The two major options are Blu-ray and interactive high definition (iHD). In the case of Blu-ray, its application environment is based on the procedural environment of GEM. In the case of iHD, the application environment is HTML + Time (based on the XHTML + SMIL profile.)

Other technology options for the digital television application environment include MPEG-4, Flash, and MHEG. First, MPEG-4 video codecs are already part of the standards, while the interac-

Figure 3. Worldwide standardization situation of the application environment for iDTV (Cesar, in press)



tive part of MPEG-4 is a popular research topic, but it is not (yet) a real commercial alternative (Cesar, 2005; Baker, 2006).

Flash, on the other hand, is a popular solution for interactive multimedia on the Internet. Still, it has major problems such as: it is a non-structured technology, it uses a scripting language, it is a proprietary format, and dynamic modifications to a presentation are difficult to manage. Finally, MHEG, because of its popularity on digital television in the UK region, is a standard that will be supported by MHP.

Services

The development of MHP applications have become a lucrative business, as can be seen from the number of graphical authoring tools available on the market (e.g., Cardinal Studio[6]). The range of available services is ample;[7] some examples include games, commercials, home shopping, and banking. Nevertheless, the two major services are the electronic program guide (EPG) and the Super Teletext. The former shows information related to the scheduled audiovisual content. The latter is a new version of the traditional teletext service, in which multimedia content is broadcast within the MPEG-2 stream.

This subsection illustrates current services by taking a look at the work of a Dutch digital television company called Stoneroos. First, we will describe an adaptable EPG and then an interactive application called C-Majeur.

Stoneroos has developed a personalized program guide that uses XML TV. This way, the graphical layout of the application can be adapted. Figure 4 shows the Dutch layout, while Figure 5 shows the British Broadcasting Corporation (BBC) layout.

C-Majeur is a weekly television program about classical music. Stoneroos created an interactive episode of C-Majeur, shown in Figure 6. View-

Figure 4. Screen shot of Stoneroos program guide (Dutch layout)

Figure 5. Screen shot of Stoneroos program guide (BBC layout)

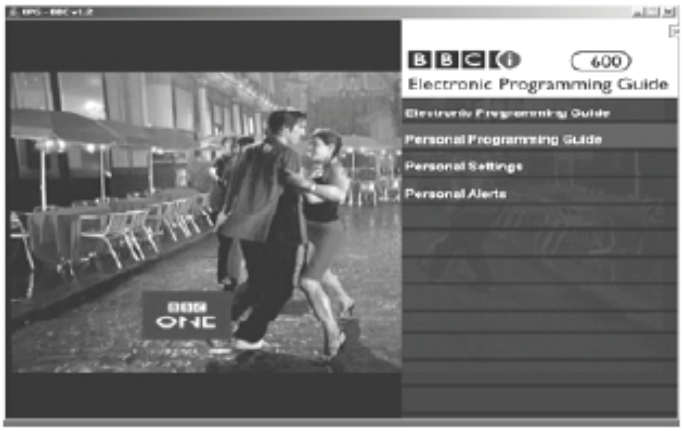
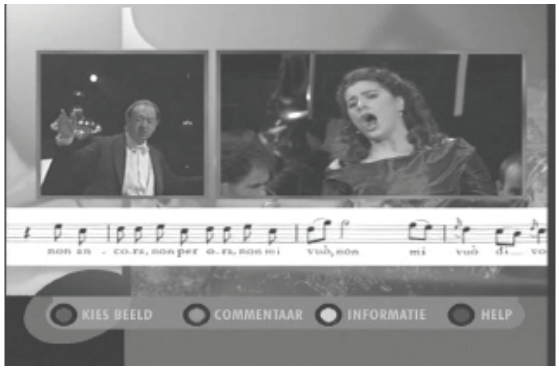


Figure 6. Screen shot of C-Majeur



ers could select extra commentary and a karaoke version of the content in which a scrolling score of the vocals was shown along with close-ups of the soloist and conductor. The interactive extras were available during the entire week following the initial broadcast.

Summary

The following list summarizes the state of the art:

- **Standards:**
  - **Procedural:** Java-based
  - **Declarative:** Based on XHTML(+SMIL), CSS, and DOM extensions
- **Services:** A number of selection-based services (e.g., EPG and Super Teletext) plus discount interactive solutions such as SMS voting
- **Current configuration:** Low-print hardware configuration (around 250MHz, 16MB-64MB of RAM, and 8MB-32MB of flash memory), remote control (input), and television set (output). Some models include storage capacity and PDR capabilities (e.g., Tivo, iHD, and Blu-ray devices).
- **Distribution mechanisms:** Broadcast

As a final summary of the current state of the art, Figure 7 depicts the current model of interactive television. The interactive content is broadcasted to the receiver, which includes an application environment. Then, this content is visualized through the television set and the user can interact with the receiver using a remote control. The thickness of an arrow represents the amount of content that is transmitted. The receiver might have a return path connected to a server. Finally, as the figure shows, even though many people can watch the television set at once, only one person can interact with it.

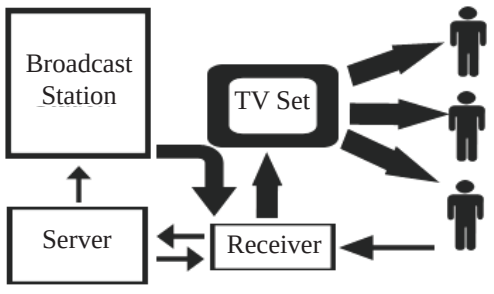
FUTURE OF INTERACTIVE DIGITAL TELEVISION

We can distinguish four different topics on which research should be focused: (1) definition of the concept interactive digital television (IDTV), (2) extensions of current standards, (3) service development, and (4) configuration of digital television receivers. The following subsections elaborate on those issues.

Definition of iDTV

One of the most important questions to be answered by this chapter is: *what do we mean by iDTV? Is it IDTV a dodo of history like the paper-*

Figure 7. Current model of iDTV



less office and the information superhighway? Have we already made of IDTV a meaningless concept?

In our opinion, IDTV is not about plugging your television to the phone line, to use your mobile phone for voting, or for that matter, to use your normal telephone to call to a television program and achieve your 15 seconds of fame. IDTV is about the potential impact a user has on the digital content. Some forms of interaction include applications that enhance the broadcast content and possibility of selecting different subtitles streams. Higher level of interaction is provided by PDR equipment in the form of pause, play, and skip content (Agamanolis & Bove, 2003) since the user actually affects the status of the content.

Further interactive capabilities include, for example, the possibility to control the content that the user consumes (i.e., personalized television) and end-user enrichment of the incoming material (Cesar, Bulterman, & Jansen, 2006). In our opinion, the research community should focus on the definition of new models that define the iDTV paradigm such as Chorianopoulos (2004) has done with his virtual channel proposal.

Standards

All in all, the definition of the GEM standard is good news for the digital television community. It is a worldwide solution for a digital television application environment. Even though its market penetration is low, apart from the Italian market in which proper subside policies have been applied, its procedural environment is a technology success (e.g., Blu-ray Java intends to reuse it).

Nevertheless, the definition of a suitable declarative environment is a challenge. Currently, apart from the structure and the visual style of the documents, the regional standards define three types of DOM events: user, internal, and broadcast generated (e.g., starting of a show). The major limitation of these standards is that they require the use of scripting for handling these events. For example, in order to synchronize the audio-visual content and an application, a script is activated when a broadcast event is received. This limitation can be overcome by adopting multimedia declarative approaches such as SMIL (Bulterman & Rutledge, 2004) or nested context language (NCL) (Soares & Rodrigues, 2005).

SMIL is the W3C standard for multimedia presentations. SMIL is a declarative format: rather than encoding the functionality of an (interactive) piece of content as a script or within the definition of a particular media codec, SMIL describes the set of interactions among media objects. Then, SMIL allows a particular renderer (or player) to implement this functionality in a manner that best suits the needs of the application runtime environment. The structure of an SMIL presentation follows the simple standard XML form:

```
<smil... >
  <head>
    general declarations for metadata,
    layout and interactive control
  </head>
  <body>
    temporal schedule of contents and
    content events
  </body>
  </smil>
```

Any object within a time container (i.e., *par*, *seq*, and *excl*) can have a scheduled or interactive begin time. The events associated with interactive begin times can be triggered by user interactions, by external script objects (via a set of SMIL DOM events), or by events that are triggered within associated content streams.

Hence, the main advantage of using SMIL is two-fold:

1. SMIL is a well-developed and widely deployed language, available on desktop and mobile devices.
2. Being declarative, SMIL code is small

(compared with Java), and it is easily verifiable—meaning that the risk of introducing viruses and other unwanted interaction side effects is limited.

As shown in Figure 8, SMIL defines 10 major functional grouping elements and attributes (modules). This results in the definition of a number of profiles. Each profile provides a fixed collection of elements and attributes, drawn from one or more modules. The purpose of the profile model is to enable the customizing of the integration of SMIL's functionality into a variety of XML-based languages. The major profiles are:

- **SMIL 2.1 Language:** The host-language version of SMIL 2.0
- **SMIL 2.1 Extended Mobile:** A rich language subset for advanced mobile devices
- **SMIL 2.1 Mobile:** A baseline set of features for general mobile devices
- **SMIL 2.1 Basic:** A baseline set of features for low-powered devices with a minimal set of elements and attributes.

In order to facilitate the use of SMIL in an IDTV application environment, W3C is working on two new profiles for SMIL:

- **SMIL iDTV:** A profile containing the major modules to support a broad range of interaction facilities in a lightweight set-top box
- **SMIL Enhanced iDTV:** A profile containing additional enhancements to allow a richer processing across a set of devices

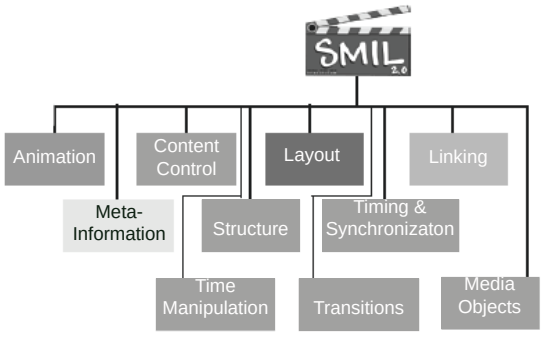
Some of the services these profiles are intended for include stand-alone applications (e.g., photo slides visualized in the television set), semi-synchronized applications (e.g., Teletext with extra audio files and animations), and services synchronized with the broadcast stream (e.g., statistics of a football match).

Another declarative solution for IDTV, similar to SMIL, is NCL. NCL claims to do for nonlinear television programs what HTML did for the Web. But, NCL is focused on temporal synchronization of multimedia objects in general and not on user interaction in particular. NCL supports the typical media objects (e.g., video and image), as well as HTML objects.

The Brazilian[8] government is currently standardizing the application environment for their digital television system. One of the alternatives is MAESTRO. MAESTRO is a complete solution, based on NCL.

There are several alternatives for the procedural and declarative middleware by using MAESTRO. The first one, shown in Figure 9, is

Figure 8. SMIL module architecture



a resident middleware with only the declarative module. Procedural codes could be launched from declarative applications because MAESTRO allows procedural objects in NCL documents.

The second alternative, shown in Figure 10, is MAESTRO coupled with a procedural middleware. Of course this alternative will require a set-top box with better performance. However, a better support for all kinds of applications is achieved.

It must be mentioned that both alternatives already presented can run not only NCL but also HTML applications. This means that any content developed for the three main digital TV systems (ARIB, ACAP, and MHP) is supported by MAESTRO.

The third alternative runs MAESTRO triggered by a procedural middleware in Java, as shown in Figure 11. MAESTRO was also implemented as an XLET that can be exported to the user set-top box. Therefore, any application developed in NCL will be able to run in any of

Figure 9. MAESTRO alone

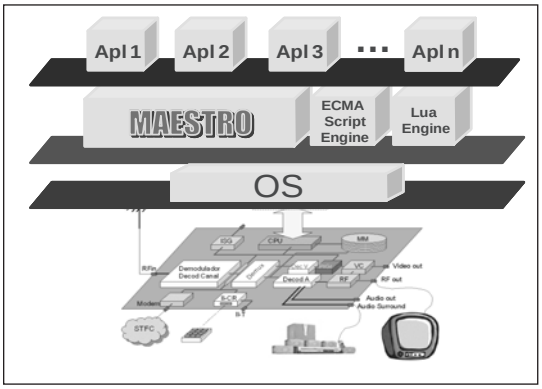


Figure 10. MAESTRO + procedural middleware

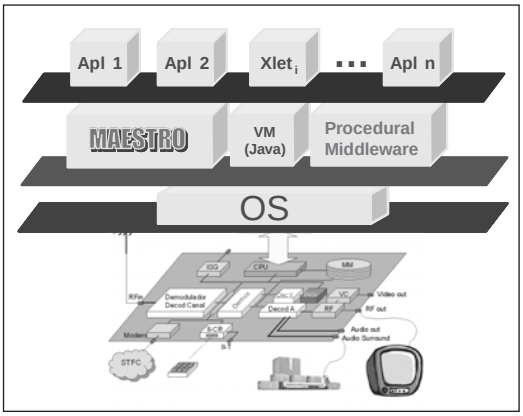
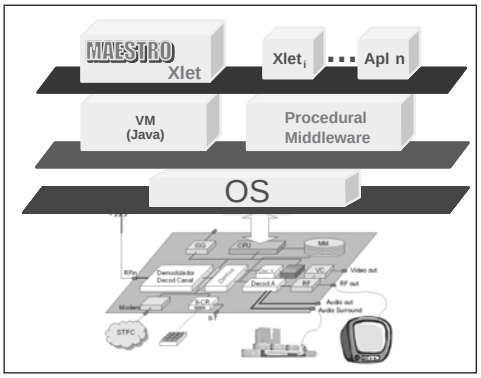


Figure 11. MAESTRO as XLET



the main digital TV systems.

Both SMIL and NCL are interesting proposals for a declarative solution that fits the television model; both are focused on the temporal synchronization of multimedia objects, while HTML was mostly intended for text content. They are different, though, in a number of issues. For example, while SMIL defines strict temporal and spatial semantics (e.g., par and seq), NCL uses templates. These templates can include any kind of semantics.

Finally, another interesting extension to current standardization efforts is to include multi lingual information framework (MLIF) (ISO AWI24616)[9] (Cruz-Lara, Gupta, & Romary, 2005) support. MLIF defines an abstract model to encode the textual information of multimedia services (e.g., subtitles, captions, and t-learning resources). Currently, different groups are working on this issue (e.g., Timed Text in W3C). Hence, MILF does not create a new format from scratch, but deals with the issue of overlap between the formats. The final intention is to create a core framework, in which all the other formats will be integrated. Their research is especially relevant in the case of services serving multilingual communities (e.g., Spain) and in the case of reusing services for different countries (e.g., European services).

Services

A number of innovative services can be provided to the digital television viewers. In this chapter we concentrate on one important topic: recommender systems (Smyth & Cotter, 2000).

Because the amount of content available to end users is continually increasing, there is a need of content personalization. This personalization is a filter and takes place prior to the actual decision of what to watch. For example, Stoneroos has developed I-Fancy. I-Fanzy is a personalized program guide shown in Figures 4 and 5. It uses TV Anytime metadata and its content can be accommodated to different layouts (e.g., Dutch and BBC layout). In our opinion, personalized television is an essential research topic because of its non-intrusive nature and the big amount of available digital content.

Other Distribution Methods

The long unforeseen threat to iDTV is the P2P network. P2P networks have proven to be the killer application of broadband access. These networks are proving to be economically very efficient, and with the BBC leading adoption, this will provide a serious alternative to internet protocol television (IPTV) networks based on client-server technologies.

P2P networks are starting to offer a complete triple play to users for TV, voice, and related IP services. Unfortunately, the type of person-to-person interactivity that a P2P network can sustain is poorly supported by an iDTV platform. If P2P networks based on "Super-peers" are the future of TV, as the BBC's iMP experiment would suggest, the research community has a clear set of challenges to address. These are challenges that can only be addressed from a user-centric viewpoint, commercial and business interests will not design these networks, they will evolve as memes. They will evolve directly in response to user needs, and if not driven by common open innovation platforms, then by the open source communities alone.

One interesting project researching these issues is the I-Share project (Pouwelse, Garbacki, Epema, & Sips, 2005).

Other Devices

In addition to traditional input/output devices, research should be focused on extended user interaction in the television domain. For example, Figure 12 shows two devices already at home that can be used for enriching the television experience.

In relation to their output capabilities, one interesting research topic is that of personal devices. Personal devices, apart of incorporating rendering capabilities, are aware of the identity of the user. Thus, these devices (e.g., a handheld device) can personalize television content in a non-intrusive manner. We propose a number of scenarios:

1. **Extra information:** Personal devices can provide enhanced information a user might be interested in, while watching a television program (Finke & Balfanz, 2002, 2004), for example, the statistics of a player during a football match. The key issue is that the information is displayed in the personal device, as thus, it does not disturb the other television viewers in the room.
2. **Audio:** Because the personal device knows the identity of the viewer, it can provide a subset of the content. Imagine, for example, an elderly man who is watching television. He might have lost some of his auditory

Figure 12. Interactive devices at home

capabilities. In this case, the personal device might render the audio of the program to the headphones of the elderly person. But, the actual shared volume is maintained at the same level for the other members of the family.

3. **Text:** Similar to the previous example, the personal device might render personally the subtitles of a movie. For example, if I am in a foreign country watching a movie with some foreign friends. I would like to receive the subtitles in my mother tongue, but, not necessarily the rest of the people watching television would like to see those subtitles.

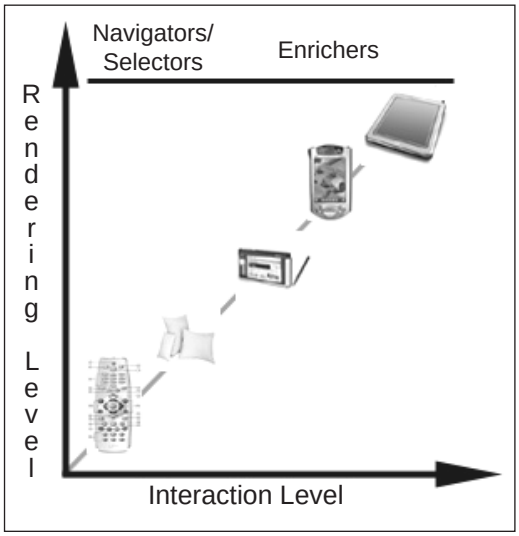
In order to implement the scenarios presented previously, we need a non-monolithic[10] renderer. A nonmonolithic renderer is one that it is capable of deliver parts of a multimedia presentation and extra information to several renderers. The audiovisual content is delivered to the television set. While, user interfaces, extra information, and specific parts of the content are delivered in a synchronized manner to other personal devices.

In addition to output devices, other devices can extend the limited remote control interaction provided by current television systems. For example, gesture recognition and voice interaction (Berglund, 2004; Berglund & Johansson, 2004) are interesting topics that need to be further researched.

Figure 13 shows a categorization of devices depending on their rendering and interaction capabilities. We can differentiate two levels of user interaction, when watching content:

- **Navigation/selection.** This level of interaction allows the user to navigate and select content (e.g., go to the next scene).
- **Enrichment.** This level of interaction allows the user to produce his/her personalized copy of the television content. Some examples of this augmentation includes subsetting the

Figure 13. Classification of devices at home



content (e.g., deleting scenes) and adding new content (e.g., to include your own audio commentary of a movie).

One example of navigation/selection is ambient technology. We can use everyday objects at home, such as an intelligent pillow, to gather information about the user. These devices can gather context-based information such as the level of excitement of a user. Then, the television renderer can act accordingly to the gathered data (e.g., by pausing the television program in case the level of excitement is too high).

Personal devices can be used to enrich content (Cesar et al., 2006). Figure 14 shows a screen shot of content augmentation (in the form of an image) over television content. In this case, the end user has highlighted one part of the screen (in this case a tie). For example, he could include, as well, some text such as "I would like this tie for my birthday!" Research in this topic is related to social TV and collaborative work. The major requirement for a content enrichment system is that the base content must not be altered (personal copies should be generated as overlapping layers of content). In addition, because television watching is an entertainment activity, the enrichment process should be fast and simple. We have termed this process as *Authoring from the Sofa*.

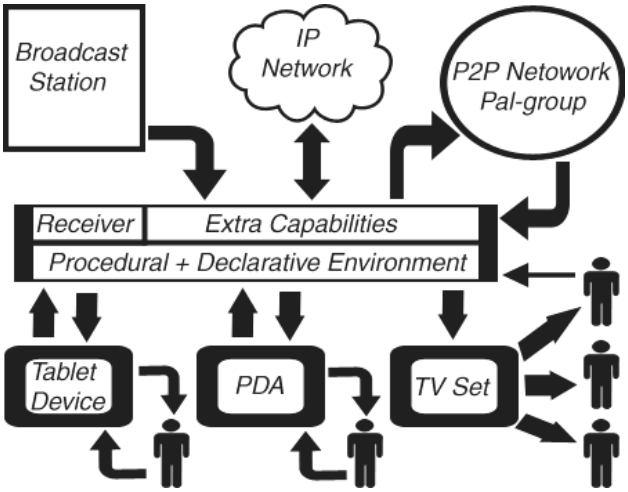
Apart from the inclusion of media objects, content enrichment includes virtual edits (e.g., alteration of the presentation timeline and exclusion of media objects) and creation of conditional navigation paths. These capabilities are researched within the European Passepartout[11] project. The scenario in this project is called Maxima. Maxima is the content manager in the house. She selects content and edits it for her children.

AGENDA

As a summary, the agenda we propose in this chapter for the future research in iDTV is depicted in Figure 15. From the distribution/post-distribution of content, digital television content (and services) can be transmitted using any kind of network (e.g., broadcast, P2P networks, and mobile networks). Then, the user can utilize any kind of device to interact with and visualize the

Figure 14. Screen shot of end-user content augmentation

Figure 15. Proposed agenda for iDTV



television content. Each of the devices provides different levels of interaction depending on their limitations. For example, the user can interact with one remote control and one television set (rather limited scenario). But, the user might interact, as well, with a personal tablet augmenter or a personal digital assistant (PDA), non-intrusive interaction is the concept to describe this kind of interaction. Finally, the receiver (and the interactive devices) includes a suitable declarative environment (e.g., SMIL, MAESTRO) and storage capabilities. The receiver might well include extra capabilities such as a recommender engine based on TV Anytime standard and a multilingual framework.

CONCLUSION

The traditional definition of iDTV as the provision of a return path focuses on the receiver capabilities instead of on the user potentiality. In order to avoid that iDTV becomes a dodo of history, we should move our focus from the receiver to the user. iDTV should not be restricted to one television set and a remote control (intrusive interaction), but extended to other digital devices at home (non-intrusive interaction).

One essential research area is to rethink the validity of the current standards and their evolution. Based on their market penetration of MHP, for example, we might conclude that they are similar to the European Constitution; the parliament of each country approves it, but the citizens are not so sure about it. In our opinion, the problem is in the acceptance timetable of new technologies. Nevertheless, standards should continue evolving (e.g., declarative environment) in order to provide the users the best possible alternatives.

Finally, we should try to model the current situation, in which the traditional role of users as consumers is shifting to become producers (e.g., blogging and personal Web pages) and distributors (e.g., iPod). Much research is needed in topics such as television content enrichment, post-distribution

mechanisms (e.g., P2P networks), adaptability of interactive services to other devices, and providing an ambient experience (Baker 2006b).

ACKNOWLEDGMENT

This work was supported by the ITEA project Passepartout and by the NWO project BRICKS.

REFERENCES

Agamanolis, S. P., & Bove, V. M., Jr. (2003). Viper: A framework for responsive television. *IEE Multimedia, 10*(3), 88-98.

Aroyo, L., Nack, F., Schiphorst, T., Schut, H., KauwATjoe, M. (2007). Personalized ambient media experience: Move me case study. In *Proceedings of the 12th International Conference on intelligent user interfaces* (pp. 298-301)

Baker, K. (2006a). Jules Verne project: Realization of reactive media using MPEG4 on MHP. In *Proceedings of the IEEE International Conference on Consumer Electronics.*

Baker, K. (2006b). Intrusive interactivity is not an ambient experience. *IEEE Multimedia, 13*(2), 4-7.

Berglund, A. (2004). *Augmenting the remote control: Studies in complex information navigation for digital TV.* Unpublished doctoral dissertation, Linkoping University, Sweden.

Berglund, A., & Johansson, P. (2004). Using speech and dialogue for interactive TV navigation. *Universal Access in the Information Society, 3*(3-4), 224-238.

Bulterman, D. C. A., & Rutledge, L. (2004). *SMIL 2.0: Interactive multimedia for Web and mobile devices.* Heidelberg, Germany: Springer-Verlag.

Cesar, P. (2005). *A graphics software architecture for high-end interactive TV terminals.* Doctoral dissertation, Helsinki University of Technology, Finland.

Cesar, P., Bulterman, D. C. A., & Jansen, A. J. (2006a). An architecture for end-user TV content enrichment. In *Proceedings of the 4th European Interactive TV Conference* (pp. 39-47).

Cesar, P., Vuorimaa, P., & Vierinen, J. (2006b). A graphics architecture for high-end interactive television terminals. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), 2*(4), 343-357.

Chorianopoulos, K. (2004). *Virtual television channels: Conceptual model, user interface design and affective usability evaluation.* Unpublished doctoral dissertation, Athens University of Economic and Business, Greece.

Cruz-Lara, S., Gupta, J. D., Fernández García, & Romary, L. (2005). Multilingual information framework for handling textual data in digital media. In *Proceedings of the Third International Conference on Active Media Technology* (pp. 82-84).

European Telecommunications Standards Institute. (2003). *Digital video broadcasting (DVB)—Multimedia home platform (MHP) specification* 1.0.3. (Doc. No. ETSI TS 101 812 v1.3.1).

European Telecommunications Standards Institute. (2004). *Digital video broadcasting (DVB)—Globally executable MHP (GEM).* (Doc. No. ETSI TS 102 819 v.1.2.1).

European Telecommunications Standards Institute. (2005). *Digital video broadcasting (DVB)—Multimedia home platform (MHP) specification 1.1.* (Doc. No. TS 101 812 v1.2.1).

Finke, M., & Balfanz, D. (2002). Interaction with content-augmented video via off-screen hyperlinks for direct information retrieval. In *Proceedings of the International Conference in Central*

Europe on Computer Graphics, Visualization and Computer Vision (pp. 187-194).

Finke, M., & Balfanz, D. (2004). A reference architecture supporting hypervideo content for IDTV and the Internet domain. *Computers & Graphics, 28*(2), 179-191.

ITU-T. (2001). *Worldwide common core—Application environment for digital interactive television services* (Doc. No. ITU-T J.200).

ITU-T. (2003). *Harmonization of procedural content formats for interactive TV applications* (Doc. No. ITU-T J.202).

ITU-T. (2004). *Harmonization of declarative content format for interactive TV applications* (Doc. No. ITU-T J.201).

Jensen, J. F. (2005). Interactive television: New genres, new format, new content. In *Proceedings of the Second Australasian Conference on Interactive Entertainment* (pp. 89-96).

Morris, S., & Smith-Chaigneau, A. (2005). *Interactive TV standards: A guide to MHP, OCAP and JavaTV.* Burlington, MA: Focal Press.

Pouwelse, J. A., Garbacki, P., Epema, D. H. J., & Sips, H. J. (2005). The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems* (pp. 205-216).

Smyth, B., & Cotter, P. (2000). A personalized television listings service. *Communications of the ACM, 43*(8), 107-111.

Soares, L. F. G., & Rodrigues, R. F. (2005). Nested context model 3.0. Part 5—NCL (Nested Context Language). (Tech. Rep. MCC-26/05, PUC-Rio). Rio de Janeiro, Brazil.

ENDNOTES

1. The architecture presented here is the one of OtaDigi. The author collaborated with this project; the intention was to set up a digital television broadcast system for the Otaniemi region, Finland. More information can be obtained in: http://www.otadigi.tv
2. http://www.mhp.org/
3. This solution has not been fully supported by the UK and France.
4. http://www.acap.tv/
5. http://www.arib.or.jp/english/
6. http://www.cardinal.fi
7. Please refer to the MHP Knowledge Project for a complete study on the issue (http://www.mhp-knowledgebase.org/)
8. It is important to remark that Brazil is the fifth most populated country in the world.
9. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37330&scopelist=PROGRAMME
10. Our first results in this topic can be found in http://www.ambulantplayer.org
11. http://www.citi.tudor.lu/QuickPlace/Passepartout/Main.nsf

This work was previously published in Interactive Digital Television: Technologies and Applications, edited by G. Lekakos, K. Chorianopoulos & G. Doukidis, pp. 91-111, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 3.9
Design Diagrams as Ontological Sources:
Ontology Extraction and Utilization for Software Asset Reuse

Kalapriya Kannan
IBM India Research Labs, India

Biplav Srivastava
IBM India Research Labs, India

ABSTRACT

Ontology is a basic building block for the semantic web. An active line of research in semantic web is focused on how to build and evolve ontologies using the information from different ontological sources inherent in the domain. A large part of the IT industry uses software engineering methodologies to build software solutions that solve real-world problems. For them, instead of creating solutions from scratch, reusing previously built software as much as possible is a business-imperative today. As part of their projects, they use design diagrams to capture various facets of the software development process. We discuss how semantic web technologies can help solution-building organizations achieve software reuse by first learning ontologies from design diagrams of existing solutions and then using them to create design diagrams for new solutions. Our technique, called OntExtract, extracts domain ontology information (entities and their relationship(s)) from class diagrams and further refines the extracted information using diagrams that express dynamic interactions among entities such as sequence diagram. A proof of concept implementations is also developed as a Plug-in over a commercial development environment IBM's Rational Software Architect.

INTRODUCTION

A Scientific American article describes evolution of Web that consisted largely of documents for humans to read and that included data and infor-

mation for computers to manipulate. In order, to help the people and machines to communicate concisely, this huge repository of information (Web) have to be re-engineered to define shared and common domain theories that are machine processable. This is the precise aim of Semantic Web to build metadata – rich Web where presently human-readable content will have machine-understandable semantics. Semantic Web achieves the increased demand of shared semantic and a web of data and information derived from it through the adaptation of common conceptualizations referred to as Ontologies. Ontologies are fundamental building blocks of Semantic Web and therefore cheap and fast construction of domain-specific ontologies is crucial for the success and the proliferation of the Semantic Web. We explore how a semantic web may impact software engineering where common conceptualizations (e.g., software, work products, experience) from previous projects need to be reused in new projects.

The chapter will contribute in the following ways: (a) It will introduce an ontology learning technique from design diagrams (specifically UML) which is in line with the research direction of building ontologies from different sources to enable a rich semantic web; (b) It will show a concrete semantic web application how the semantic web technology of ontology can promote software reuse; (c) It will put recent research on transformations from UML to OWL and vice-versa in perspective; and (d) it will motivate more research effort in semi-automatically building integrated information models.

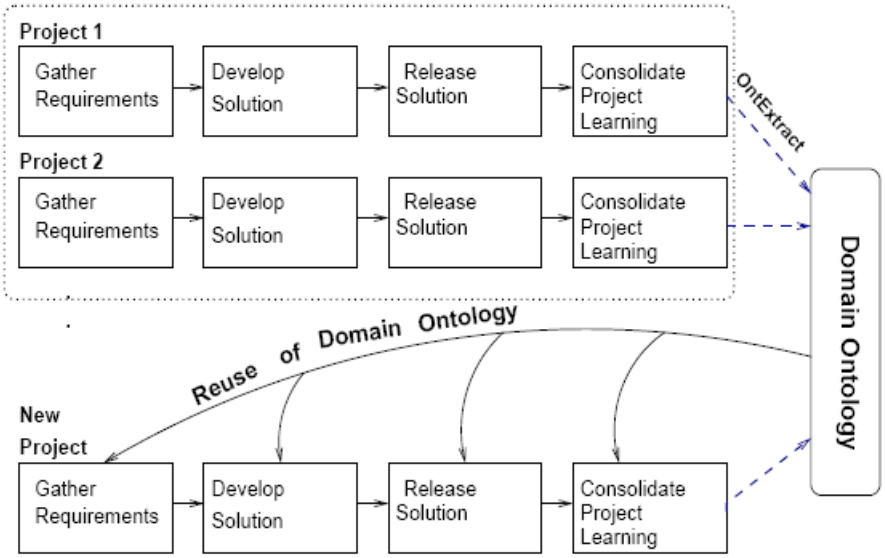
Ontology Learning and Sources

Ontology learning facilitates the construction of ontologies by ontology engineering. Ontology learning includes a number of complementary disciplines that feed on different types of unstructured, semi-structured and fully structured data in order to support a (semi -) automatic, cooperative ontology engineering process. Ontology learning is quiet a tedious task and often requires manual intervention. Ontology learning algorithms have concentrated in finding efficient methods of automatically extracting ontology information. Today, research apart from automatic ontology learning has also focused on finding sources (RDF, html, unstructured text etc) of ontological information. In our work we concentrate on identifying one such source i.e., design diagrams and provide an (semi-) automatic technique to extract (domain) ontological information. In the process of software solution development, large amount of ontological information is implicitly modeled in design diagrams. Our work aims at extracting this implicit ontology from the design diagrams. It largely helps reuse of the concepts in other software solutions.

In Figure 1, a simplified project cycle is shown. The project requirements are collected; the solution is developed and tested, and finally released to the customer. At the end of the project, a report is generated to capture the learning. However, it is rare that project-end deliverables or reports are explicitly used to improve the solution development process for new projects.

Our key observation is that the similarity among the projects alludes to the existence of a veritable domain of discourse whose ontology, if created, would make the similarity across the projects explicit. However, manually creating such domain ontologies is infeasible due to the training needed for the software professionals and the efforts required. Although many ontology engineering tools have matured over the last decade, manual ontology acquisition remains a tedious, cumbersome task that can easily result in domain knowledge acquisition bottleneck. We note that design diagrams are integral part of software project deliverables as they document crucial facets of the solution dealing with software artifacts, their relationships and runtime behavior. Much, but not all, information in theses diagrams are ontological information. If one were to extract and explicitly represent them, they would enable

Figure 1. Custom Software Projects and the Impact of OntExtract



shared and common domain theories that would not only enable better understanding of the domain but also allow software professionals to reuse the domain theories in creation of new solutions over their project cycle. This is illustrated in the lower half of Figure 1.

Using UML-2 as the design diagram representation, we present the OntExtract tool to extract ontologies from project deliverables and show that they can be reused for new projects. UML has become a de-facto standard for modeling software development activities. In UML, static behavior of the solution is captured by diagrams like the class diagram and dynamic behavior by diagrams like the sequence diagram. Two types of elements are present in UML diagrams: (a) those representing environmental artifacts representing characteristics of the problem space, and (b) those representing implementation artifacts introduced to ease a form of software implementation. We assume that the generated ontology should be general enough to span implementation trends. Our method, OntExtract, reads UML diagrams and proceeds by preserving the environmental artifacts, removing the implementation artifacts and maintaining the consistency of the resulting diagram. The output ontology is represented using UML class notations to ease the understanding by developers who may not be familiar with a new (ontology) representation language like OWL. However, using emerging UML to OWL (and vice-versa) transformation methods, the ontology can be maintained in multiple representations. Moreover, with UML diagrams from multiple projects, the emergent ontology can be refined. Over time, the ontology will capture the learning from previous projects.

The ontology can be used for any purpose including software reuse for new solution creation. For reuse, as the solution process is being followed, the initial UML diagrams for the new solution can be created by filtering the relevant

content from the ontology. As an example, we show how the sequence diagram from the new solution can be used as a requirement over the ontology and an initial class diagram for the solution can be created.

Contributions

The main contributions of our work can be listed as follows:

1. Concept of Design diagrams (in specific UML diagrams) as sources of ontological information.
2. An automatic ontology extraction technique called 'OntExtract.
3. OntExtract Methodology from UML diagrams of a single solution of a specification:
    a. Concept of identifying the elements that relate to concepts in domain.
    b. Preserving those elements that express domain concepts.
    c. Eliminating and reducing the impact of elimination of components that doesn't relate to concepts.
4. Ontology aggregation methodology from Multiple UML diagrams.
5. The impact of extracted ontology in solution building. Hence, it is important to apply semantic techniques in this domain.
6. Relationship with approaches for information transformation: UML to OWL and vice- versa, UML as ontology language.
7. Challenges in extending information models with more ontological data sources.

We have implemented OntExtract as a plug-in using IBM Rational Software Architect. It benefits two groups of people: (a) Software professionals and (b) Ontology engineers. Software professionals benefit by using this tool to obtain domain models that allow reuse of design decisions at all stages of a new software project. Ontology engineers can use our tool to create/ maintain domain ontology considering design diagrams as a new source of information. Initial results show that the created ontologies are accurate and help in better software reuse for new solutions.

BACKGROUND

In this section we provide the background knowledge about the design diagrams and the standards used for creating and representing design diagrams and ontology. We also provide the same example considered in our work for evaluation purpose.

Design Diagrams and UML 2

Design Diagrams are used to represent visually software solution for a problem. They express environmental relationship (entities and relationship among entities that represent domain concept in domain ontology) among entities in the solution developed for requirement specification of a problem. In addition design diagrams express implementation artifact that model implementation entities and relationship needed for development of software solutions. We use UML 2 since it has become a de-facto standard for modeling software development activities. UML (Terry Quatrani, 2001), is a standard modeling language for specifying, constructing, visualizing and documenting artifacts of the software system. UML consists of 9 set of diagrams to model various software artifacts. In our work we use class diagram that belongs to the category of the structural diagrams that represents static behavior of the system to extract domain concepts. Sequence diagrams are used for further refining the extracted ontology. OCL is the constraint language with UML but it is not used widely in commercial projects.

Ontology

Ontologies are meta data schemes, providing controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics and inter-relationships with other concepts. Ontology uses the following elements to express environmental artifacts: Classes represents general things in the domain of interest, Individuals defines instance of an class and includes concrete objects such as people, animals etc., Attributes describes the general properties of the classes, Relationship represents the relationships in terms of subsumption (IS-A) relationship, meronymy (HAS-A) relationship that exists among things. The OWL language (Deborah, 2004) is a leading standard for ontology representation.

Example Domain

We use the case study of student course registration taken from the Rational tutorial (Terry, 2002) as the running example in the chapter:

At the beginning of each semester students may request a course catalogue containing a list of course offerings for the semester. Information about each course, such as professor, department, and prerequisites will be included to help students make informed decisions. The new on-line registration system will allow students to select four course offerings for the coming semester. In addition, each student will indicate two alternative choices in case a course offering becomes filled or canceled. No course offering will have more than ten students. No course offering will have fewer than three students. Once the registration process is completed for a student, the registration system sends information to the billing system, so the student can be billed for the semester. Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students signed up for their course offering. The billing system will credit all students for courses dropped during this period of time.

For purpose of experimentation and evaluation, in this work we consider several solutions for the above problem. Figure 3 gives the UML class diagram for a solution we have proposed. This solution is henceforth referred to as Solution A (SolA). We consider two other solutions (class diagrams) taken from tutorial website of Rational Software Corporation (Rational, 2000), (referred to as Solution B (SolB) and class room lecture of Prof. Dr. Bettina Berendt (Bettina, 2005), (referred as Solution C (SolC)). We assume that these input class diagrams are correct. We represent classes and association between two classes as nodes and edges, respectively.

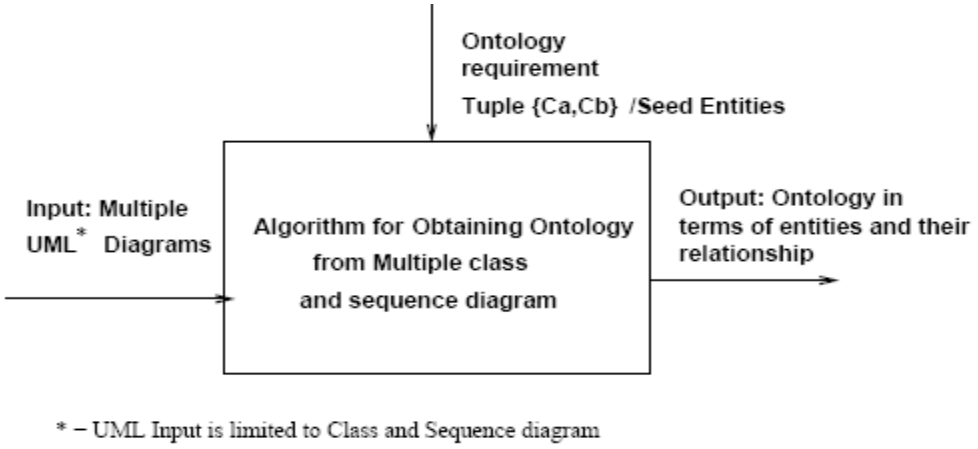
We use the terms nodes and classes interchangeably, and similarly the terms edges and associations. The term Root node is used to refer to a node that is placed at the highest level of the tree and has no parents.

USING UML DIAGRAMS AS ONTOLOGICAL SOURCES

Figure 2 gives the system architecture of OntExtract. OntExtract takes UML diagrams (class and sequence diagram) of multiple projects and extracts the domain ontology. Domain requirements are taken as a tuple of keywords.

UML elements can be classified broadly into two categories: (a) Implementation Artifacts: They are introduced by the architect of the software solution to capture those elements that represent low-level details of the solution such as messages exchanged and their behavior. Such artifacts are either full-fledged classes or specific attributes of a class. (b) Environmental Artifacts:

Figure 2. System overview



They include ontological entities and relationship among such entities. Relationship includes generalization (IS-A), aggregation/composition (HAS-A) and USING association. We use the stereotypes to identify UML elements. Implementation artifacts do not contribute to domain concepts while environmental artifacts such as IS-A, HAS-A, USING relationship and entities participating in such relationship reflects domain concepts. All UML diagrams have implementation or environmental artifacts. However, static information such as classes (concepts), attributes and relations (semantics) can be obtained from class diagrams. Therefore, in our we use class diagrams to extract domain information (ontology) and explore whether interaction diagrams such as sequence diagram would help identify more domain concepts.

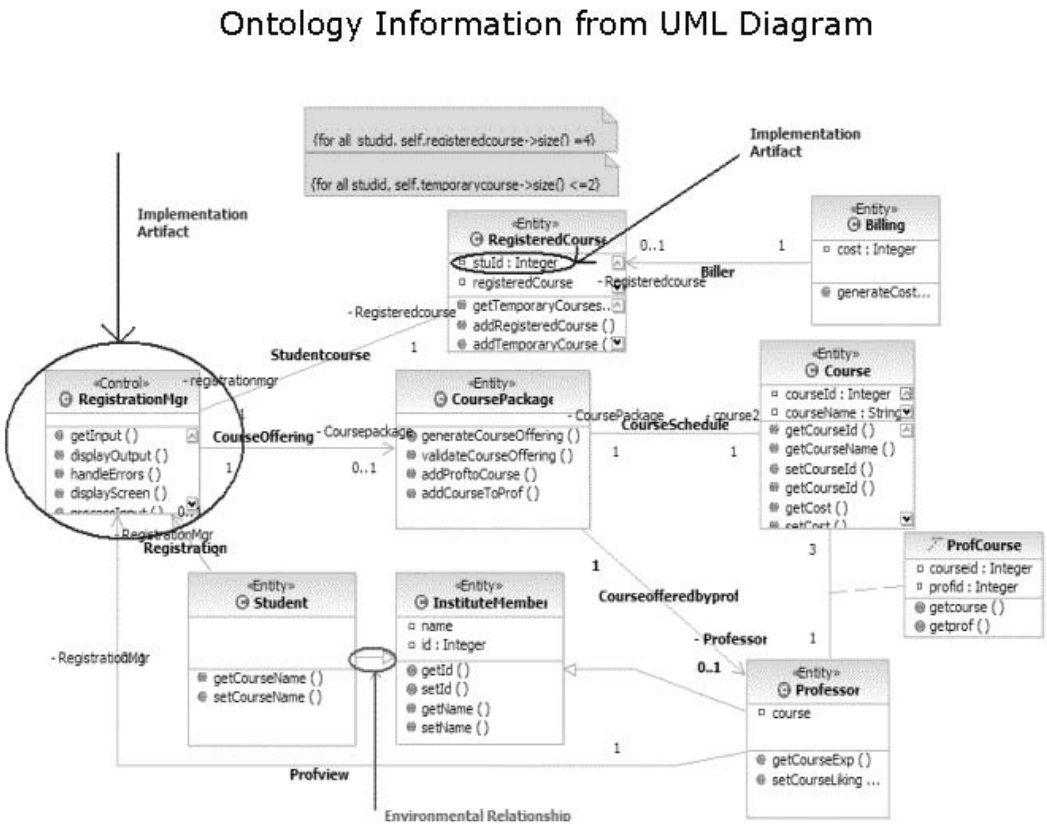
Ontology Extraction from Class Diagram

Class level implementation artifact: UML defines three types of stereotyped classes (a) Boundary class (b) Controller class and (c) Entity class. A boundary class models interaction between the software system and the environment. A control class acts as coordinator of activities. An entity class models information that has been stored by the system and its associated behavior. By and large, entity classes reflect real world entity and are part of environmental artifacts. Boundary and Controller classes are implementation artifacts, used to ease software development.

Attribute level implementation artifact: Attributes are storage elements of a class instance. Some attributes are introduced to maintain property values of an entity class while others are used purposes such as maintaining uniqueness, class level state, providing identities to entities. Attributes of the form of second category are implementation artifacts.

In Figure 3 we identify some of the implementation and environmental artifacts. In this Figure, the class RegistrationMgr is a class level implementation artifact (co-ordinates messages between other entities). Attribute 'stuID' in the class RegisteredClass is an attribute level implementation

Figure 3. Ontology information from class diagram

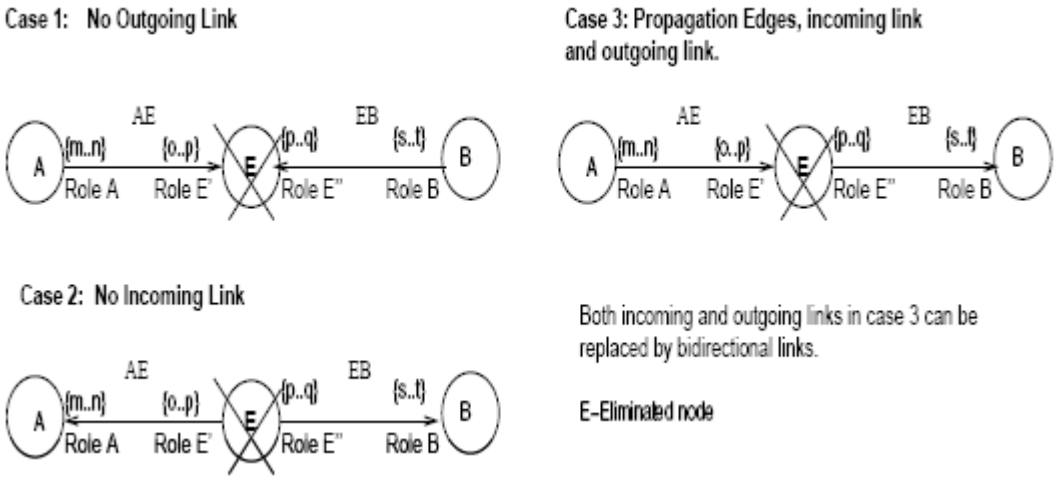


artifact introduced to maintaining uniqueness of the entity. Environmental artifacts such as class 'Student' inherits from 'InstituteMember' can also be seen in this Figure. Environmental artifacts reflecting domain concepts, OntExtract proceeds by removing implementation artifacts and preserving environmental artifacts.

OntExtract: Elimination of Implementation Artifact

Elimination of a class level attribute affects nodes that are associated with the eliminated node through inheritance, aggregation/composition and USING relationship. Nodes that are related to eliminated node through the above relationships is also eliminated. Links representing USING relationship between the neighbors of eliminated node result in dangling edges. In order to build relationship rich ontology, we aim to preserve this relationship by the process of propagation. Propagation results in new relationship (edges) among neighbors. The properties of the new relationship depends on the properties of two dangling edges that results in propagation. Figure 4 shows the nature of edges that are atypical of

Figure 4. Elimination of a node and its dangling edges



a relationship. In Figure 4, case 1 and case 2 do not have propagating links. We are interested in case 3 that has propagation edges where a new association is created between every incoming, bidirectional and outgoing links.

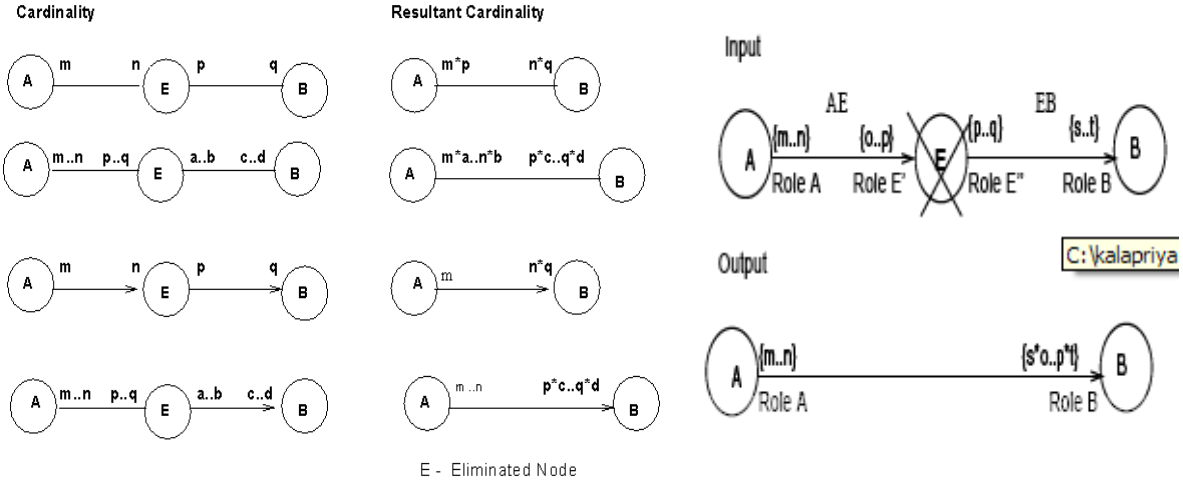
The attributes such as name, direction, cardinality, role names of the new association are obtained as follows: Direction: The direction of the new edge depends on whether the dangling edges were bidirectional or directed. New edge assumes the direction of the dangling edge(s) if the dangling edges were directed otherwise it is bidirectional. Role name: The new edges assume the role name of the nodes that the new edge connects. Cardinality: Cardinality of the new edge is obtained by finding the minimum and maximum number of instances of nodes that the new edge connects, can be associated through the node that is eliminated. In other words, the cardinality is calculated as the cross product of cardinality of the two links that results in propagation. Figure 5 (a) shows the propagation of cardinality constraints among the dangling edges.

Figure 5 (b) shows the new edges with the properties after elimination of a node (refer case 3 of Figure 4) and Figure 6 shows the class diagram of the student registration example after elimination of the class level implementation artifacts.

We leave the elimination of attribute level implementation artifacts as future work. Identifying these artifacts is difficult since there is no explicit notation is available in UML to annotate attributes according to intended usage and requires profiling of the code.

Algorithm 1 describes the steps for removing class level implementation artifact and propagating relationships. The algorithm gets and removes controller and boundary classes in the UML diagrams (line 1). A new association is created for every incoming (line 12), bidirectional edge (line 17) (to the controller/boundary class) to outgoing edge (from the controller/boundary class) line 4. The properties of the new association are obtained from the incoming/bidirectional - outgoing edge pair. In the algorithm, the terms ‘this end’ refers to the end associated with a neighbor nodes and

Figure 5. (a) Cardinality constraints propagation; (b) Role names and Direction



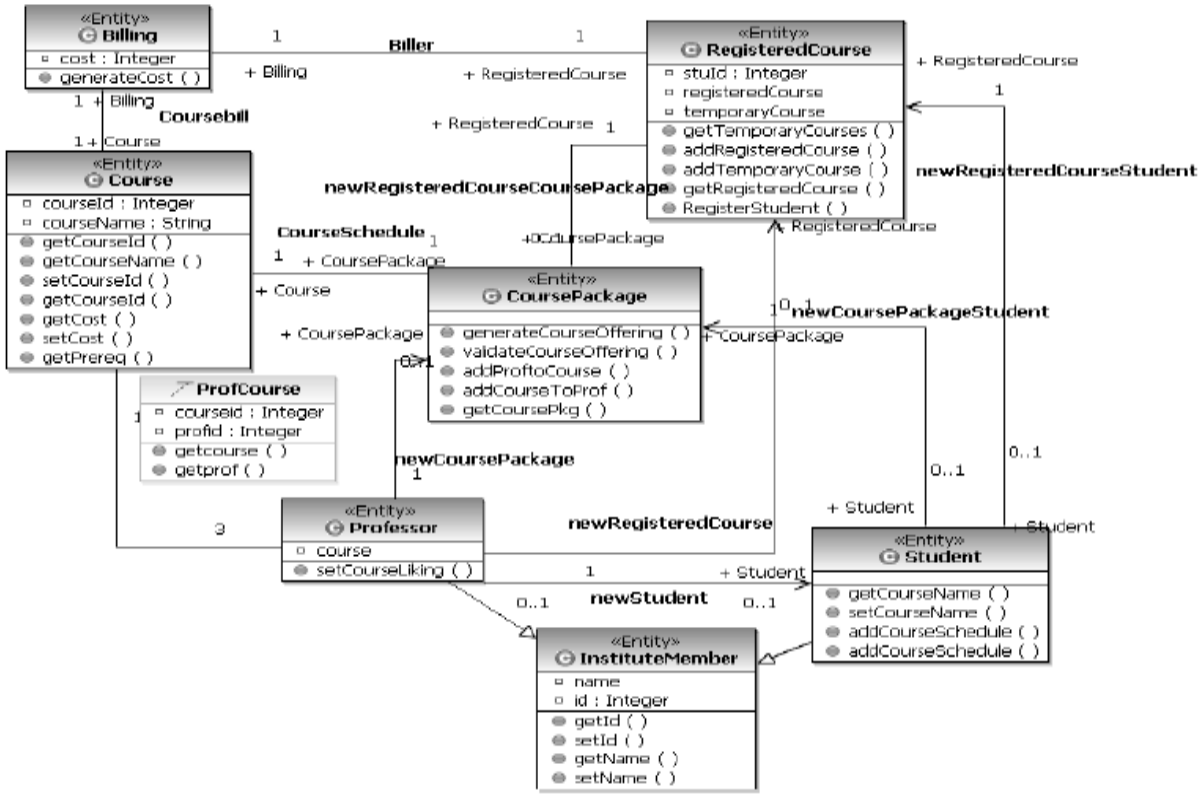
(a) *(b)*

Algorithm 1.

Algorithm 1 : Elimination of Class Level Artifacts

```
1: List lst =getControllerAndBoundaryClasses();
2: for all  (cls = classes) in lst do
3:     List lstIncoming = getIncomingLinks(cls);
4:     List lstOutgoing = getOutgoingLinks(cls);
5:     List lstbi = getBidirectionalLinks(cls);
6:     for all  lnkIn = link in lstIncoming do
7:         node thisNode = getNode(lnkIn);
8:         thisProp = getLinkProperties(lnkIn);
9:         for all (lnkOut = link) in lstOutgoing do
10:            otherNode = getNode(lnkOut);
11:            otherProp = getLinkProperties(lnkOut);
12:            createNewDirectedEdge(thisNode, otherNode,thisProp,otherProp);
13:        end for
14:        for all (lnkbi=link) in lstbi do
15:            otherEndNode = getNode(lnkbi);
16:            otherProp = getLinkProperties(lnkbi);
17:            createDirectedNewEdge(thisNode, otherNode,thisProp, otherProp);
18:        end for
19:    end for
20: end for
```

Figure 6. Student registration class diagram after elimination of class level implementation artifacts



'other end' refers to the end associated with the other neighbor, between which a new association is created. Property role name, cardinality constraints are obtained in line 8 and 16. The new association assumes a new name for reference. The resultant class diagram is henceforth referred as Eliminated Entities Class Diagram (EECD).

OntExtract: Preserving Environmental Artifact

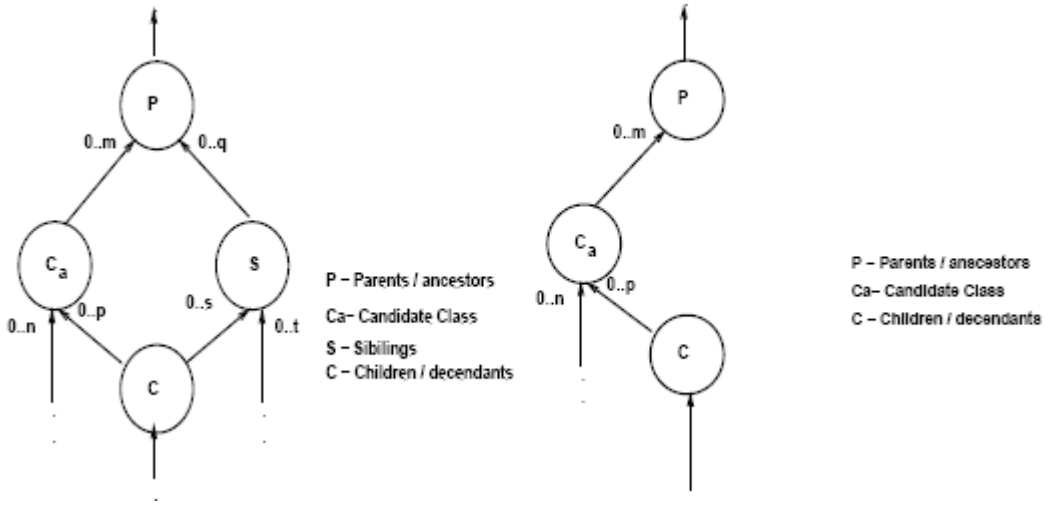
The EECD obtained as a result of elimination of implementation artifact represents in general all conceptual entities which can be directly reused in other similar software projects. In addition to this, we provide provisions for extraction of ontological information for specific domain requirements provided by the software professional. For example, in the context of 'professor' (in Figure 3) students billing information is not relevant.

The domain requirement is provided by the architect as a tuple of keywords. Let us consider an example in which we are interested in extracting student domain information i.e., input keyword is {student}. We assume that this input keyword directly maps to an entity class in EECD. This

class is henceforth referred to as the candidate class. Preserving environmental artifact includes identifying those entities that reflect domain concepts in the resultant EECD. All entities that are related through inheritance (IS-A), aggregation/ composition (HAS-A) and USING relationship (collectively known as environmental relationship) directly reflects domain information with respect to the input tuple.

For inheritance relationship, all ancestor classes influence the candidate class and all descendants classes are influenced by candidate class. Figure 7 (a) gives the generic graph of the candidate class and its neighbors in a class diagram. Therefore, with respect to inheritance relationship all ancestors and descendants of the candidate class are preserved. The siblings of the candidate class do not affect the candidate class

Figure 7. Generic Model (a) neighbors (b) inheritance output



(a) *(b)*

Algorithm 2.

Algorithm 2 Inheritance - obtaining neighbors for candidate class

```
1: ancestorNodes = getNodes(getallPathstoRoot(candidateClass)); {Get all ancestor nodes}
2: addNodestoInheritance (ancestorNodes, InheritanceRelationshipGraph);
3: for all (nodes) do
4:     if isPathtoCandidateClass(node,candidateClass) then
5:         addNodestoInheritance (nodes,InheritanceRelationshipGraph);
6:     end if
7: end for
```

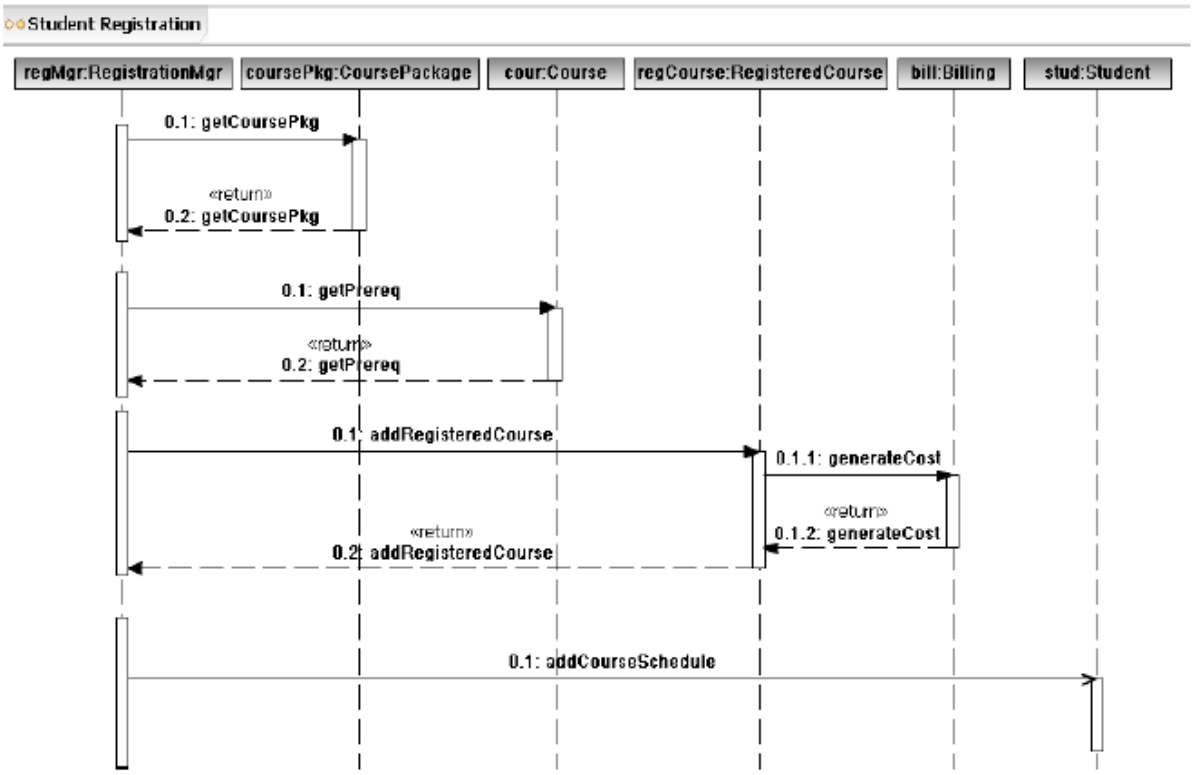
and therefore do not reflect concepts of the domain represented by the candidate class. Algorithm 2 presents the algorithm to obtain all ancestors and descendants for the candidate class. To extract the nodes that are ancestors for the candidate class, the algorithm finds all the paths from the candidate class to the root node (line 1). To find a descendant node the algorithm checks if a node has route to the candidate class in line 4. Figure 7 (b) gives the generic output structure (tree) after preserving inheritance relationship.

Preserving aggregation/composition relationship is similar to the technique presented for extracting inheritance relationship. 'USING' relationship is denoted by Directed and Bidirectional links. It represents exchange of messages between two classes during execution. We are interested in the classes that help to achieve the functionality (use case) by the relationship. Run time information such as classes related to candidate class through 'USING' relationship is not directly available from class diagram. The actual set of related classes can only be obtained from diagrams that reflect dynamic behavior (diagrams that describe the behavior of classes). Two UML diagrams, activity diagrams and sequence diagrams are candidates that provide dynamic information.

Ontology Refinement Using Sequence Diagram

Sequence diagrams describe the services provided by objects through exchange of messages.

Figure 8. Sequence diagram for functionality 'course registration'

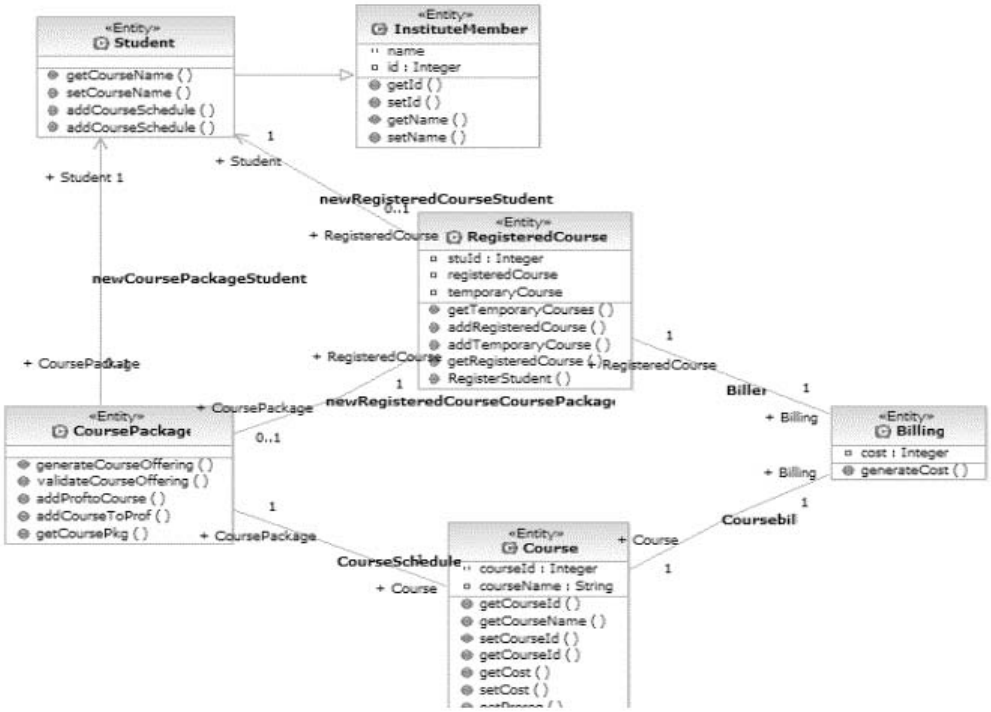


Activity diagram describe sequence of activities and is generally more expressive than the state diagram. Based on the above comparison, we propose to use sequence diagram to extract classes that are related to the candidate class through 'USING' relationship. All sequence diagrams that has the candidate class are considered. Figure 8 gives the sequence diagram for the use case 'course registration'. Classes CoursePackage, Course, RegisteredCourse, Billing that are related to the candidate class {Student} through these sequence diagrams are preserved as ontological information.

Figure 9 gives the Ontology Diagram (OD) that represents the extracted ontology information for domain requirement {Student}.

Sequence diagrams acts as filters for extracting entities that are associated with the candidate class to achieve a certain functionality (use case) or to provide a service. Using Model Driven Architecture (MDA), the architects/professionals begin by describing the use cases (functionalities/services) from the functional requirements of a problem. By modeling these services into a set of sequence diagrams our approach can be used to obtain ontology customizable for the new problem. This helps to obtain existing solution approaches for the new problem and thereby largely help software reuse.

Figure 9. Extracted Ontology Diagram (OD)



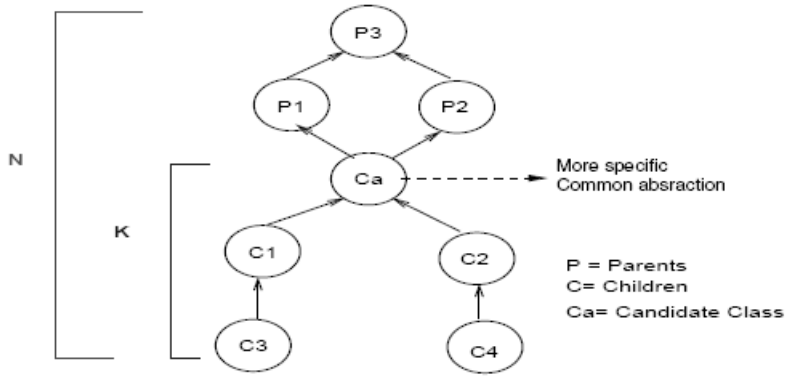
ONTOLOGY EXTRACTION FROM MULTIPLE UML DIAGRAMS

Methodology

With view of improving the quality of the extracted ontology, we propose an ontology refinement technique in which OntExtract extends to aggregate ontology from multiple Ontology Diagrams (OD) with respect to a domain requirement. The refinement process helps to correct/update/change to a fine granularity and increases the confidence of the correctness of the extracted ontology. These design diagrams can be different solution of same problem or solution(s) of different problem(s). Ontology aggregation from multiple OD can be done in two ways (a) Incremental approach, aggregates ontology by considering one relationship at a time, (b) Consolidated approach, aggregates ontology by considering all relationship in an ontology diagram. Consolidated approach increases the complexity of analysis of relationships since a single node can have multiple types of relationship. In addition, the consolidated approach does not handle propagation of relation inherently (explicit/ manual invention is required to handle propagation) whereas incremental approach handles propagation inherently (aggregation is based on all relationship but incrementally). Therefore, we use incremental approach by taking one relationship from multiple ontology diagrams to aggregate ontology.

Aggregation of ontology from multiple ontology diagrams requires adding/deleting/ updating ontology entities. We use the metric semantic distance to arrive at a decision whether to add/delete/update entities. Semantic distance measures the closeness of two entities in ontology diagram. Semantic Distance between nodes is reduced to distance between concepts in a graph. The distance between concepts is determined according to their positions (measured as levels). When a hierarchy is N = (n+1) layered, K/N nodes is connected to the class in the K^{th} level where ($0 < K < N$). Figure 10 (refer to output inheritance diagram Figure 9) diagram shows a sample inheritance graph in the ontology diagram with the candidate class in the K^{th} level. Value of K/N determines the characterization of the graph, if K/N is very low, then the relationship structure represents more generalization then specialization with respect to candidate class and vice-versa. The more general is the structure, the more re-usable is the extracted ontology. Therefore, our algorithm aims to keep

Figure 10. Semantic distance desired

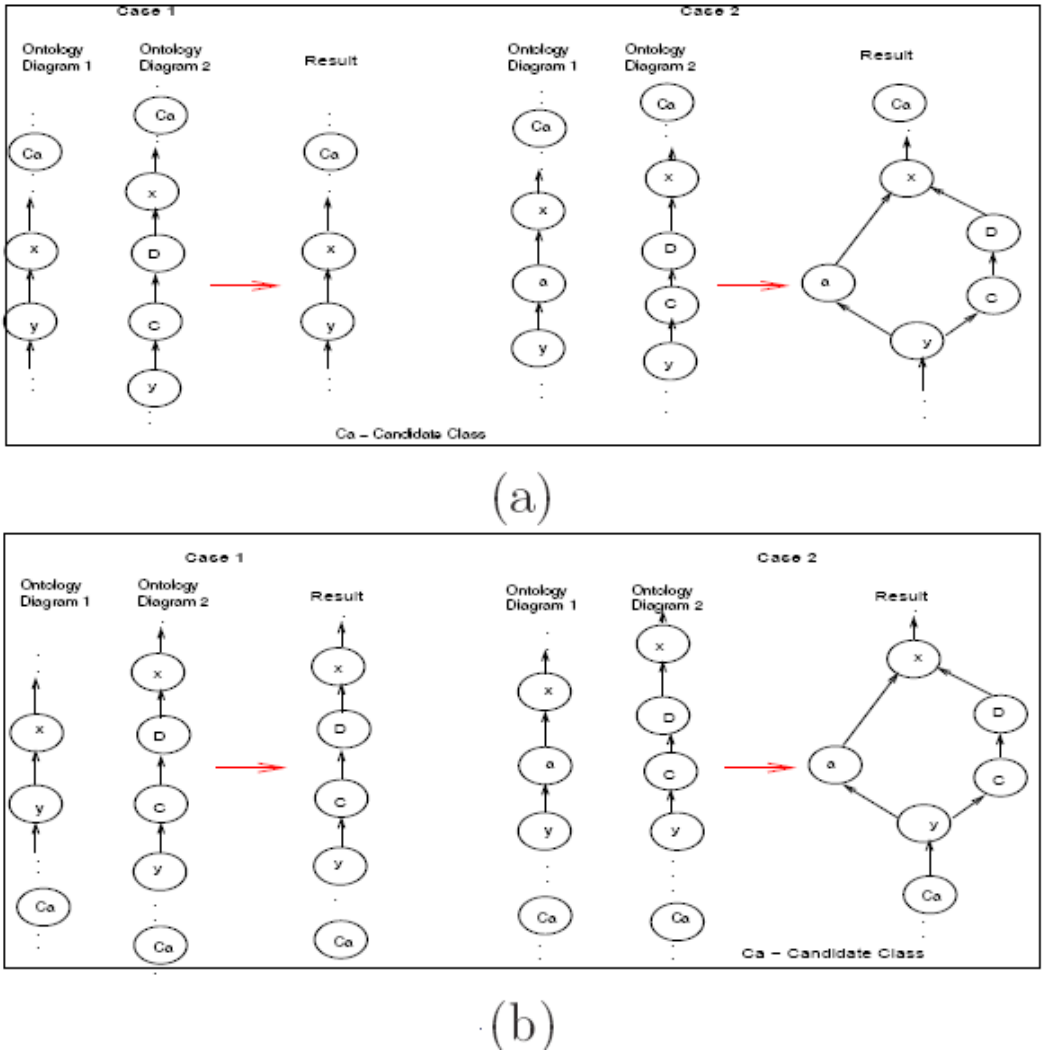


the K/N as low as possible. More generalization the structure represents more re-usable is the extracted ontology. Therefore, our algorithm aims to keep the K/N as low as possible.

Aggregating Ontology from Multiple Inheritance/ Aggregation/ Composition Relationship Graph

We use the edge counting graph (Rada, 1989) method to calculate the semantic distance between the nodes since it gives a good estimate of how close two nodes are. In edge counting method, semantic distance between two nodes is measured as the number of edges between two nodes. In order to maximize generalization nodes, a smaller path between any two nodes in the levels above the level of the candidate class in a ontology diagram is replaced by a longer path (if one exists) from another ontology diagram(s). Figure 11(a) gives various input and output diagram for generalization nodes in an inheritance. The Figure shows the

Figure 11. Ontology aggregation (a) specialization (b) generalization



replacement of the smaller paths by longer paths if present in the input diagrams. The Figure also shows that if there are no common nodes, all the paths are preserved. In order to keep specialization nodes as low as possible, a longer path between two nodes below the level of the candidate class in a ontology diagram is replaced by a shorter path (if one exists) from another ontology diagram(s). Figure 11(b) shows the replacement of longer path(s) between any two nodes by shorter paths if available in the input diagrams. If no common nodes are present all the nodes in the ontology diagrams are preserved in the resultant structure. The resultant diagram obtained by aggregating inheritance/aggregation/com position relationship is referred to as the Partial Aggregated Relationship Diagram (PARD).

Aggregating Ontology from Multiple 'USING' Relationship Graphs

For aggregating 'USING' relationship, all the elements that are related to the candidate class through 'USING' relationship from the ontology diagram(s) are added to PARD. The resultant diagram is the final Aggregated Ontology Diagram (AOD).

Inconsistency

In general, aggregating information from multiple ontology diagrams would give rise to two kinds of inconsistencies: Class Level Inconsistency: This inconsistency arises when a class (identified by name) appears in multiple ontology diagrams but the attributes of the class are different in different in ontology diagrams. For example Registered-Course in an ontology diagram has {StudentId, CourseId} as two attributes and has {StudentId, CourseId, BillAmt} in another ontology diagram. Our aggregation algorithm handles this inconsistency by merging the list of attributes (attribute richness) in the resultant AOD. Association Level Inconsistency: This inconsistency arises when association between two nodes in one ontology diagram is different than in another ontology diagram. If such association exists between two nodes we preserve all the associations between the two nodes. In ontology each association is represented as a property. Properties are first class elements in ontology and therefore can exist independent (of the class). In ontology end points of the property are defined by defining the domain (source) and range (destination) for the property. This implies there can be multiple properties whose end points and range are same, but their utilization depends on the context. By maintaining these associations (as properties) we are precisely extracting the domain (context) in which a entity exits. This is one of the important benefits of aggregating domain information from multiple ontology diagrams.

ONTEXTRACT: IMPLEMENTATION, EVALUATION AND ASSESSMENT

To put our approach into practice, we have implemented our approach as a plug-in using IBM Rational Software Architect (IBM Enterprise Architect Kit for SOA, 2006) using Java language. The plug-in reads in the class and outputs an ontology diagram that is represented using UML notation. Our tool provides provisions for software professionals to extract relevant ontology as applicable to a new problem by providing two optional input holders: (a) sequence diagram holder and (b) domain requirement holder. Both these inputs help extract refined or customized ontology. We have evaluated two cases: (a) Sample Domain - Student course registration, (b) Large scale real world example from IBM. We refer to the extracted ontology of SolA as Base Ontology [BaseOnt] (Figure 6). In general ontology evaluation approaches are classified into four categories (i) those that evaluate the ontology by comparing it to the golden standard, (ii) those that evaluate the ontologies by plugging them in

an application and measuring the quality of the results that the application returns, (iii) those that evaluate ontologies by comparing them to unstructured or informal data (e.g. text documents) and (iv) and those based on human interaction to measure ontology features not recognizable by machines. In our work we have used all the above mentioned work to prove two main aspects described as follows:

1. First we proceed to show that our extracted ontology by OntExtract is reasonable and correct.
2. The extracted ontology is reusable in new solutions (due to the fact that the created ontology overlaps with the design requirements of a new solution).

The first aspect evaluates the validity of the extracted ontology. We use the approach of comparing the extracted ontology with the golden standard to show that our algorithm extracts ontology similar to existing standard ontologies. In the second aspect, we use the approach of plugging the extracted ontology in an application and measuring the quality of results to evaluate the useful of the extracted domain ontology. Finally, we take the help of domain experts (humans) and bring out the importance of our tool in help them with defining better UML diagrams.

OntExtract Extracts Reasonable Ontology

The aim is to show that OntExtract extracts reasonably accurate ontology (serves the intended purpose of extracting ontology). By this we also show that our approach is correct.

Methodology

We evaluate our approach for correctness of is-a, has-a and other associations between entities relations by comparing the extracted ontology with manually built Golden standard (ontology). Golden standard ontology consists of set of entities and relationships that are considered as a good representation of the concepts of the problem domain under consideration. The golden ontology is often manually created by experts (Hassell, 2006) or obtained by merging multiple individual ontologies that are correct. We compare the extracted ontology to a golden ontology, considered as a good representation of the concepts of the problem domain under consideration. High values of matching entities between the two ontologies indicate that our approach OntExtract extracts similar ontology to existing ones.

In our case we consider domain 'University' (whose subset is the course registration) and obtain the golden standard ontology by aggregating several individual 'University' ontologies from (Jeff, 2000; University Ontology, 2002; Donnie Dorr, 2000). Let UML1, UML2 and UML3 be notations that represent these ontologies respectively. We first show that these individual ontologies are correct and therefore the aggregation of these ontologies results in golden standard ontologies. We use the evaluation method presented by Bruno et al (2005) to measure the correctness of individual ontologies. This method assigns grades (depending on the correctness) to individual ontologies and determines the corrections based on these grades. It uses the combined approaches proposed in (Gmez-Prez, 1994; Oltramari, 2002) considered as traditional evaluating methodologies. To begin with it assigns each ontology a grade starting at 20. For each error encountered the grade value is decreased.

Table 1 presents the various criteria s considered for assigning grades. Ontologies that have grades greater than 13 are considered to be satisfy correctness conditions.

Table 2 presents the consolidation of the grades obtained for individual ontology diagrams that we have considered to obtain the golden ontology. From Table 2 it can be seen that the UML diagrams that we have considered to obtain the

Table 1. Criteria considered for grade assignment

Criteria	Points
Completeness Errors (not satisfying the requirements)	-1
Wrong usage of is-a relationship	-1
Wrong usage of has-a relationship	-1
Wrong usage of association	-1
Modeling errors	-2
Clarity errors	-1
Consistency errors	-10

Table 2. Grades for UML diagrams considered for merging

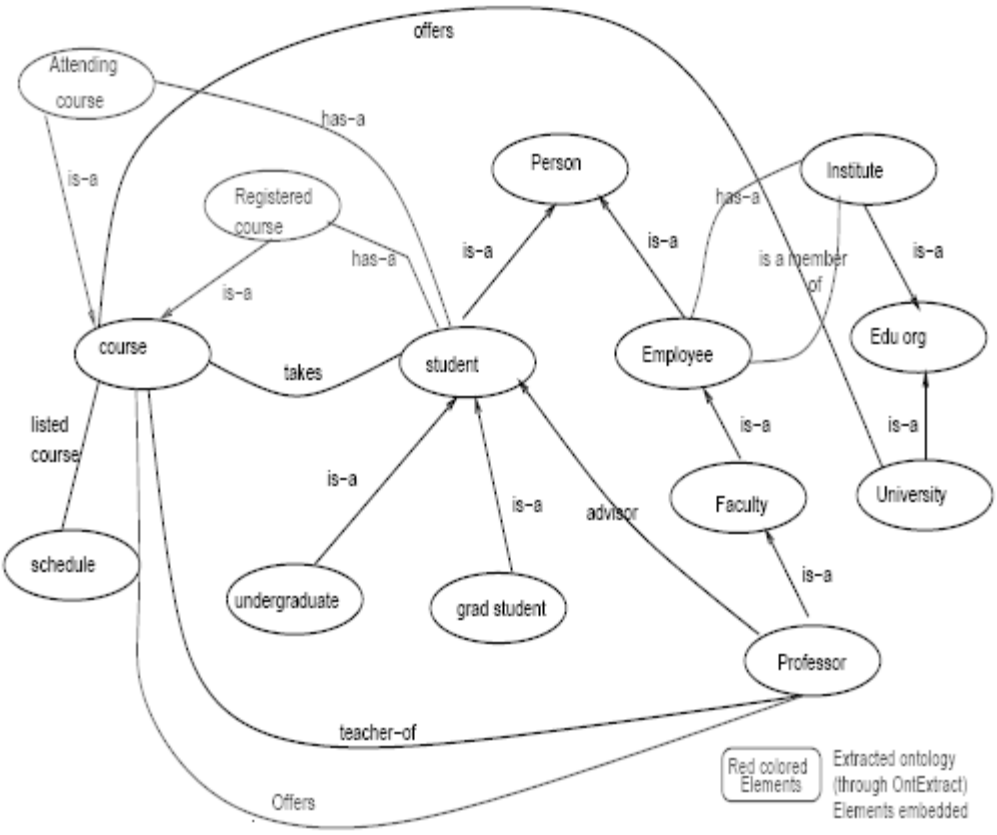
Criteria	UML 1	UML 2	UML 3
Completeness error	0	0	0
Wrong usage of is -a relationship	0	0	0
Wrong usage of has-a relationship	0	0	0
Modeling errors	-2	-2	-1
Clarity errors	-1	0	-1
Consistency errors	0	0	0
Total grade	**17**	**18**	**18**

golden ontology are correct (have grade points greater than 13). We aggregate these individual ontologies to obtain the golden standard. The reason why consider a golden ontology and not individual UML diagrams considered to aggregate is primarily because the individual ontologies considered might be not include all/most of the components of the domain. Complete or partially complete domain ontology can be obtained by merging multiple ontologies.

We present the rules considered for aggregating these correct UML diagrams to obtain a golden standard. The aggregation of these individual ontologies is done using the following rules: (a) union of all relationships is considered between two similar classes (b) for any association all the classes that it connects are considered. A golden ontology thus obtained is shown in Figure 12.

For purposes of comparison, we define Entity Level Relationship Diagram (ELRD), which is a high level abstracted diagram of the extracted ontology containing only the entities and relationship after removing the properties of class and associations. The following rules are used to obtain ELRD from either EECD or OD. (a) Properties of a class in a UML diagram (attributes) are specific to

Figure 12. Golden Ontology – University



a specification. For example, property 'studentid' may appear in some university diagram. In some other university it may be 'studentName'. Since our aim is show that the extracted information is ontology we omit the properties and consider only the entity and relationships. Similarly, attributes of associations are omitted. (b) Inheritance link is replaced by a directed link; Aggregation, composition and using are replaced by just a link. (c) Inheritance is replaced by text 'is-a', Aggregation/compositions is replaced by text 'has-a', using is replaced by the appropriate verb (d) 'has-a' and 'is-a' relationship is given high priority. If two nodes are connected by has-a or is-a, then all other relationship is ignored. (e) Any variable containment is treated (including pointer containment) as 'has-a' relationship. (f) CASE is ignored (g) Naming conflicts are manually resolved through annotations using external sources. (h) Terms are extended with meanings to increase the scope of matching Figure 13 presents the ELRD of the BaseOnt. Base-ELRD is the term used to refer to this ELRD. We compare Base-ELRD with the golden ontology of university (Figure 12). Table

Figure 13. ELRD for student course registration

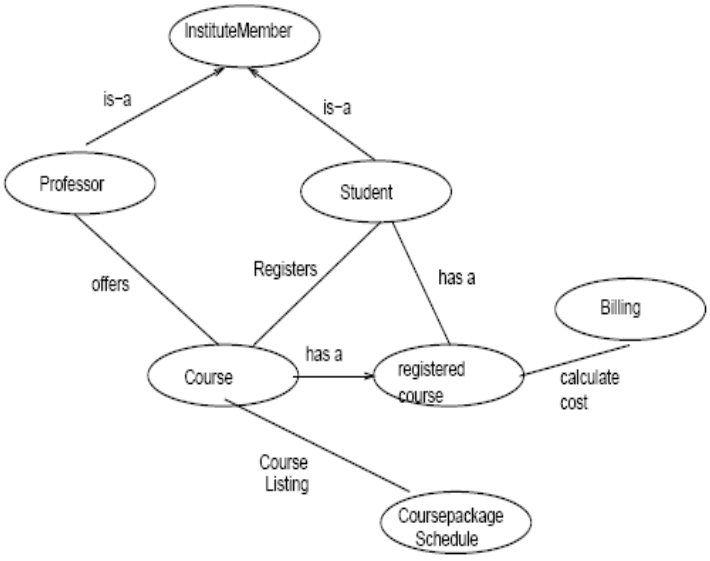


Table 3. Inference rules applicable for matching

Criteria	Name	Extensions to criteria
is-a, has-a		Matched as such
Other IS-A relation-ship(s)	Inference 1	A is-a B is-a C implies A is-a C
	Inference 2	In the above, Property of C is inherited by A
	Inference 3	A has-a B has-a C =) A has-a C
Using relationship	Offer	Matches offer of the standard ontology
	Registers	Since student has registered course and registered course has course, one can conclude that student has course

3 presents the inference criteria for determining matches between the golden ontology and the ELRD elements.

Results

Equation 1 gives the quantitative measure of entity level similarity (E_{sim}) observed as the ratio of the number of matching entities(s) between the golden ontology and Base-ELRD to the total number of entities present in the Base-ELRD of the extracted ontology:

$$E_{sim} = n^{E}_{match} / n^{E}_{ELRD} \quad (1)$$

where n^{E}_{match} is the matching entities between the golden ontology and Base-ELRD and n^{E}_{ELRD} give the number of total number of entities present

in the Base-ELRD diagram. In the similar way, Equation 2 gives the quantitative measure of relationship level similarity (R_{sim}):

$$R_{sim} = n^{R}_{match}/n^{R}_{ELRD} \quad (2)$$

where n^{R}_{match} is the number of matching relationship(s) between the golden ontology and the Base-ELRD and n^{R}_{ELRD} gives the number of total number of relationship elements present in the Base-ELRD diagram.

Table 4 gives the number of matching entity and relationship elements between the golden ontology and Base-ELRD (Figure 12 and Figure 13). The class Course is related to registered course through inference property of 'HAS-A' relationship. Using Table 4 and Equations 1 and 2, we calculate the matching (similarity) entities and relationships metrics as presented in Table 5.

The extracted ontology from the UML diagrams by preserving relationship and entities shows about 85.7% match with the golden standard and about 75% match with the relationships. High values of matching entities and relationships between the extracted ontology through OntExtract and golden ontology show that our approach extracts ontology similar to existing (golden) ontologies. In addition, our method detects other elements that do not have corresponding matches in the golden ontology thereby enriching the existing ontologies with additional elements. This directly proves that our OntExtract method is correct since otherwise there would have been large mismatch between the golden ontology and ontology extracted through OntExtract.

Reusability Evaluation

Design of software solutions is greatly enhanced if the software architect has/gets 'a-priori' knowledge about concepts involved in the problem specification (Marcus A. Rothenberger, 1999). The aim is to show that the extracted ontology is reusable in a number of situations like: (a) Reuse in new solution development (b) Reuse in new problem scenario.

Reuse In New Solution Development

Let us consider a solution for a problem domain to be developed by different group of architects. A solution provided by a group will greatly help the other architects to detect artifacts and provide better solution in the problem. We illustrate this by showing that there exists large amount of common concepts among the solutions, if the solution is developed by different group of architects.

Table 4. Matching found for golden ontology and extracted ontology

Matching entities	Matching Relationship
Student, Professor, Course (registered course, through inference property of 'HAS-A' relationship), schedule/courselisting, Institutemember. Total: 6	is-a (Professor, institutemember), is-a(Student, institutemember) , has-a (student,course ;sub-Category registered course), Offer (professor, course) term 'offer' is matched with 'teacher-of', has-a (course, registered course), has-a (course [subcategory: registeredcourse], courselisting). Total:6

Methodology: We take two different solutions SolB and SolC provided by different group of experts for the course registration problem specification. Comparisons are made to determine the number of common matching entities between these solutions and the BaseOnt. High value of this metric indicates that the ontology from one solution is highly re-usable for new solutions of the problem.

We measure entity reuse (E_{reuse}) as the ratio of number of matching entities with the BaseOnt (n^{BE}_{match}) to total number of entities present in the solution (n^{SE}_{sol}) and is given by Equation 3. In the similar way, Equation 4 gives the relationship reuse R_{reuse}:

$$E_{reuse} = n^{BE}_{sim} / n^{SE}_{Sol} \quad (3)$$

$$E_{reuse} = n^{BR}_{sim} / n^{SR}_{Sol} \quad (4)$$

where n^{BR}_{sim} is the number of matching relationship(s) with the BaseOnt and n^{SR}_{Sol} is the number of relationship(s) present in the solution. Table 3 was used to obtain the matching entities.

SolB with Base Ontology: The ERLD of the extracted ontology for Solution 1 is provided in Figure 14.

Figure 14. Sol B ELRD diagram

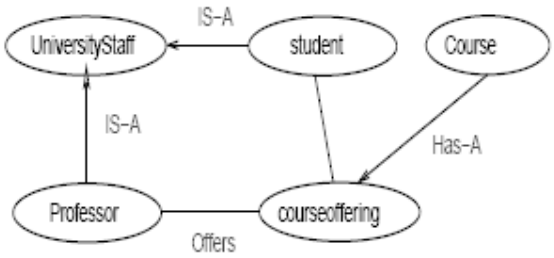


Table 6. Matching for SolB and BaseOnt

Matching Entities	Matching Relationship
Course, Student, courseoffering, Professor, Universitystaff	is-a (professor,university staff),is-a(professor, univeristystaff), has-a(course, courseoffering), Using (Courselisting, student)

Table 7. Comparison of Solution 1 with BaseOnt

Criteria	Number value
Number of entities present in solution 1 ELRD Number of entities matching with base ontology Number of relationship(s) present in the solution 1 ELRD Number of relationship(s) matching with base ontology	5 5 5 4
Re-use of entities present E_{reuse} Re-use of relationship present R_{reuse}	5/5 * 100 = 100% 4/5*100 = 80%

Table 6 presents the matching elements between the ELRD of SolB and Base Ontology. In some cases, inference rules were extended to find similar relationships. For example has-a(course, courseoffering) in Solution 1 is similar to has-a(course, courseoffering) in the BaseOnt through 'courseSchedule'. Classes universityStaff in Solution 1 and instituteMember in BaseOnt represent the same entity and is resolved manually.

Equation 3 and 4 are used to measure re-usability as presented in Table 7. Reuse of entities and relationship(s) has been observed as high as 100% and 80% respectively.

SolC with Base Ontology: Figure 15 gives the ELRD diagram for Solution 2. Table 8 gives the matching elements between ELRD of Sol2 and BaseOnt. Class person match is obtained using extended annotations using external source. Reusability is calculated as shown in Table 9.

Our results indicate that there is indeed great amount of reuse ranging from 70% -100% entity level and 57% to 80% relationship level reusable

Figure 15. Sol C ELRD diagram

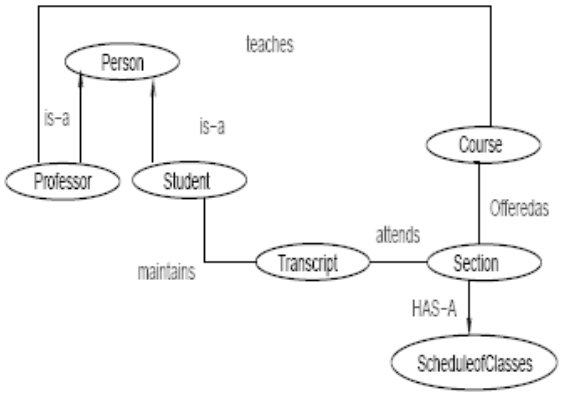


Table 8. Matching for Solution C and BaseOnt

Matching entities	Matching relationships
student, professor, person, course, scheduleclass	is-a(student, person), is-a(professor, person), teaches(professor,course), has-a(course, schedulecourse)

Table 9. Comparison of Solution 2 with BaseOnt

Criteria	Number value
Number of entities present in solution 2 ELRD Number of entities matching with base ontology Number of relationship(s) present in the solution ELRD Number of relationship(s) matching with base ontology	7 5 7 4
Re-use of entities present E_{reuse} Re-use of relationship present R_{reuse}	5/7 *100 = 71.4% 4/7*100 = 57.14%

components. This shows OntExtract provides provisions for reusing previously available ontology to new solutions.

Reuse in New Problem Scenario

Our approach provides provisions for the software professionals to model the use cases of the new problem as sequence diagram and extract related ontology.

Methodology: The architect provides his solution as design diagrams. We extract the domain concepts through OntExtract and call the ontology as ArchitectOnt. We model the use-case requirement as a sequence diagram and use this as a filter to extract ontology from the BaseOnt. We call this ontology as NewScenarioOnt. We find the number of elements of NewScenarioOnt is present in the ArchitectOnt. High values of this metric indicate that large number of domain entities can be reused.

We take an example where the software architects want to design requirements to implement 'ClearDues' use-case of the administration department. The 'ClearDues' checks if the student is valid and have cleared all the courses with clear grades. The architect intends to design new classes and relationship with existing course registration system. For the sake of brevity, only the ELRD diagram (Figure 16) of the class diagram developed by the architect (ArchitectOnt) is provided.

Figure 16. Class diagram by architect

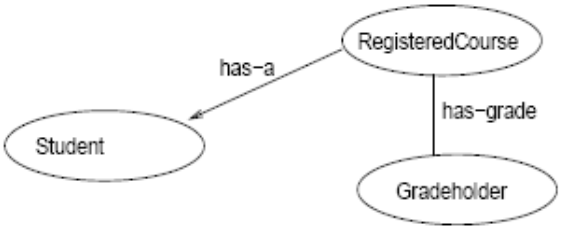


Figure 17. Relevant ontology extracted

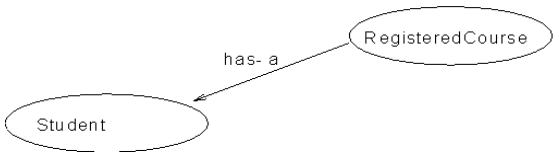
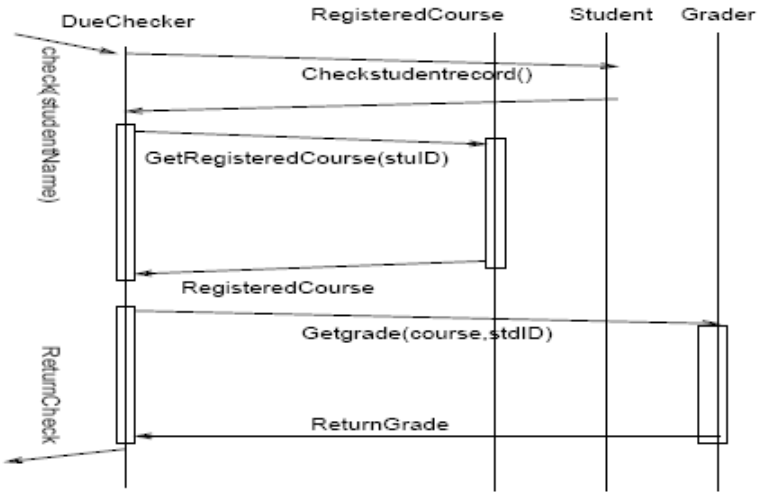


Table 10. Reuse in similar but different domain

Criteria	Number value
Number of matching entities between the NewScenarioOnt and ArchitectOnt Number of matching relationship(s) between the NewScenarioOnt and ArchitectOnt Total number of entities present in the NewScenarioOnt Total number of relationship present in the NewscenarioOnt	2 1 2 1
Reuse of entities Ereuse Reuse of relationship Ereuse	2/2*100 = 100% 1/1*100 = 100%

Figure 18. Sequence diagram for Use case "ClearDues"



We use the sequence diagram pertaining to 'ClearDues' (Figure 18) as a filter to extract ontology from Figure 6. This ontology extracted is henceforth referred to as the NewScenarioOnt. All the classes in the sequence diagram are considered to extract NewScenarioOnt. The NewScenarioOnt extracted in ELRD form is given in Figure 17.

The results presented in Table 10, indicates that the sequence diagram of a new functionality helps identifying reusable components from previously available solutions. Though this situation is ideal and has 100% reusable components, in practice the amount of re-usability might be less than the one presented here. However, the domain ontology is intended for reuse of domain concepts wherever required; one might expect high values of reusability.

Evaluation on Real World Example

We have taken a real world scenario called "Secure Trade Lanes" (Schaefer, 2006). The description is public but the design diagrams are confidential. It was made available from a business unit in IBM to evaluate our approach on a large scale problem. There were 25 component level diagrams, interaction diagrams (sequence diagram) for use cases. Practical limitations were observed in applying our approach to the real world scenario. The classes were not stereotyped. We had stereotyped the classes manually, since our tool relies on the stereotypes for identifying class level implementation artifacts. The sequence diagrams were at the component level. With the help of architects we had interpreted class level sequence diagrams before using our tool.

The work on real world problem was presented to a group of architects and developers to demonstrate the utility of our plug-in. Their feedbacks indicate that our tool will be useful to develop better software solution architecture.

Finally as a part of evaluation we present the practical usability of our tool. Table 11 provides the benefits of work for different category of

Table 11. Advantages and relevance for practice of OntExtract

Users	Advantages	Impact on software development process
Architects	• Provides Domain View from previous knowledge base (rather than implementation) • Provides existing solution approaches for the new problem and helps extract customized ontology	• Obtaining complete domain prior to development is complicated. Our plug-in helps to provide domain view. • Better architectures by providing view of constraints, entities and relationships that would have been otherwise omitted.
Developer	Helps justify implementation decisions and model constraints from existing solutions	Re-usability
Reasoners	• Uniform understanding of the domain • Aggregation of ontology diagrams provides reuse through inference	• Aggregated Domain view from multiple design diagrams • Fosters reuse of components across different but related problems

people involved in design and development of software solutions.

CONTRIBUTIONS, LIMITATIONS, RELATED AND FUTURE WORK

We have introduced a novel automatic approach for extracting domain ontology for software asset reuse from multiple design diagrams like UML. We observe that dynamic behavior diagrams act as powerful filters in extracting customizable ontology for new scenario. Our OntExtract extends to aggregate ontology from multiple design diagrams. We have developed an ontology extraction plugin over IBM Rational. We have evaluated our work with a sample domain and a real world example. Our approach shows promising results to provide domain view and reuse of the domain theories to develop new scenarios.

Our main contribution in this article can be summarized as follows:

1. An approach for extracting ontology from UML class diagrams.
   - An approach to refine the extracted ontology using sequence diagram.
   - To obtain ontology relevant to a domain by specifying domain requirements as a set of key words.
   - To generate ontology customizable to a new problem domain.
2. An approach to aggregate the extracted ontology using multiple class diagrams of either same or different solutions.

Limitations

The following are the limitations of our work:

1. In our work we have shown extraction of ontology as relevant to a single domain requirement. An incremental approach of extracting ontology with respect to individual domain requirements followed by aggregation of these models will not satisfy the requirements of obtaining single unified ontology with respect to both domains. This is primarily due to fact that individual domain ontology can vary a lot. There will be missing associations between those entities that were extracted with respect to a domain requirement and those entities that were extracted with respect to other domain requirements. A different approach is required to capture more than one domain requirement and extract ontology with respect to both of them.
2. Although we have used analysis class diagram in our work, we are currently working on methodologies that would help identification of stereotypes for classes. For example, we are exploring methods such as code analysis, pattern analysis to categorize classes. In the code analysis methodology, we map the classes in UML diagram to specific stereotype based on the implementation. For example, entity beans can be categorized as entity class, session beans as controller class and servlets and JSPs are boundary class. In pattern based analysis, we analyze the interaction pattern among the class to arrive at a decision of the category of the classes. Such methodologies would enable and promote usage of more commonly available implementation class diagrams as sources of ontology.

Related Work

Significant work has been done in the area of ontology learning and extraction (Alexander, 2001). Various approaches have been used to extract and learn ontology: (a) pattern based approach (Morin, 1999) where a relation is recognized when a sequence of words in the text matches a pattern, (b) association rule approach (Adriaans ,1996) where rules are used to match the query, typically used for data mining process, (c) conceptual pruning approach (Faure D, 2000) where the concepts are grouped according to semantic distance, (d) ontology pruning approach (Kietz, 2000), where the objective is to build a domain ontology based on different heterogeneous sources, (e) concept learning approach (Hahn, 2000), where the ontology is incrementally updated with new concepts. OntExtract is precisely an extraction technique that combines rule based approach along with conceptual pruning. Rules define the UML entities to be removed and preserved, while concept pruning groups entities based on semantic distance.

While the above-mentioned work concentrates on ontology sources and techniques to obtain ontological information, there has been significant amount of work done in using UML as an ontology language (Baclawski, 2002). Some work, such as presented in (Cranefield, 2001), aim at converting an UML specification to an Ontology language specification such as OWL (Deborah 2000). Our work does not concentrate on converting or mapping the UML notation (specification) to ontology language specification but concentrate on identifying and preserving elements that represents domain knowledge (ontology).

Close to our work along the lines of using UML diagrams in the field of domain engineering is the work presented in (Cranefield, 2003). The aim of their work is to express the ontology using UML and to automatically generate related ontology

specific content language along with corresponding java classes and an RDF based serialization mechanism. The work in (Maurizio, 2000) aims at providing a domain engineering approach and the supporting tools to define software product lines. The approach concentrates on domain engineering (identifying domain, domain analysis, and reusable domain components) and expressing these entities using UML.

Future Work

The work presented in this article brings forward various promising areas as follow-up work which we have set as for exploring in the future. We present some of them below:

1. Exploring extraction of ontology by considering other UML diagrams such as object diagrams and use case diagrams has been left for future work. For example, object diagrams provide information about individuals which are one of the constituents of ontology. Time dependent activities can be captured as ontology from dynamic UML diagrams such as activity and use case diagrams. State diagrams are excellent sources of events that can be captured as ontology.
2. Our experience from the real world example that classes are not stereotyped point out that learning stereotype can also be a promising direction. Learning stereotypes are important not only in the context of ontology learning but also in the context of developing readable models.

Summary

Ontology extraction methods are of great importance to the Semantic Web, because it establishes (semi-) automatic methods of constructing fast and confident ontologies. We have presented a novel automatic ontology extraction methodology (OntExtract) that exploits commonalities in the domain software solutions to build ontologies which would not only help ontology engineers but greatly fosters software reuse. Our work uses design diagram as ontological sources to extract and explicitly represent these common domain concepts.

OntExtract tool provides software professionals a-priori information about domain theories thereby helping them to build faster and improved software solutions. With OntExtract, software professionals can extract domain ontology and ontologies customized to new scenarios. The prime target application of our ontology extraction being software reuse, it by itself serves as a measure for evaluating the resultant ontology. Evaluation results indicate that our method greatly promotes software reuse.

REFERENCES

Adriaans P & Zantinge D (1996), *Data Mining,* Addisson-Wesley.

Bettina Berndt (2005), UML II: Class diagrams, from UML to Java, other UML diagrams, Lecture Series Information systems, http://warhol.wiwi.huberlin.de/berendt/MEMS/information_systems_presentation_oct05.PPT

Bruno,G & Sofia,P. H (2005), *Experiences in Evaluation and Selection of Ontologies,* KCAP 05 workshop on Ontology Management, Oct. 2005 , pp. 25-32.

Baclawski, K. et al (2002), *Extending the Unified Modeling Language for ontology development,* Berlin, NewYork, Journal Software and Systems Modeling (SoSyM), Vol.1, No.2, pp. 142-156.

Cranefield, S (2001), *UML and the Semantic Web,* Palo Alto, Proceedings of the International Semantic Web Working Symposium (SWWS).

Cranefield,S. , Jin,P. & Martin., P. (2003), *A UML ontology and derived content language for a travel*

- *Booking Scenario*, World Wide Web Consortium Web page, http://www.w3.org/RDF/

Donnie,D. (2000), *CS Dept Ontology in SHOE Ontologies in DAML Format*, http://www.cs.umd.edu/projects/plus/DAML/onts/cs1.0.daml

Faure D & Poibeau T (2000), *First experiments of using semantic knowledge learned by ASIUM for information extraction task using INTEX*, Berlin, Germany, Proceedings of the Workshop on Ontology Learning, 14th European Conference on Artificial Intelligence.

Gmez-Prez, A. (1994), *Some Ideas and Examples to Evaluate Ontologies*, tech. report KSL-94-65, Knowledge System Laboratory, Stanford Univ., 1994.

Hassell, J., Aleman-Meza, B., & Arpinar**,** I.B. (2006), *Ontology-Driven Automatic Entity Disambiguation in Unstructured Text*, th International Semantic Web Conference (ISWC 2006), Athens, GA, November 5–9, 2006, I, Lecture Notes in Computer Science, vol. 4273, Springer, 2006.

Hahn U & Schulz S (2000), *Towards Very Large Terminological Knowledge Bases: A Case Study from Medicine*. In Canadian Conference on AI 2000: 176-186.

IBM Enterprise Architect Kit for SOA (2006), http://www.ibm.com/developerworks/architecture/kits/archkit2/index.html?S_TACT=105AGX23&S_CMP=AKBDD

Jeff Heflin (2000), University Ontology, http://www.cs.umd.edu/projects/plus/SHOE/onts/univ1.0.html

Kietz JU, Maedche A & Volz R (2000), *A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet*, Juan-Les-Pins, Franc EKAW'00 Workshop on Ontologies and Texts.

Marcus A. Rothenberger, *System Development with Systematic Software Reuse: an Empirical Analysis of Project Success Factors*, Technical report, http://wi99.iwi.uni-sb.de/de/Doktoranden-Seminar_PDF/D_Rothenberger.pdf

Maurizio Morisio, Guilherme H. Travassos & Michael E. Stark (2000), "Extending UML to Support Domain Analysis", Proceedings of the 15th IEEE international conference on Automated software engineering, Page: 321.

Morin E (1999), *Automatic acquisition of semantic relations between terms from technical corpora*, TermNet-Verlag, Vienna, Proc. Of the Fifth Int. Congress on Terminology and Knowledge Engineering (TKE-99).

Oltramari, A. , Gangemi, A. , Guarino. N, & Masol.C (2002), *Restructuring WordNet's Top-Level: The OntoClean approach*, presented at LREC 2002.

Deborah L,M. , Frank,V. H., (2004) , OWL- Web Ontology Language (2004) , http://www.w3.org/TR/owl-features/

Rada.R, Hafedh Mili, Ellen Bicknell, & Maria Blettner (1989). *Development and Application of a Metric on Semantic Nets*, IEEE Transactions on Systems, Man and Cybernetics, 19:17-30.

Rational Software Corporation (2000), Rational Tutorial, www.ibm.com/developerworks

Schaefer,S., (2006), *Secure Trade Lane: A Sensor Network Solution for More Predictable and More Secure Container Shipments*, Portland, Oregon, USA, Dynamic Languages Symposium, Pages: 839 - 845.

Terry Quatrani (2001), *Introduction to the Unified Modeling Language*, Rational Developer Network, http://rational.net//

Terry Quatrani (2002) Visual Modeling with Rational Rose 2002 and UML, (pp 1-100), The Addison-Wesley Object Technology Series.

ADDITIONAL READING

Babenko. L. P. (2003), *Information Support of Reuse in UML-Based Software Engineering, Cybernetics and Systems Analysis,* Hingham, MA, USA, Kluwer Academic Publishers, Volume 39, Issue 1, PP: 65 - 70.

Benslimane, S. M, Malki .M., & Lehirech, A., (2006), *Towards ontology-based semantic web from data-intensive web: A reverse engineering approach,* International Conference on Computer Systems and Applications, pp. 771-778

Carlos,P., Amaia, B., Tim, S., Jessica, A.& Manuel, C., (2004), *A Framework for Ontology Reuse and Persistence Integrating UML and Sesame,* Springer Berlin / Heidelberg, Lecture Notes in Computer Science, Volume 3040/2004, pp: 37-46.

Daniel O, Andreas, E., Steffen,S & Raphael Volz (2004), *Developing and Managing Software Components In An Ontology-Based Application Server,* Toronto, Ontario, Canada, In 5th International Middleware Conference, volume 3231 of LNCS, pp. 459-478. Springer.

Falkovych, K. , Sabou, M. & Stuckenschmidt. H (2003), *UML for the Semantic Web: Transformation-Based Approaches,* Amsterdam, OS Press, Vol. 95 (2003) 92-106

Guizzardi, G., Herre, H. & Wagner G. (2002), *On the General Ontological Foundations of Conceptual Modeling.* Berlin, 21 Intl. Conf. on Conceptual Modeling (ER 2002). Springer-Verlag, Berlin, Lecture Notes in Computer Science, pp : 65—78

Guizzardi, G. & Wagner G. (2002), *Using Formal Ontologies to define Real-World Semantics for UML Conceptual Models.* In 1 Workshop on Application of Ontologies to Biology, European Media Laboratory, Heidelberg, Germany.

Happel, H.-J., Korthaus, A., Seedorf, S., & Tomczyk, P.(2006), *KOntR: An Ontology-Enabled Approach to Software Reuse,* San Francisco, In: Proc. of the 18th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE).

Hyoil, H. & Ramez,E., (2003), *Ontology extraction and conceptual modeling for web information***,** Hershey, PA, USA, Information modeling for internet applications, Pages: 174 - 188 .

Stephen,C., & Martin P., (1999), *UML as an Ontology Modelling Language***,** Sweden, Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence, volume 23, pp: 46-53

Szyperski. C (2002), *Component Software - Beyond Object- Oriented Programming,* London, 2nd edition, Addison-Wesley.

Odell, J. & Bock, C. (1998), *A More Complete Model of Relations and their Implications: Roles.* Journal of OO Programming, May, 51-54.

Uschold, M., Clark, P., Healy, M., Williamson, K., & Woods, S. (1998), *Ontology Reuse and Application.* Italy, Formal Ontology in Information Systems. IOS Press PP:179-192.

Welty, C.A. & Ferrucci, D.A.(1999) ,A formal ontology for re-use of software architecture documents, Cocoa Beach, FL, USA, Automated Software Engineering, PP: 259-262.

Yang, J. & Chung, I. (2006), *Automatic Generation of Service Ontology from UML Diagrams for Semantic Web Services* , Springer Berlin / Heidelberg, Lecture Notes in Computer Science, Volume 4185/2006, pp: 523-529.

This work was previously published in The Semantic Web for Knowledge and Data Management: Technologies and Practices, edited by Z. Ma & H. Wang, pp. 288-315, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 3.10
Evolution in Model-Driven Software Product-Line Architectures

Gan Deng
Vanderbilt University, USA

Jeff Gray
University of Alabama at Birmingham, USA

Douglas C. Schmidt
Vanderbilt University, USA

Yuehua Lin
University of Alabama at Birmingham, USA

Aniruddha Gokhale
Vanderbilt University, USA

Gunther Lenz
Microsoft, USA

ABSTRACT

This chapter describes our approach to model-driven engineering (MDE)-based product line architectures (PLAs) and presents a solution to address the domain evolution problem. We use a case study of a representative software-intensive system from the distributed real-time embedded (DRE) systems domain to describe key challenges when facing domain evolution and how we can evolve PLAs systematically and minimize human intervention. The approach uses a mature metamodeling tool to define a modeling language in the representative DRE domain, and applies a model transformation tool to specify model-to-model transformation rules that precisely define

metamodel and domain model changes. Our approach automates many tedious, time consuming, and error-prone tasks of model-to-model transformation, thus significantly reducing the complexity of PLA evolution.

INTRODUCTION

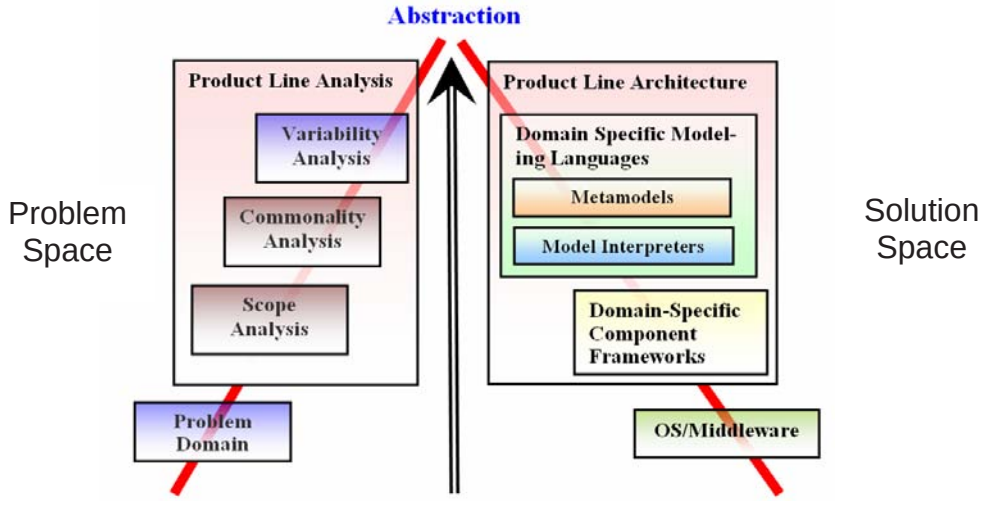
Software *product-line architectures* (PLAs) are a promising technology for industrializing software-intensive systems by focusing on the automated assembly and customization of domain-specific components, rather than (re)programming systems manually (Clements & Northrop, 2001). A PLA is a family of software-intensive product variants developed for a specific domain that share a set of common features. Conventional PLAs consist of *component frameworks* (Szyperski, 2002) as core assets, whose design captures recurring structures, connectors, and control flow in an application domain, along with the points of variation explicitly allowed among these entities.

PLAs are typically designed using *scope/commonality/variability* (SCV) analysis (Coplien, Hoffman, & Weiss, 1998), which captures key characteristics of software product-lines, including: (1) *scope*, which defines the domains and context of the PLA, (2) *commonalities*, which name the attributes that recur across all members of the product family, and (3) *variabilities*, which contain the attributes unique to the different members of the product family.

Motivating the Need for Model-Driven Software Product-Line Architectures

Despite improvements in third-generation programming languages (such as C++, Java and C#) and runtime platforms (such as CORBA, J2EE and Web Services middleware), the levels of abstraction at which PLAs are developed today remains low-level relative to the concepts and concerns within the application domains themselves, such as manually tracking the library dependency or

Figure 1. Using DSMLs and domain-specific component frameworks to enhance abstraction and narrow the gap between problem and solution space of software-intensive systems



ensuring component composition syntactical and semantic correctness. A promising means to address this problem involves developing PLAs using *model-driven engineering* (MDE) (Schmidt, 2006), which involves systematic use of models as key design and implementation artifacts throughout the software lifecycle. MDE represents a design approach that enables description of the essential characteristics of a problem in a manner that is decoupled from the details of a specific solution space (e.g., dependence on specific OS, middleware or programming language).

As shown in Figure 1, MDE-based PLAs help raise the level of abstraction and narrow the gap between the problem space and the solution space of software-intensive systems by applying the following techniques:

- **Domain-specific modeling languages:** A *DSML* (Gray et al., 2007) consists of *metamodels* and *model interpreters*. A metamodel is similar to the grammar corresponding to a programming language that defines a semantic type system that precisely reflects the subject of modeling and exposes important constraints associated with specific application domains. Model interpreters can read and traverse the models, analyze them, and help create the executable system based on these models. DSMLs help automate repetitive tasks (Gray, Lin, & Zhang, 2006) that must be accomplished for each product instance, including generating code to glue components or synthesizing deployment and configuration artifacts for middleware platforms and the underlying operating systems (Balasubramanian et al., 2006).
- **Domain-specific component frameworks:** Through SCV analysis, object-oriented extensibility capabilities are often used to create domain-specific component frameworks, which factor out common usage patterns in a domain into reusable platforms (Clements & Northrop, 2001). These platforms, in turn, help reduce the complexity of designing DSMLs by simplifying the code generated by their associated model interpreters and addressing the product-line specific functional and systemic concerns, including quality of service (QoS) concerns, such as latencies, throughput, reliability, security, and transactional guarantees. Throughout the rest of the chapter we use the two terms "domain-specific component frameworks" and "component frameworks" interchangeably.

MDE helps software developers explore various design alternatives that represent possible configurations for a specific instance of the product family. For example, a product instance ultimately needs to be deployed into a specific target running environment, where all software components must be deployed and mapped to available hardware devices and configured properly based on the specific software/hardware capabilities of the devices. If the PLAs are intended for use with different hardware devices, however, the mappings between the software components and hardware devices cannot be known *a priori* when the software PLAs are developed. Instead of analyzing every product instance individually and manually writing source code or scripts repetitively for every different target execution environment, an MDE-based approach to PLA deployment and configuration automates such repetitive and labor-intensive tasks by integrating domain knowledge and expertise into metamodels and model interpreters. Hence, a DSML infuses intelligence into domain models, which helps address many "what if" problems, such as "what glue code or configuration scripts must be written if the product is to be deployed into an environment with XYZ requirements?" These "what if" scenarios help developers understand the ramifications of

design choices of software-intensive systems at a higher level of abstraction than changing source code manually at the implementation level.

Challenges with Evolution of Model-Driven Software Product-Line Architectures

Although an MDE-based approach helps improve productivity of software-intensive systems by raising the level of abstraction through composition of DSMLs and domain-specific component frameworks, it is hard to evolve software PLAs by incorporating new requirements. Examples of such requirements include using new software platforms or applying the current PLA in a new use case that may impose a different set of concerns than those handled by the current PLA. Consequently, in addition to assisting in the exploration of design alternatives among product instances, an MDE-based PLA technology must also address the *domain evolution problem* (Macala, Stuckey, & Gross, 1996), which arises when existing PLAs must be extended or refactored to handle unanticipated requirements.

Depending on the scopes of the DSMLs and domain-specific component frameworks, unanticipated requirements can be either functional requirements or nonfunctional requirements, or both. For example, consider an MDE-based PLA that is available on two different component middleware technologies, such as Enterprise Java Beans (EJB) (Sun Microsystems, 2001) and CORBA Component Model (CCM) (OMG, 2006). A goal of a DSML that supports PLA is to selectively use the technologies within a product instance based on the system requirements. The metamodel of the DSML must define proper syntax and semantics to represent both component middleware technologies. With domain evolution, if the CCM technology must be replaced by another emerging middleware technology such as Web Services, the MDE-based PLA must evolve accordingly to satisfy the new requirements, that is, new syntax and semantics must be introduced into the metamodel and new domain-specific component frameworks must be developed based on the emerging Web Services technology.

Unfortunately, adding new requirements to MDE-based PLAs often causes invasive modifications to the PLAs in the DSMLs and component frameworks if DSMLs and component frameworks were not initially designed to be extensible to incorporate such new requirements. Conventional MDE tools do not handle the domain evolution problem effectively because they require significant handcrafted changes to existing PLAs, at both the component framework level and the DSML level. The domain evolution problem is particularly hard because the coupling of architecture and infrastructure concerns often *crosscut* the component framework layer and the DSML layer (Deng, Lenz, & Schmidt, 2005) within a PLA.

Moreover, changes made on metamodels in a PLA often invalidate existing domain models based on previous versions of the metamodels (Sprinkle & Karsai, 2004), which makes the evolution process of model-driven software PLAs hard. Other examples of this problem occur in programming language or object-oriented framework design, where changes to a grammar or class hierarchy for a programming language or framework may introduce errors in existing legacy source code (Klusener, Laemmel, & Verhoef, 2005). Another example is schema evolution in a database, where changes to a database schema may render the contents of the database useless (Roddick, 1992). Just like legacy source code and contents of database, domain models are crucial assets of an organization, so they must be handled well during the metamodel evolution process.

From these observations, there are many complexities involved when MDE-based software PLAs need to evolve. Although software developers can manually update their metamodels, domain models, and component frameworks for small-scale systems, this approach is clearly tedious,

time consuming, error-prone, and nonscalable for software-intensive systems.

Solution → Systematic PLA Evolution with Automated Domain Model Transformation

To address these challenges, a layered and compositional architecture is needed to modularize system concerns and reduce the effort associated with domain evolution. With the help of this architecture, different layers of PLAs can evolve systematically and tool supported domain model evolution also becomes feasible. The overall approach can be characterized in the following ordered steps:

1. The first step deals with component framework evolution. Because component frameworks provide core functionalities to the product instances, they have the most direct impact on the PLAs. As a result, whenever PLAs need to incorporate new requirements, component frameworks must first be refactored. To reduce the impact of such evolution outside the component frameworks, the key point is using pattern-oriented software architecture (Gamma, Helm, Johnson, & Vlissides, 1995; Schmidt, 2000).
2. The second step deals with metamodel evolution. Because metamodels are used to define type systems of particular domains based on proper language syntax, a language can be decomposed into smaller units to localize the evolution impact, and allow such smaller units to be composed to form the new metamodel.
3. The third step deals with the domain model transformation. This step applies automated model transformation techniques to specify model-to-model transformation rules that define metamodel changes. The application of automated model transformation alleviates many tedious, time consuming, and error-prone tasks of model-to-model transformation to reduce the complexity of PLA evolution. In particular, when an existing DSML in a PLA is changed, the domain models defined by this DSML can be migrated automatically to the new DSML by applying a set of model transformation rules.

While the three-step approach above could be applied to any model-driven software PLAs, this chapter focuses on distributed real-time embedded (DRE) PLAs, which are among the most difficult software-intensive systems to develop because such systems have limited resources and must communicate via the network to meet stringent real-time quality-of-service (QoS) assurance and other performance requirements. A representative software-intensive DRE system is used throughout the chapter as a case study to describe how to evolve PLAs systematically and minimize human intervention. Along with presenting the approach for domain evolution of MDE-based PLAs, the chapter also describes key concepts, such as model-driven engineering, product-line architectures, and model transformations that are important for developing and evolving PLAs for large-scale software-intensive systems.

The remainder of this chapter is organized as follows: We first evaluate related work that supports evolution of software PLAs for DRE systems and compare it with our approach. Then, we describe a conceptual architecture of MDE-based PLAs for DRE systems and define the key elements in this architecture as background of this chapter. After that, we introduce a representative case study of a PLA for avionics mission computing used throughout the chapter. Then, we describe the challenges involved when facing evolving model-driven PLAs and present the solutions to address these challenges. Lastly, we present our concluding remarks and lessons learned.

RELATED WORK

This section surveys the technologies that provide solutions to MDE-based software PLA evolution for software-intensive systems. The related work has been categorized along two dimensions based on the syntax of the modeling mechanism the software PLA evolution relies on, that is, a graphical based modeling approach or a text-based modeling approach.

Graphical Modeling Approaches

A UML metamodel for software PLA evolution (Mens & D'Hondt, 2000) has been developed based on the concept of an *evolution contract*. The idea of an evolution contract is that when incremental modifications and evolution of software artifacts are made on a software product line, a formal contract must be defined between the provider and the modifier. The purpose of the contract is to define the evolution behavior formally. A UML metamodel has been defined to capture the formal evolution contract. This offers a generic MDE-based mechanism for dealing with unanticipated evolution. By documenting model evolution through formal models, incompatibilities or undesired behavior across different modeling artifacts can be detected when models are upgraded, or when different software developers independently make changes to the same or related parts of a model. This approach allows conflicts to be detected regardless of the specific kind of model that is under consideration. The approach has been integrated into third-party CASE tools, such as IBM Rational Rose (IBM, 2007).

KobrA (Atkinson et al., 2002) is another approach based on UML for component-based software PLAs that support model-driven representation of components. In this method, evolution management in software PLAs is divided into three activities, that is, configuration management, change management, and maintenance planning. Configuration management in KobrA is a static method for bringing together different artifacts within a PLA. Change management consists of techniques to evaluate evolution requests. The use of appropriate formal change operators to evolution requests assists in traceability within change propagations in a PLA. Maintenance planning is responsible for constructing infrastructure for the change and configuration management activities. The idea of KobrA is based on a change-oriented model, that is, new versions are obtained from changes applied to some artifacts in the product line. To support the evolution in KobrA, the *evolution graph* technique (Atkinson et al., 2002) is proposed to capture version histories of different artifacts of the PLA and trace the dependencies.

Another technique similar to the evolution graph is called *design decision tree* (DDT) (Ran & Kuusela, 1996), which is a formal approach to incrementally document, refine, organize and reuse the architectural knowledge for software design. The formalism is a hierarchical organization of design patterns that is a partial ordering of design decisions put in the context of the problem requirements and the constraints imposed by earlier decisions. This model integrates architectural knowledge of software design into a software development process. A DDT contains system-wide design information in a form that can be used to analyze change requests and determine their impact on system structure. Because the tree is maintained throughout the lifecycle of a PLA, it can be used as the main repository of design knowledge (Karhinen & Kuusela, 1998). Such a repository can be used to analyze the impact of new requirements to the existing requirement space and to investigate the changes that different implementation strategies may cause to the system structure, which makes it possible to classify different options and to react to them and analyze their architectural implications.

Summary

The related work described above adopts a domain-independent modeling technique to capture the software PLA evolution requirements either explicitly or implicitly. Our approach is similar to these related works in the sense that they all provide visualization capabilities through graphical modeling tools for PLAs. Our approach, however, uses a domain-specific modeling technique that adds additional abstractions representing domain concepts to the modeling languages that are not available in general-purpose domain-independent modeling languages such as UML. DSMLs thus require less effort and fewer low-level details to specify a given system (Tolvanen & Kelly, 2005).

Text-Based Modeling Approaches

Architectural Description Language (ADL) is an important technique in this dimension that facilitates software PLA evolution. The Mae environment (Hoek, Mikic-Rakic, Roshandel, & Medvidovic, 2001), for example, uses ADL to facilitate incremental evolution by capturing all changes made to any architectural elements within a PLA. A key concept in the Mae environment is a system model that allows architectural concepts and configuration management to be mapped with each other through ADL syntax, that is, the ADL allows users to describe what and how the changes should be made to the model. The essence of the approach lies in the use of this model to integrate change management concepts (such as revisions, variants, and configurations) with architectural concepts (such as components, connectors, subtypes, and styles) through ADL descriptions. By mapping the generic system model onto a specific ADL, the design analyses of a software PLA and its evolution can be adapted for the purpose of maintaining the consistency of the architectural configurations captured by the model.

Similar to the idea of the Mae environment, the Koala Component Model (Ommering, Linden, Kramer, & Magee, 2002) also uses ADL to explicitly describe the architecture and provides a platform-centric approach to the design of PLAs for consumer electronics software. Specifically, the Koala Component Model allows variability options to be modeled explicitly via a property mechanism. Using a third-party versioning system, Koala can be used to capture the evolution of a PLA.

XADL (Dashofy, Hoek, & Taylor, 2003) is an XML-based ADL that is constructed from a set of extensible XML schemas. XADL also defines a set of associated libraries that provide a programmatic interface to XADL documents, and provide runtime facilities to create, store, and modify XADL documents. XADL and its associated libraries provide three important benefits for the purposes of supporting software PLA evolution: (1) the core of the XADL language supports variability in both space and time; in XADL, variabilities of artifacts are a natural and integral part of the language, (2) the language can be extended, which allows individual activities in the lifecycle to be able to attach additional information, and (3) the library provides a generic interface to easily access XADL documents, which supports the rapid construction of new tools supporting PLA evolution.

Summary

The related work described in this section all use text-based languages (such as structural languages or XML) to either explicitly capture the PLA evolution activities, or implicitly associate the evolution requirements with actual software components. Our approach is similar to this dimension of the related work in the sense that PLA evolution can be captured through the software PLA architecture itself, rather than through a separate dedicated language.

Hybrid Approaches

Some technologies span both text-based and graphical-based approaches. A well-known example in this category is called QVT (Query/View/Transformation) (OMG, 2005b), which is the OMG standard for model-to-model transformations. This technology provides a standard language to transform UML or custom model types from one type to another. It accepts XML Interchange (XMI) (OMG, 2005b) as input and output. Typical usage scenarios include automating transformation of a high-level design model into a more detailed model, transforming a UML model into a custom data model, or transforming one custom model type into another. The core benefits of this feature set are a standards-based language to express common model transformations with traceability, which provides repeatable results.

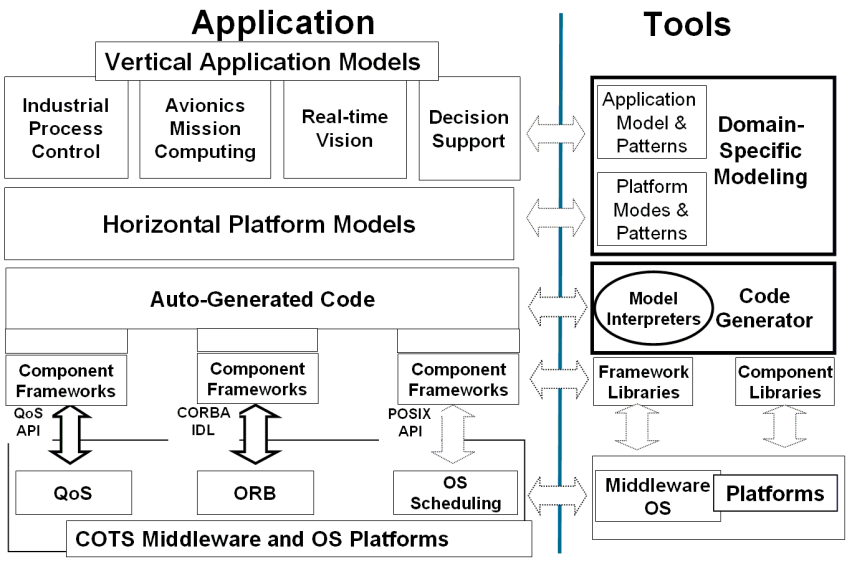
Summary

The specification of QVT defines both graphical syntax and textual syntax for the transformation language, but so far there lacks a full implementation of the specification. Moreover, while QVT is restricted to only XMI to XMI transformations, our approach does not have this restriction, so it can exploit any internal representation of the DSMLs.

MDE-BASED PRODUCT LINE ARCHITECTURE FOR DRE SYSTEMS

This section introduces an architecture of a MDE-based product line architecture for software-intensive DRE systems, focusing on the design concepts, common patterns, and software meth-

Figure 2. MDE-based product-line architecture for DRE systems



odology. An MDE-based design and composition approach for DRE systems entails the combination of DSMLs with reusable component frameworks. Figure 2 illustrates the high-level design principles and an overall architecture of an MDE-based PLA solution for software-intensive DRE systems that exploits a *layered* and *compositional* approach. This architecture takes advantage of layering and composition design principles (Krueger, 2006) to make the associated PLAs easier to develop and evolve than *ad hoc* approaches.

As shown in Figure 2, the PLA architecture is based on a core set of COTS middleware and OS platforms, component frameworks and domain-specific modeling languages. The right side of the figure shows the technologies available to implement the design artifacts on the left side. For example, the "Generator Technology" shown on the right can be used to build model interpreters that automatically generate code to bridge the gap between models and component frameworks.

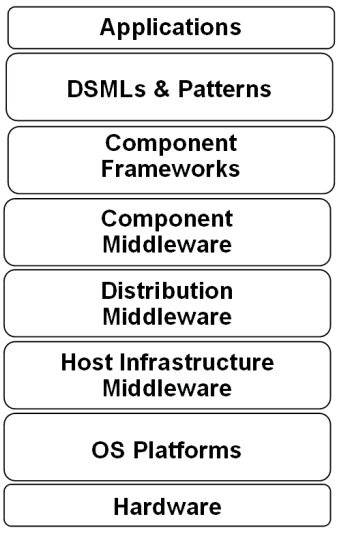
The remainder of this section introduces and defines key terms and concepts in the architecture shown in Figure 2.

Commercial-Off-The-Shelf (COTS) Middleware and OS Platforms

COTS middleware and OS platforms provide the infrastructure upon which DRE systems run. Many DRE systems are based on OS platforms with real-time scheduling capabilities. Examples of such OS platforms include *VxWorks* (Wind River Systems, 1998), *Timesys Linux* (Timesys, 2002), and *Windows CE* (Microsoft, 2007). Middleware is an enabling technology that allows multiple processes running on one or more machines to interact across a network. Middleware can be further decomposed into multiple layers (Schmidt, 2002), such as those shown in Figure 3 and described below:

- **Host Infrastructure Middleware:** The host infrastructure layer resides directly atop the operating system and provides a set of higher-level APIs that hide the heterogeneity of different operating systems and network protocols. The host infrastructure layer provides generic services to the upper middleware layers by encapsulating functionality that would otherwise require much tedious, error-prone, and nonportable code, such as socket programming and thread manipulation primitives. Examples of such middleware include ACE (Schmidt, 1993), Real-time Java (Bollella et al., 2000) and Rocks (Zandy & Miller, 2002).
- **Distribution Middleware:** The distribution layer resides atop the host-infrastructure layer and provides high-level programming abstractions, such as remote object operations. Using the distribution layer, a developer can write a distributed application in a similar way to a stand-alone application.

Figure 3. OS, middleware, DSML and application layer relationships



CORBA 2.x (OMG, 2003), DCOM (Microsoft, 2000), Java RMI (Sun Microsystems, 2000) and Data Distribution Service (DDS) (OMG, 2004a) are the main solutions to distribution middleware.

- **Component Middleware:** The component middleware layer resides atop the distribution middleware layer and adopts the component-based software engineering approach (Heineman & Councill, 2001) to allow maximum reuse of software components. Component middleware also provides mechanisms to configure and control key distributed computing aspects, such as connecting event producers to event consumers and managing transactional behavior, separate from the functional aspects of the application. Examples of component middleware platforms include Enterprise Java Beans (EJB) (Sun Microsystem, 2001) and OMG Corba Component Model (CCM) (OMG, 2005a).

Because many DRE systems require a loosely-coupled distribution architecture to simplify extensibility, COTS middleware typically provides event-driven publish/subscribe communication mechanisms, which help reduce ownership costs by defining clear boundaries between the components in the application. Such mechanisms reduce dependencies and maintenance costs associated with replacement, integration, and revalidation of components. COTS middleware and OS platforms are designed to maintain the commonality, portability, reusability, and applicability of software for different domains.

Component Frameworks

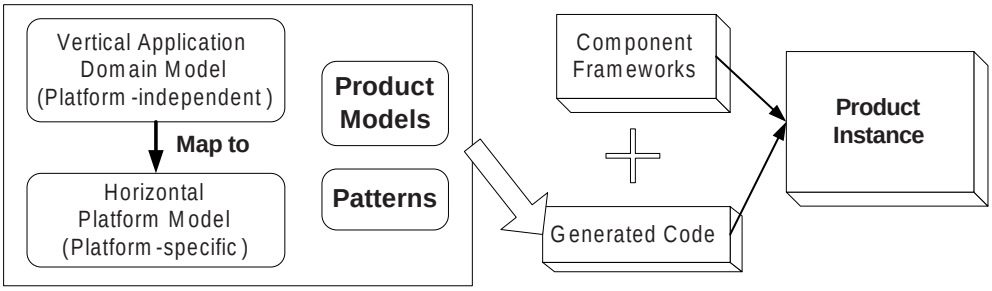
Component frameworks provide reusable domain-specific building blocks for PLAs of DRE systems. As illustrated in Figure 3, component frameworks reside atop COTS middleware and OS platforms. The key difference between component frameworks and component middleware is that the latter is domain independent while the former is domain-specific. Component frameworks define "semicomplete" applications that embody domain-specific object structures and functionality to raise the level of abstraction at which the software product instance is composed, and offer product-line specific environments to capture the variabilities. Components in such a framework coordinate with each other to provide core functionalities for a family of related applications. Complete applications can be composed by inheriting from or instantiating framework components.

Examples of component frameworks include the Boeing Bold Stroke product line architecture (Schulte, 2003) in the avionics mission computing domain and Siemens Building Technology APOGEE product line architecture (Siemens, 2007) in the building automation domain. For example, the Boeing Bold Stroke PLA supports many Boeing product variants using a component-based platform. The Boeing Bold Stoke PLA supports systematic reuse of mission computing functionality and is configurable for product-specific functionality and execution. The philosophy of component frameworks is to develop reusable components that are well-defined and have specific use contexts and variability points, which helps reduce the effort associated with using low-level middleware interfaces or OS APIs.

Domain-Specific Modeling Languages (DSMLs) and Patterns

DSMLs and patterns facilitate the model-based design, development, and analysis of DRE systems. Figure 4 shows how DSMLs and patterns can be combined with component frameworks to build product instances. A DSML can represent either a vertical application domain model (specific to

Figure 4. Integration of domain-specific modeling and component frameworks



concerns within a specific industry or domain) or a horizontal model (generic to concerns that span several domains).

Vertical application domain models address the problems arising within a particular domain, and they are often modeled in a platform-independent manner (Frankel, 2003). Examples of such vertical application domains include industrial process control, telecommunications, and avionics mission-critical systems. Some DSML examples developed for vertical domains include the *Saturn Site Production Flow* (SSPF), which is a manufacturing execution system serving as an integral and enabling component of the business process for an automotive factory (Long, Misra, & Sztipanovits, 1998). Another example is the *Embedded System Modeling Language* (ESML) (Karsai, Neema, Abbott, & Sharp, 2002), which models mission computing embedded avionics applications in the Boeing Bold Stroke PLA.

Horizontal platform domain models are also called platform-specific models (Frankel, 2003). A platform-specific model is a model of a system that is linked to a specific technological platform (e.g., a specific middleware platform, operating system or database). An example of a DSML for horizontal platforms is the *Rhapsody* modeling environment (iLogix, 2006), which allows application generation for embedded software platforms based on many real-time operating systems. Other examples of DSMLs for horizontal platforms include the *Platform Independent Component Modeling Language* (PICML) (Balasubramanian et al., 2005a) and J2EEML (White, Schmidt, & Gokhale, 2005), which facilitate the development, deployment, and configuration of QoS-enabled component-based DRE systems based on CCM and EJB, respectively.

The main idea is that it should be possible to use a model transformation technique to transform vertical application domain models to a horizontal platform domain model. Regardless of whether the DSMLs target horizontal or vertical domains, *model interpreters* can be used to generate various artifacts (such as code and metadata descriptors for deployment and configuration), which can be integrated with component frameworks to form executable applications or simulations. Key advantages of using DSMLs and patterns in PLAs are to rigorously capture the key roles and responsibilities of a product instance and help automate repetitive tasks that must be accomplished for each product instance.

In summary, an MDE-based PLA for software-intensive systems must be based on an architecture that adheres to well-documented

principles of architectural design with a clear separation of commonalities and appropriate provisions for incorporating variations by integrating vertical/horizontal DSMLs, component frameworks, middleware and OS platforms. In this architecture, MDE technologies are used to model PLA features and glue components together; for example, they could be utilized to synthesize deployment artifacts for standard middleware platforms (Balasubramanian et al., 2006).

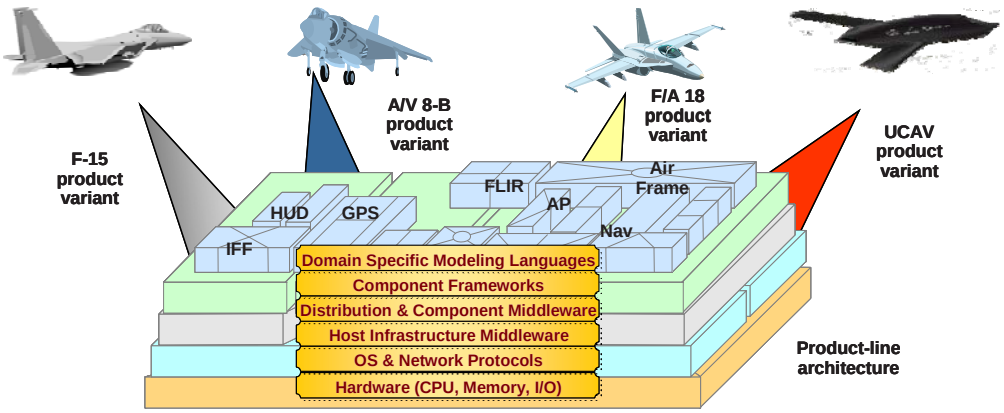
OVERVIEW OF THE BOEING BOLD STROKE PLA AND EQAL MDE TOOL

This section introduces a case study based on a real-time avionics mission computing product line called Boeing Bold Stroke and describes the structure and functionality of the *Event QoS Aspect Language* (EQAL) MDE tool based on this product line. The Boeing Bold Stroke PLA supports many Boeing product variants (e.g., F/A-18E, F/A-18F, F-15E, and F-15K) using a component-based publish/subscribe pattern (Gamma et al., 1995). The EQAL MDE tool is intended to reduce many complexities associated with the integration, deployment and configuration of different implementations of publish/subscribe mechanism. The Bold Stroke PLA and its associated models in EQAL will serve as the case study throughout this chapter.

Overview of Boeing Bold Stroke Product Line Architecture

Figure 5 illustrates the Boeing Bold Stroke PLA (Sharp, 1999), which was developed by Boeing in the mid-1990s to support systematic reuse of avionics mission computing functionality and is configurable for product-specific functionality (such as heads-up display, navigation, and sensor management) and execution environments (such as different networks/buses, hardware, operating systems, and programming languages) for a variety of military aircraft. Bold Stroke is a very complex framework with several thousand components implemented in several million lines of C++ code.

Figure 5. Boeing bold stroke product line architecture



The Boeing Bold Stroke architecture contains a set of event-driven component-based component frameworks built atop (1) The ACE ORB (TAO) (Schmidt, Levine, & Mungee, 1998), which implements key Real-time CORBA (OMG, 2005a) features, and (2) TAO's Real-time Event Service (Harrison, Levine, & Schmidt, 1997), which implements the publish/subscribe architectural pattern. Bold Stroke uses a Boeing-specific component model called PRISM (Roll, 2003), which implements a variant of the CORBA Component Model (CCM) atop TAO.

Following the CCM specification, PRISM defines the following types of ports, which are named interfaces, and connection points components used to collaborate with each other:

- **Facets**, which define named interfaces that process method invocations from other components.
- **Receptacles**, which provide named connection points to facets provided by other components.
- **Event sources and event sinks**, which indicate a willingness to exchange event messages with one or more components via event channels.

Bold Stroke is a representative PLA for DRE systems in the real-time avionics mission computing domain. Its event-driven communication architecture employs a control flow/data flow (Sharp, 1999) principle, where control flow represents the movement of execution through a software system, while the data flow represents the movement of data through a software system. Depending on requirements, different product variants in the Boeing Bold Stroke PLA may require different levels of QoS assurance for event communication, including timing constraints, event delivery latency, jitter, and scalability. Even within the same product variant, different levels of QoS assurance must be ensured for different communication paths, depending on criticality of the data. For example, the communication path between a collision radar component and the LED display component must have much more stringent timeliness deadline requirements than regular GPS components and navigation display components.

To alleviate the complexity in provisioning the event-driven publish/subscribe services and their QoS assurance in the Boeing Bold Stroke PLA, we designed an MDE-based tool called the Event QoS Aspect Language (EQAL) that can automate and simplify the integration of publish/subscribe services into QoS-enabled component-based systems.

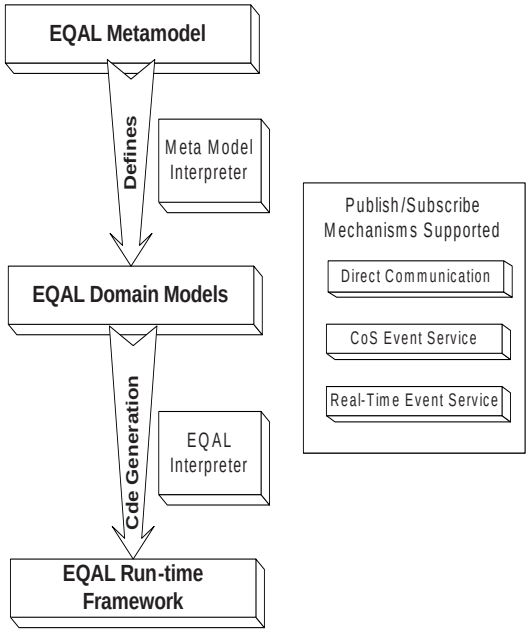
Overview of the EQAL MDE Tool

One core part of the EQAL MDE tool is the EQAL DSML (Edwards, Deng, Schmidt, Gokhale, & Natarajan, 2004), which is implemented using the Generic Modeling Environment (GME) (Lédeczi, Nordstrom, Karsai, Volgyesi, & Maroti, 2001). The GME is a toolkit that supports the development of DSMLs. The EQAL DSML provides an integrated set of metamodels, model interpreters, and standards-based component middleware that allow DRE system developers to visually configure and deploy event-driven communication mechanisms in DRE systems via models instead of programming them manually. The EQAL DSML is an example that supports a horizontal platform domain; that is, it is not restricted to a particular vertical application domain, but instead can be leveraged by multiple vertical domains. In this case study, we describe how EQAL was applied to the Bold Stroke avionics mission computing PLA.

As shown in Figure 6, EQAL is a layered architecture that supports several types of abstractions, which are subject to change stemming from domain evolution as explained below:

- The *bottom layer* in the architecture is the EQAL Runtime Framework, which is a por-

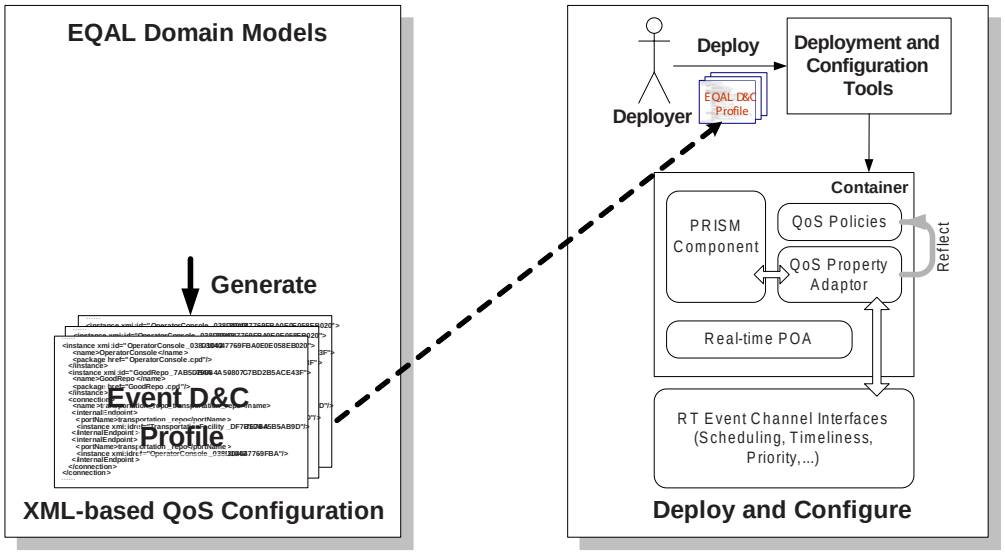
Figure 6. EQAL MDE tool architecture



table, OS-independent middleware framework based on light-weight CCM (OMG, 2004b). The EQAL Runtime Framework provides an extensible way to deploy various event-driven publish/subscribe mechanisms, including a two-way event communication mechanism based on direct method invocation instead of using a mediator channel.

- The *middle layer* in the EQAL architecture is a set of domain models that represent instances of the modeled DRE systems. These models are created using the EQAL DSML and are used to capture the structural and behavioral semantic aspects of event-driven DRE systems.
- The *top layer* of the EQAL architecture consists of a metamodel that enables developers to model concepts of event-driven DRE systems, including the configuration

Figure 7. Code generation from EQAL domain model



and deployment of various publish/subscribe services. This layer also contains several model interpreters that synthesize different types of configuration files that specify QoS configurations, parameters, and constraints, such as the threading model for event dispatching, event filtering configuration, and event channel federation configurations (Edwards et al., 2004). The EQAL interpreters automatically generate publish/subscribe service configuration files and service property description files needed by the underlying EQAL Runtime Framework and selected middleware.

As shown in Figure 7, EQAL allows DRE system deployers to create and synthesize publish/subscribe QoS configurations and deployments via graphical models (i.e., EQAL domain models) that are much easier to understand and analyze than hand-crafted code. During the modeling phase, EQAL ensures that dependencies between configuration parameters are enforced by declaring constraints on the contexts in which individual options are valid (e.g., priority-based thread allocation policies are only valid with component event connections that have assigned priorities). EQAL can then automatically validate configurations and notify users of incompatible QoS properties during model validation, rather than at component deployment and runtime. The generated XML-based QoS configuration and deployment descriptors can then be fed into deployment and configuration runtime tools to deploy and configure the components and real-time event channels within the Boeing Bold Stroke.

SUPPORT MDE-BASED PLA EVOLUTION WHEN FACING DOMAIN EVOLUTION

This section examines the following challenges associated with evolving MDE-based PLAs:

1. Challenges stemming from capturing new requirements into existing MDE-based PLAs for DRE systems.
2. Challenges stemming from migrating existing domain models with MDE-based PLA evolution.

For each challenge, we explain the context in which the challenge arises and identify key problems that must be addressed. Many of these challenges also exist in MDE-based PLAs for DRE systems, so they are not limited solely to event-driven DRE systems as described in our case study. In the remainder of this section, we first discuss the challenges and solutions associated with domain-specific component framework evolution and DSML evolution, then the challenges and solutions associated with domain model evolution.

Challenges Stemming from Capturing New Requirements into Existing MDE-based PLAs for DRE Systems

Context

Evolution is a natural occurrence in software development and an inevitable part of the software PLA lifecycle (Chapin, Hale, Kham, Ramil, & Tan, 2001). The changes may be initiated to correct, improve, or extend assets or products. Because assets are often dependent on other assets, changes to one asset may require corresponding changes in other assets. Moreover, changes to assets in PLAs can propagate to affect all products using these assets. A successful process for PLA evolution must therefore manage these changes effectively (McGregor, 2003).

Problem: New Requirements Impact Metamodels and Component Frameworks

DRE systems must evolve to adapt to changing requirements and operational contexts such as supporting new features. In addition, when some emerging technologies become sufficiently mature, it is often desirable to integrate them into existing PLAs for DRE systems. Figure 8 shows different affected scopes caused by such evolution.

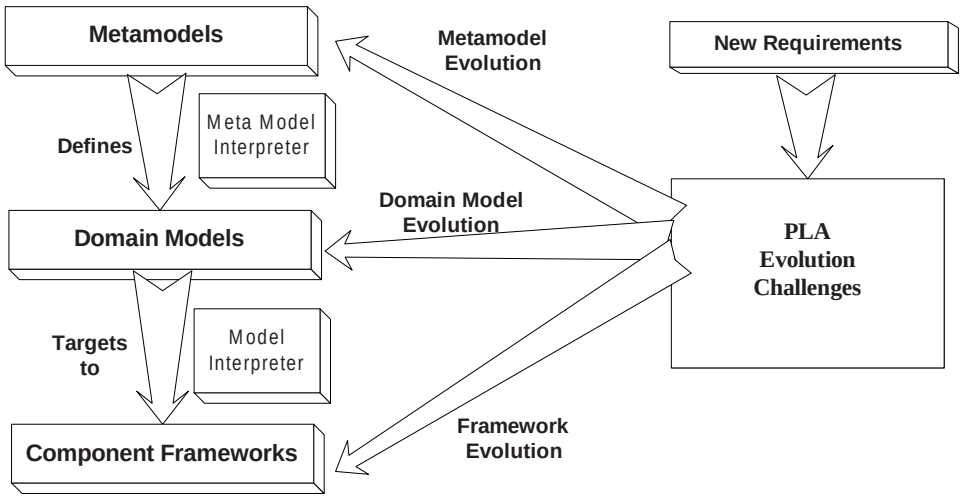
In our Boeing Bold Stroke case study, for example, depending on system requirements, different product variants in the Bold Stroke PLA may require different levels of QoS assurance for event communication, including timing constraints, event delivery latency, jitter, and scalability. Even within the same product variant, different levels of QoS assurance may be required for different communication paths, depending on system criticality (e.g., certain communication paths between components may require more stringent QoS requirements than others).

Solution: Evolve a PLA Systematically Through Framework and Metamodel Enhancement.

A layered PLA can reduce software design complexity by separating concerns and enforcing boundaries between different layers. Because different layers in a PLA need to interact with each other through predefined interfaces, to integrate new requirements into a PLA, all layers must evolve in a systematic manner. This evolution can be generalized to the following three steps:

1. **Component framework evolution.** As discussed earlier, frameworks are often built atop middleware and OS platforms to provide the runtime environment of DRE systems. As a result, whenever a DRE system must evolve to adapt to new requirements, component frameworks are often affected because they have direct impact on the system.
2. **DSML evolution.** DSML metamodels and interpreters are often used to capture the

Figure 8. Challenges stemming from adding new requirements into model-driven software PLAs



variability and *features* of DRE systems to expose different capabilities for different product variants. As discussed previously and shown in Figure 2, typically the DSMLs for vertical application domains have a higher level of abstraction than DSMLs for horizontal platform domains. These lower level DSMLs are built atop domain-specific component frameworks and are often used to glue different component framework entities together to form a complete application. Therefore, the evolution of lower level DSMLs should be performed after framework evolution is completed.

3. **Domain model evolution.** The DSML metamodel defines a type system to which domain models must conform. Because the changes to the metamodel of a DSML often invalidate the existing domain models by redefining the type system, domain model evolution must be performed after the DSML evolution.

In the remainder of this section, we further elaborate the solution approach and describe how it applies to our case study.

Component Framework Evolution

Component frameworks consist of a set of core reusable components that can be configured using well-defined interfaces. In order to capture the commonalities of software PLAs, one must formulate a set of usage patterns. The component frameworks encapsulate these usage patterns and provide reusable libraries that contain wrapper façades for the underlying implementation classes and shield component developers from tedious and error-prone programming tasks associated with lower-level details. Component frameworks are typically designed by analyzing various potential problems that the frameworks might address and identifying which parts of each solution are the same and which areas of each solution are unique through the SCV analysis.

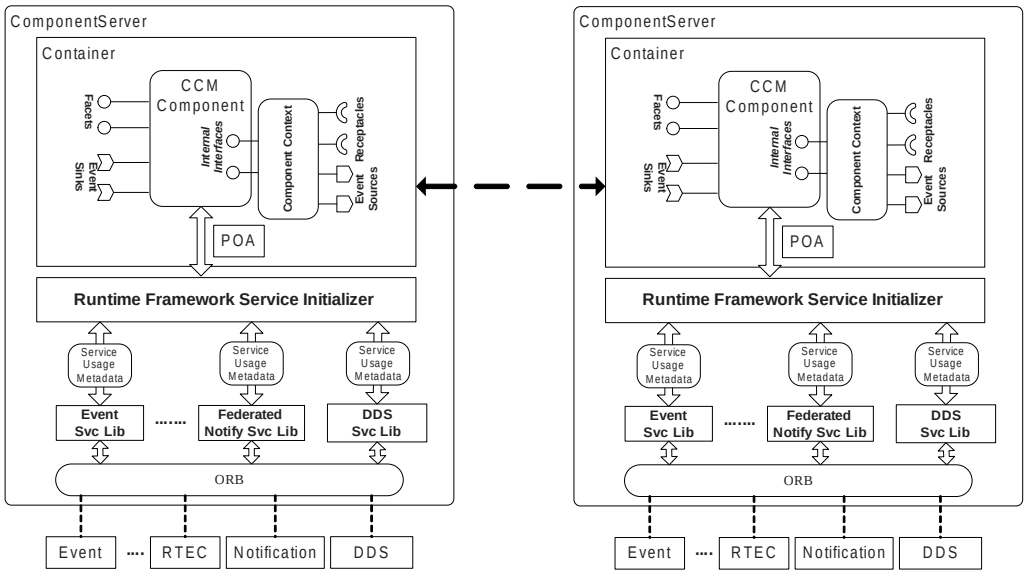
The first step is to define the domains (i.e., the problem areas a framework addresses) and the context of the framework. The next step is to define the attributes that recur across all members of the family of products based on the framework. The final step is to describe the attributes unique to the different members of the family of products. The SCV analysis requires extensive knowledge about the domain and the PLA requirements so one can reason what parts of the system should be implemented by the framework (commonalities) and what parts of the system should be specialized in subclasses or parameters (variabilities). To implement such design usually requires effective and skillful use of programming language features, such as templates and virtual functions, in conjunction with design patterns (Gamma et al. 1995).

Applying the Solution to the EQAL Case Study

In our EQAL case study, the scope is to design a framework to simplify the event communication between the event sources and event sinks of PRISM components. The commonality in this scope is straightforward, that is, every software product instance should implement an event-driven publish/subscribe pattern. The variability of the EQAL Runtime Framework results from different concrete service types that provide different interfaces and different QoS mechanism for the event communication, as shown in Figure 9. Because different real-time publish/subscribe services depend on different representations of real-time QoS properties, the EQAL Runtime Framework implements the adapter pattern that converts a service-independent representation of real-time properties into a service-specific representation.

The benefits of EQAL's design are twofold: (1) component developers need not concern

Figure 9. Component framework architecture of EQAL

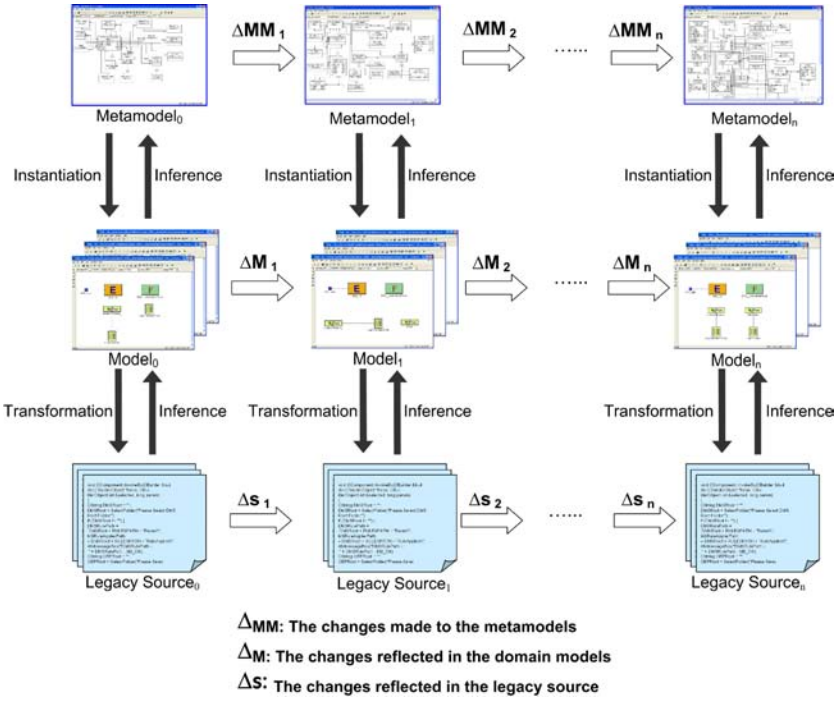


themselves with peculiar configuration interfaces, and (2) no matter what changes occur to the underlying publish/subscribe services, the interface exposed to components does not change. The EQAL Runtime framework also implements the strategy pattern to enhance the extensibility by allowing new publish/subscribe services to be easily plugged-in. This design results in a pluggable publish/subscribe service implementation that is interchangeable and extensible, and enables all event communication mechanisms supported by EQAL to provide the same interface, yet can also be configured with different strategies and QoS configurations even facing the domain evolution of adding new publish/subscribe service types.

DSML Evolution

The core component in the DSML is the metamodel. To help understand the context of domain evolution, Figure 10 presents a matrix of several evolution tasks that require automated assistance to manage the various dependencies among metamodels, instance models, and corresponding source code. As shown at the top of Figure 10, a metamodel represents a modeling language definition that is instantiated to represent end-user intentions in a specific domain. Elements in the instance models (middle of figure) have metatypes that are specified in the metamodel. A vertical transformation (i.e., a transformation that goes across abstraction layers) exists between the instance models and the legacy source code at the bottom, which represents updates that are needed in one artifact that are triggered by a change at a different layer of abstraction. Correspondingly, horizontal transformation occurs at the same layer of abstraction to address changing requirements (i.e., the Δ at each layer represents a horizontal transformation).

Figure 10. A matrix of evolution activities within DSMLs



To simplify the evolution of DSMLs, the reusability of metamodels is crucial when the domain becomes complex. Ideally, a metamodel can be developed based on a set of reusable (usually smaller) metamodel units, with different units capturing different aspects in the domain of interest. For example, these metamodel units might include different variations of signal flow, finite state machines, data type specifications, and petri-nets. The unified metamodel can then extend these units and glue them together. This technique is called *compositional metamodeling* (Karsai et al., 2004), and the main motivation of this technique is to make metamodels more scalable and easier to evolve.

Composition metamodeling provides a capability for reusing and combining existing modeling languages and language concepts. When changes need to be made in the metamodel units to reflect a better understanding of the given aspect in the domain, such changes can be propagated automatically to the metamodels that utilize them. Furthermore, by precisely specifying the extension and composition rules, models specified in the original domain language can be automatically translated to comply with the new, extended and composed, modeling language. Another important benefit of compositional modeling is its ability to add new capabilities while simultaneously leveraging prior constraints and model generators of existing DSMLs. Thus, it is ideal for evolving existing DSMLs to address new requirements.

Applying the Solution to the EQAL Case Study

EQAL is implemented within GME, which offers the compositional modeling capability. When new publish/subscribe services are integrated, a new DSML can be designed within GME and import the old EQAL metamodel as a reusable "library." Apart from being read-only, all objects in the metamodel imported through the library are equivalent to objects created from scratch. Because the new publish/subscribe services share much commonality between the existing publish/subscribe services that EQAL already supports, when the old EQAL metamodel is imported as a library, subtypes can be created and instances from the metamodel library can refer to library objects through references.

Challenges Stemming from Migrating Existing Domain Models with MDE-Based PLA Evolution

Context

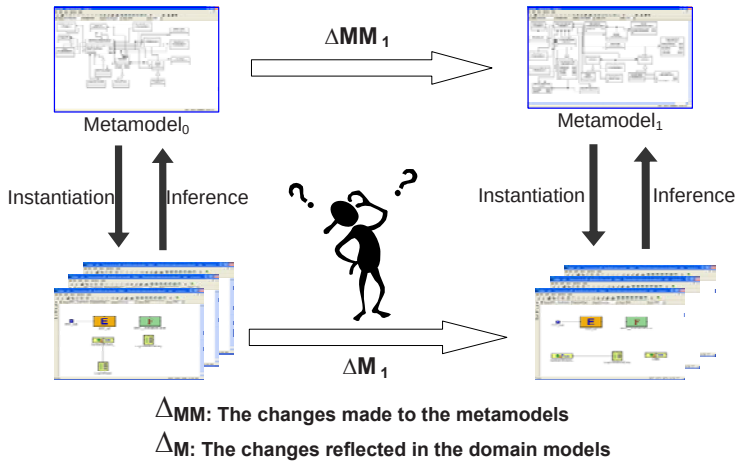
The primary value of the MDE paradigm stems from the models created using the DSML. These models specify the system from which the executable application can be generated or composed. Changes to the system can be modeled and the resulting executable model is thus a working version of the actual system. Unfortunately, if the metamodel is changed, all models that were defined using that metamodel may require maintenance to adapt to the semantics that represent the system correctly. Without ensuring the correctness of the domain models after a change to the domain, the benefits of MDE will be lost. The only way to use instance models based on the original metamodel is to migrate them to use the modified metamodel. During this migration process, we must preserve the existing set of domain model assets and allow new features to be added into domain models; ideally, with as little human intervention as possible.

Problem: Existing Domain Model Evolution Techniques Require Excessive Human Intervention

As illustrated in Figure 11, to preserve the existing set of domain model assets, old domain models must be transformed to become compliant with the changed metamodel. In the MDE research community, particularly in the DSML community, research has been conducted on using model transformation to address metamodel evolution (Sprinkle et al., 2003, Gray et al., 2006, Jouault & Kurtev, 2006). The underlying structure of models, particularly visual models, can be described by graphs. Model transformation research has therefore often been conducted in the context of graph transformation. In particular, recent research (Balogh & Varro, 2007; Vizhanyo, Agrawal, & Shi, 2004) has shown that graph transformation is a promising formalism to specify model transformations rules.

Most existing model transformation techniques, however, require the transformation be performed *after* the domain metamodel has changed. For example, when an old metamodel is modified and a new metamodel based on it is created, the model transformation must consider both the old metamodel and new metamodel as input, and then manually specify the model transformation rules based on these two metamodels by using a transformation specification language provided by the transformation tool. Although such a design approach could solve the model transformation problem, it introduces additional effort in specifying the model transformation rules, even if the metamodel evolution is minor (e.g., a simple rename of a concept in the metamodel). This additional effort is particularly high when the metamodels are complex, because the transformation tool must take both complex metamodels as input to specify the transformation.

Figure 11. Domain model evolution problem



Solution: Tool-Supported Domain Model Migration

To preserve the assets of domain models, our approach is to integrate *model migration* capabilities into the metamodeling environment itself. This approach is sufficiently generic to be applied to any existing metamodeling environment. A description of the change in semantics between an old and a new DSML is a sufficient specification to transform domain models such that they are correct in the new DSML. Moreover, the pattern that specifies the proper model migration is driven by the change in semantics and may be fully specified by a model composed of entities from the old and new metamodels, along with directions for their modification.

Below we describe how syntactic and semantic based model transformation approaches can be integrated to address the domain model migration problem.

Integration of Syntactic-Based and Semantic-Based Domain Model Evolution

The purpose of a DSML metamodel is to properly define the syntax and semantics to precisely describe software-intensive systems at a higher level of abstraction. As a result, both the syntax and the semantics of a metamodel can be affected when it is migrated from one version to another version. Consequently, to migrate a source domain model to the destination model, we must deal with both the syntax aspect and the semantics aspect.

Based on the characteristics of metamodel change, we have classified 8 atomic types of metamodel changes, as shown in Table 1. From this table, we can see that the all "additions" (addition of new type, new attribute or new association between types) to the metamodel will not affect the domain models because the target metamodel is a *superset* of the source metamodel and all the relationships of the source metamodel will still

Table 1. How atomic types of metamodel changes will affect domain models

Type of Metamodel Changes		Domain Model Change Required?
Additions		
1	Addition of new type	No
2	Addition of new attribute of a type	No
3	Addition of association between types	No
Deletions		
4	Deletion of an existing type	Yes
5	Deletion of an attribute of a type	Yes
6	Deletion of association between types	Yes
Modifications		
7	Replacing one type with another type	Yes
8	Replacing one association with another	Yes

be preserved. On the other hand, other types of metamodel changes including all "deletion" and "modifications" will result in unavoidable domain model change because the target metamodel is no longer a *superset* of the source metamodel.

These results provide intuition into the problem of domain model evolution. In some cases, the semantics can be easily specified. For example, if the metamodel designer deletes an atom called "foo" in the metamodel and creates a new atom called "bar" we can then specify the semantics of the change as:

```
replace(Atom("foo") -> Atom("bar"));
```

Syntactic metamodel changes, however, can often affect semantic changes, which result in a highly challenging task in model migration, that is, *semantic migration*. Semantic migration requires that the meaning of the old domain models be preserved after the transformation and that the new domain models conform to the entire set of static constraints required in the new domain.

For model migration, we generalized two approaches to perform model transformation with semantic migration. In the first approach, given two *distinct* metamodels, source metamodel and destination metamodel, we can perform a transformation that converts the source models in entirety to the destination models. This means that a complete set of rules is needed to convert each entity in the models. In the second approach, we create a *unified* metamodel (*old + new*), such that both old and new domain models are valid. Developers can then write transformation specifications that convert those parts of the model belonging to the source part of the paradigm to equivalent models in the destination part of the paradigm.

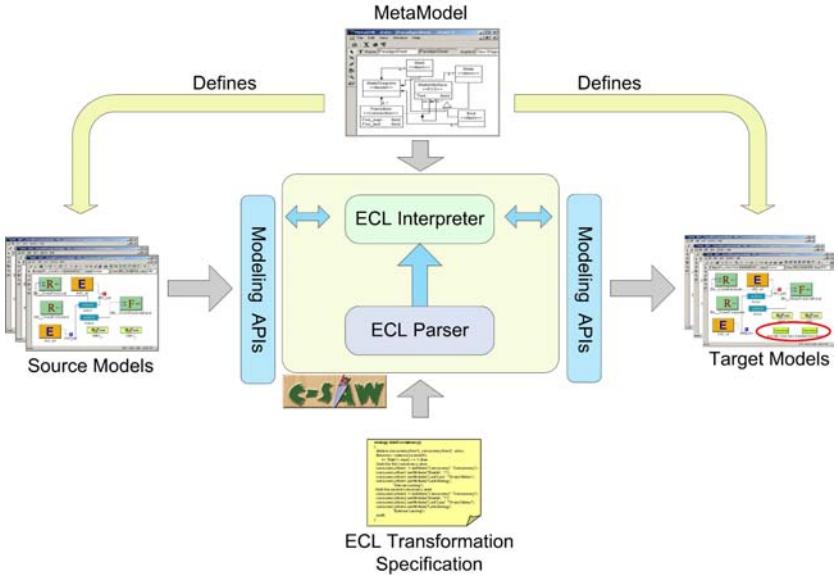
We have found that the second approach is much cleaner and user-friendly than the first approach because it requires much less human effort. In particular, in this second approach, after the unified metamodel is formulated, we can use an "SQL-like" declarative language that allows one to query and change the model to define model transformation rules. The *Embedded Constraint Language* (ECL), used by the C-SAW GME plug-in (Gray, Bapty, Neema, & Tuck, 2001), is such a language. ECL is a textual language for describing transformations on visual models. Similar to the Object Constraint Language (OCL) defined in OMG's UML specification, ECL provides concepts such as collection and model navigation. In addition, the ECL also provides a rich set of operators that are not found in the OCL to support model aggregations, connections, and transformations. ECL is a declarative language that allows one to specify the formal transformation rules of the syntax translator to capture the semantic migration.

In previous work, we showed how ECL can be used to accomplish several model transformation tasks (Gray et al., 2006). As an input language to C-SAW, ECL can support aspect modeling, as well as the ability to scale a base model to a larger model with replicated structures. Figure 12 illustrates an input source model being transformed by an ECL transformation rule to generate a new target model. An example of using ECL to handle the domain model migration in our case study is described in the next subsection.

Applying the Solution to the EQAL Case Study

In an old version of the EQAL metamodel there is a modeling object type called "`EventChannelGateway`," which can be used to federate different event channels together (Edwards et al., 2004). The definition of such a modeling element in a metamodel is similar to defining a class in C++ or Java. With domain evolution, this

Figure 12. Model transformation using the ECL (Gray et al., 2006)

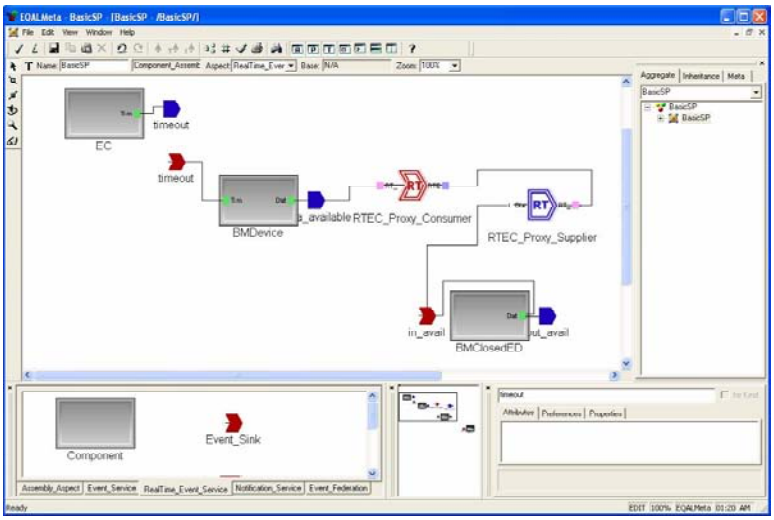


`EventChannelGateway` object type needs to be defined as an *abstract base* type (similar to the *abstract base* class concept in C++ or Java), and two new derived types called `IIOPGatway` and `UDPGateway` are defined in order to configure different underlying transport protocols between event channels. An issue arises regarding the type assignment of `EventChannelGateway` elements; depending on the context, these elements could be migrated to either the type of `IIOPGatway` or `UDPGateway`. In cases like these, it is quite challenging to discover the semantics of the change, that is, the semantics of the model elements cannot be deduced from the syntax. To require that such algorithms provide actual semantic migration capabilities necessitates human input because semantic changes in metamodels cannot be captured through syntactic changes alone.

Figure 13 shows the BasicSP application scenario (Balasubramanian et al., 2005b) in the Boeing Bold Stroke PLA. We use the BasicSP scenario as an example to showcase the problems encountered when evolving PLAs for component-based DRE systems and motivate the need of ECL for model transformation. In this figure, two component instances named `BMDevice` and `BMClosedED` are connected with each other through a real-time event channel provided by TAO's Real-time Event Service. An event channel consists of one `RTEC_Proxy_Consumer` module and `RTEC_Proxy_Supplier` module, which could be configured with various QoS settings, such as event dispatching threading models, priority configuration and periodic event processing configurations. Consider a domain evolution scenario, where the Real-time Event Service is not the desired choice for a particular Bold Stroke product variant, so it must be replaced with the TAO Federated Notification Service. In this case, the current domain model of Figure 13 will become invalid and must be migrated to the new EQAL DSML that supports the configuration of TAO's Federated Notification Service.

With ECL, a model transformation rule can be defined to accomplish the model migration task

Figure 13. EQAL configuring real-time event service between two components



Box 1.

```
 1. strategy ChangeToFNS() {
 2.    declare FNS_Proxy_Consumer, FNS_Proxy_Supplier : model;
 3.
 4.    // Find interested model elements…
 5.    if(atoms()->select(a | a.kindOf() = “RTEC_Proxy_Consumer”)->size() >= 1) then
 6.
 7.    //get the RTEC_Proxy_Consumer model element
 8.    //and its connections
 9.    …
10.    //delete the RTEC_Proxy_Consumer model element
11.    RTEC_Model.deleteModel(“RTEC_Proxy_Consumer”, “RTEC_proxy_consumer”);
12.
13.    //add the FNS_Proxy_Consumer model
14.    FNS_Proxy_Consumer:= addModel(“FNS_Proxy_Consumer”, “FNS_proxy_consum-
er”);
15.    FNS_Proxy_Consumer.setAttribute(“Reactive”, “1”);
16.    FNS_Proxy_Consumer.setAttribute(“LockType”, “Thread Mutex”);
17.
18.    //add the connections
19.    RTEC_Model.addConnection(“Event_Source_Proxy_Consumer”, event_source, FNS_
Proxy_Consumer);
20.    RTEC_Model.addConnection(“Proxy_Supplier_Event_Sink”, FNS_Proxy_Consumer,
event_sink);
21.
22.    //do similar to the FNS_Proxy_Supplier model
23.    …
24.    endif;
25. }
```

noted above. In the ECL, a strategy represents a transformation rule that is applied to a specific location of a model. A query can be written in the ECL to define a collection of models that need to be transformed, and a strategy can be invoked on the collection. The strategy in Box 1 specifies the desired model migration. The semantic meaning of this transformation is straightforward, that is, line 1 declares the strategy based on the ECL syntax; lines 4-10 find the interested model elements and their associations that are based on TAO's Real-time Event Service; line 11 removes the found model elements, and lines 13-20 replace these model elements and associations with TAO's Federated Notification Service. (see Box 1)

CONCLUSION

Change is a natural and inevitable part of the software-intensive system lifecycle. The changes may be initiated to correct, improve, or extend assets or products. Because assets are often dependent on other assets, changes to one asset may require

corresponding changes in other assets. Moreover, changes to assets in PLAs can propagate to affect all products using these assets.

To use MDE-based PLA technologies effectively in practice requires practical and scalable solutions to the *domain evolution problem*, which arises when existing PLAs are extended or refactored to handle unanticipated requirements or better satisfy existing requirements. For example, changing metamodels in a PLA often invalidates models based on previous versions of the metamodels. Although software developers can manually update their models or components developed with a previous metamodel to work with the new metamodel, this approach is clearly tedious, error-prone, and nonscalable. A successful process for PLA evolution must therefore manage these changes effectively.

To rectify these problems, this chapter describes a layered and compositional architecture to modularize system concerns and reduce the effort associated with domain evolution. This chapter illustrates via a case study how systematic evolution with three ordered steps can maintain the stability of domain evolution against MDE-based software PLAs, and how structural-based model transformations help reduce human effort by automatically transforming existing domain models based on metamodel-based rules.

The following is a summary of lessons learned from our experience in evolving product-lines using MDE tools:

- **DSMLs and component frameworks are highly synergistic:** An MDE approach expedites PLA development with the proper integration of DSMLs and component frameworks. The component frameworks help shield the complexities of the design and implementation of modeling tools, and decouple many aspects of concerns between the modeling tools and the executable systems. In our case study, if the publish/subscribe service type is the only missing or changing concern in the Boeing Bold Stroke PLA (which is typical in our case), little new application code must be written, yet the complexity of the generation tool remains manageable due to the limited number of well-defined configuration "hot spots" exposed by the underlying infrastructure. Likewise, when component deployment plans are incomplete or must change, the effort required is significantly less than starting from the raw component middleware without MDE tool support.
- **Declarative-based model transformation alleviates transformation effort:** Structural-based model transformations help maintain the stability of domain evolution of MDE-based DRE systems by automatically migrating domain models. A declarative-based model transformation language like ECL is an ideal approach in such a case. The case study presented in this chapter highlights the ease of specification and the general flexibility provided by the transformation engine.
- **Testing and debugging of transformation specification is still hard:** Transformation specifications, such as those used to specify the transformation strategy in this chapter, are written by humans and prone to error. To improve the robustness and reliability of model transformation, there is a need for testing and debugging support to assist in finding and correcting the errors in transformation specifications. Ongoing and future work on ECL focuses on the construction of testing and debugging utilities to ensure the correctness of ECL transformation specifications.

All software in this chapter can be downloaded from our Web sites. The EQAL framework is shipped as part of the CIAO and is available at http://download.dre.vanderbilt.edu. The EQAL DSML is available at http://www.dre.vanderbilt.

edu/cosmic/. C-SAW is available at http://www.cis.uab.edu/gray/research/C-SAW/.

FUTURE RESEARCH DIRECTIONS

This section discusses the future trends in the areas of MDE and component middleware, and how together they are impacting MDE-based PLAs, particularly for DRE systems.

Emerging Interest in Domain-Specific Modeling

The interest and adoption of DSMLs over the past decade has surged. Strong support for basic research has been committed by the large European Union ModelWare and ModelPlex projects, which are funded at 30M Euros (ModelWare Project, 2006). Metamodeling tools that support DSM continue to emerge from both commercial and open source projects (e.g., Microsoft's DSL Toolkit (Microsoft, 2006) and the Eclipse Modeling Project (Eclipse, 2007)), as well as numerous academic research projects (e.g., Vanderbilt's Generic Modeling Environment (GME, 2007)). Initial success stories from industry adoption of DSM have been reported: The newly created DSM Forum (DSM Forum, 2007) serves as a repository of several dozen successful projects (mostly from industry, such as Nokia, Dupont, Honeywell, and NASA) that have adopted DSM. Over the past 5 years, the annual DSM workshop at OOPSLA (52 international participants in 2007) provides a venue for reporting experiences in DSM research and practice.

Future Research Directions of MDE Tools for DRE Systems

MDE has already played an important role in the assembly, configuration and deployment lifecycle stages of today's DRE systems. We envision next generation MDE tools will seamlessly integrate all lifecycle stages of software product lines, including requirement management, functionality specification, QoS specification, system partitioning and implementation, component assembly and packaging, system configuration, system planning and analysis and runtime system management. With such seamless integration, models will become vital artifacts in all aspects of software PLA development lifecycle, and sophisticated model transformation techniques will bridge the gap between models in different lifecycle stages. The need for seamless integration of models across the lifecycle is driving the need for integration across a collection of different modeling tools, where each offers some advanced capability not found in another tool. The need for tool integration will continue to heighten the role that model transformation plays as the key enabler of model sharing (Sendall & Kozaczynski, 2003).

Future Research Directions of Component Middleware for DRE Systems

The success of component middleware technologies has resulted in DRE systems created by customizing pre-existing COTS components rather than creating them from scratch. The increased use of pre-existing components shifts the focus from development to configuration and deployment of COTS components. With more COTS components provided by different vendors, the capability of *heterogeneous* deployment becomes a challenging task to evolve today's DRE systems.

Future component middleware technologies will enable rapid development of adaptive large scale DRE systems to accommodate changing operating environments. To facilitate the development of large-scale DRE systems, component middleware must support the agility in business service provisioning within and across organizations while ensuring the quality of service. The combination of these two techniques will finally enable software PLA developers to capture and

represent adaptability of DRE systems at the business level and automatically translate this business adaptability into component and process adaptability.

As PLAs become more complex, they will be adopted into software-intensive systems of very large-scale, as typified by the focus of Ultra Large-Scale systems (Ultra Large-Scale, 2007). In such cases, it is not unrealistic to imagine the PLAs using multiple different middleware platforms. To accommodate these requirements demands new investigation in deployment and configuration across heterogeneous middleware platforms. This heterogeneity also adds to the challenges in provisioning QoS end-to-end for these PLAs. All these require novel modeling capabilities that can abstract away the heterogeneity.

We envision these future research directions will greatly simplify the development of MDE-based PLAs and make next generation DRE systems more robust.

ACKNOWLEDGMENT

This was supported in part by an NSF CAREER award (CCF-0643725).

REFERENCES

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., et al. (2002). *Component-based product line engineering with UML*. Addison-Wesley.

Balasubramanian, K., Balasubramanian, J., Parsons, J., Gokhale, A., & Schmidt, D.C. (2005a, March). A platform-independent component modeling language for distributed real-time and embedded systems. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Francisco, CA, (pp. 190-199).

Balasubramanian, K., Gokhale, A., Lin, Y., Zhang, J., & Gray, J. (2006, June). Weaving deployment aspects into domain-specific models. *International Journal on Software Engineering and Knowledge Engineering, 16*(3), 403-424.

Balasubramanian, K., Krishna, A., Turkay, E., Balasubramanian, J., Parsons, J., Gokhale, A., & Schmidt, D.C. (2005b, April). Applying model-driven development to distributed real-time and embedded avionics systems *International Journal of Embedded Systems, special issue on Design and Verification of Real-Time Embedded Software*.

Balogh, A., & Varro, D. (2007). The model transformation of the VIATRA2 framework. *Science of Computer Programming (Special Issue on Model Transformation).*

Bollella, G., Gosling, J., Brosgol, B., Dibble, P., Furr, S., Hardin, D., & Turnbull, M. (2000). *The real-time specification for Java*. Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture—a system of patterns.* John Wiley & Sons.

Chapin, N., Hale, J., Kham, K., Ramil, J., & Tan, W. (2001, January). Types of software evolution and software maintenance. *Journal of Software Maintenance: Research and Practice*, 3-30.

Clements, P., & Northrop, L. (2001). *Software product-lines: Practices and patterns.* Addison-Wesley.

Ultra large-scale systems: The report. (2007). CMU Technical Report. Retrieved March 7, 2008, from http://www.sei.cmu.edu/uls/

Coplien, J., Hoffman, D., & Weiss, D. (1998, November/December). Commonality and variability in software engineering. *IEEE Software, 15*(6), 37-45.

Dashofy, E.M., Hoek, A., & Taylor, R.N. (2002, May). An infrastructure for the rapid development of XML-based architecture description languages. In *Proceedings of the 24th International Conference on Software Engineering*, Orlando, FL, (pp. 266-276).

Deng, G., Lenz, G., & Schmidt, D.C. (2005, October). Addressing domain evolution challenges for model-driven software product-line architectures (PLAs). In *Proceedings of the MoDELS 2005 Workshop on MDD for Software Product-lines: Fact or Fiction?*, Montego Bay, Jamaica.

DSM Forum. (2007). *From domain-specific modeling forum*. Retrieved March 7, 2008, from http://www.dsmforum.org/tools.html

Eclipse Modeling Project. (2007). Retrieved March 7, 2008, from http://www.eclipse.org/modeling/

Edwards, G., Deng, G., Schmidt, D.C., Gokhale, A., & Natarajan, B. (2004, October). Model-driven configuration and deployment of component middleware publisher/subscriber services. In *Proceedings of the 3rd ACM International Conference on Generative Programming and Component Engineering*, Vancouver, Canada, (pp. 337-360).

Frankel, D.S. (2003). *Model driven architecture: Applying MDA to enterprise computing.* John Wiley & Sons.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

GME. (2007). *Generic modeling environment.* Retrieved March 7, 2008, from http://escher.isis.vanderbilt.edu/downloads?tool=GME

Gray, J., Bapty, T., Neema, S., & Tuck, J. (2001). Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM, 44*(10), 87-93.

Gray, J., Lin, Y., & Zhang, J. (2006, February). Automating change evolution in model-driven engineering. In D. Schmidt (Ed.), *IEEE Computer, Special Issue on Model-driven Engineering, 39*(2), 51-58.

Gray, J., Tolvanen, J., Kelly, S., Gokhale, A., Neema, S., & Sprinkle, J. (2007). Domain-specific modeling. In P. Fishwick (Ed.), *CRC handbook on dynamic system modeling.* CRC Press.

Heineman, G.T., & Councill, W.T. (2001). *Component-based software engineering: Putting the pieces together.* Addison-Wesley.

Harrison, T., Levine, D., & Schmidt, D.C. (1997, October). The design and performance of a real-time CORBA event service. In *Proceedings of OOPSLA*, Atlanta, GA, (pp. 184-200). ACM.

Hoek, A., Mikic-Rakic, M., Roshandel, R., & Medvidovic, N. (2001, September). Taming architectural evolution. In *Proceedings of the 8th European Software Engineering Conference (held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering)*, Vienna, Austria, (pp. 1-10).

IBM. (2007). *Rational software*. Retrieved March 7, 2008, from http://www-306.ibm.com/software/rational/

iLogix. (2006). *Rhapsody*. Retrieved March 7, 2008 from http://www.ilogix.com/sublevel.aspx?id=284

Jouault, F., & Kurtev, I. (2006, April). On the architectural alignment of ATL and QVT. In *Proceedings of ACM Symposium on Applied Computing* (pp. 1188-1195). Dijon, Bourgogne: France.

Karhinen, A., & Kuusela, J. (1998, February). Structuring design decisions for evolution. In *Proceedings of the Second International ESPRIT ARES Workshop*, Las Palmas de Gran Canaria, Spain, (pp. 223-234). Springer-Verlag.

Karsai, G., Maroti, M., Lédeczi, A., Gray, J., & Sztipanovits, J. (2004, March). Composition and cloning in modeling and metamodeling. *IEEE Transactions on Control System Technology, Special Issue on Computer Automated Multi-paradigm Modeling, 12*(2), 263-278.

Karsai, G., Neema, S., Abbott, B., & Sharp, D. (2002, August). A modeling language and its supporting tools for avionics systems. In *Proceedings of the 21st Digital Avionics Systems Conf*erence, (Vol. 1, pp. 6A3-1-6A3-13).

Klusener, S., Laemmel, R., & Verhoef, C. (2005). Architectural modifications to deployed software. *Science of Computer Programming, 54*, 143-211.

Krueger, C.W. (2002, August). Variation management for software production lines. In *Proceedings of the Second International Conference of Software Product Lines*, SPLC 2, San Diego, CA, (pp. 37-48).

Lédeczi, Á., Nordstrom, G., Karsai, G., Volgyesi, P., & Maroti, M. (2001). On metamodel composition. In *Proceedings of the 2001 IEEE International Conference on Control Applications*, Mexico City, Mexico, (pp. 756-760).

Long, E., Misra, A., & Sztipanovits, J. (1998, August). Increasing productivity at Saturn. *IEEE Computer, 31*(8), 35-43.

Macala, R.R., Stuckey, L., & Gross, D. (1996, May). Managing domain-specific, product-line development. *IEEE Software, 14*(13), 57-67.

Mens, T., & D'Hondt, T. (2000, March). Automating support for software evolution in UML, *Automated Software Engineering, 7*(1), 39-59.

McGregor, J.D. (2003). *The evolution of product-line assets* (Tech. Rep. No. CMU/SEI-2003-TR-005m ESC-TR-2003-005).

Microsoft Corporation. (2000). *Microsoft COM technologies DCOM*.

Microsoft Corporation. (2006). *Microsoft domain-specific language (DSL) tools: Visual studio 2005 team system*. Retrieved March 7, 2008, from http://msdn.microsoft.com/vstudio/teamsystem/workshop/DSLTools

Microsoft Corporation. (2007). *Windows embedded CE 6.0*. Retrieved March 7, 2008, from http://www.microsoft.com/windows/embedded/

ModelWare Project. (2006). Retrieved March 7, 2008, from http://www.modelware-ist.org/

OMG. (2002, April). *MOF 2.0 query/views/transformations RFP*. OMG document ad/2002-04-10.

OMG. (2003, July). *The common object request broker: Architecture and specification*.

OMG. (2004a, December). *Data distribution service*. OMG document, formal/04-12-02.

OMG. (2004b).*Light-weight CORBA component model*. OMG document, ptc/04-06-10.

OMG. (2005a, January). *Real-time CORBA specification*. OMG document, formal/05-01-04.

OMG. (2005b, November). *MOF QVT final adopted specification*. OMG document, ptc/05-11-01.pdf.

OMG. (2006, April). *CORBA component model*. OMG Document formal/2006-04-01 ed.

Ommering, R., Linden, F., Kramer, J., & Magee, J. (2002, March). The Koala Component Model for consumer electronics software. *IEEE Computer, 33*(3), 78-85.

Ran, A., & Kuusela, J. (1996, March). Design decision trees. In *Proceedings of the Eighth International Workshop on Software Specification and Design*, (p 172).

Roddick, J.F. (1992). Schema evolution in database systems: An annotated bibliography. *SIGMOD Record, 21*(4).

Roll, W. (2003, May). Towards model-based and CCM-based applications for real-time systems. In *Proceedings of the International Symposium on Object-oriented Real-time Distributed Computing (ISORC)*, Hokkaido, Japan, (pp. 75-82).

Schmidt, D. C. (1993). The ADAPTIVE communication environment: An object-oriented network programming toolkit for developing communication software. *Concurrency: Practice and Experience*, *5*(4), 269-286.

Schmidt, D. C. (2002, June). Middleware for real-time and embedded systems. *Communications of the ACM*, *45*.

Schmidt, D.C. (2006, February). Model-driven engineering. *IEEE Computer*, 25-32.

Schmidt, D.C., Levine, D., & Mungee, S. (1998, April). The design and performance of real-time object request brokers. *Computer Communications, 21*, 294-324.

Schulte, M. (2003, May). Model-based integration of reusable component-based avionics system. In *Proceedings of the Eighth IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'05)* (pp. 62-71). Seattle, WA.

Sendall, S., & Kozaczynski, W. (2003, September-October). Model transformation—the heart and soul of model-driven software development. *IEEE Software*, *20*(5), 42-45.

Sharp, D. (1999, October). Avionics product line software architecture flow policies. In *Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, St Louis, MO.

Siemens. (2007). Retrieved March 7, 2008, from http://www.us.sbt.siemens.com/bau/products/default.asp

Sprinkle, J., Agrawal, A., Levendovszky, T., Shi, F., & Karsai, G. (2003, April). Domain model translation using graph transformations. In *Proceedings of the Conference on Engineering of Computer-based Systems*, Huntsville, AL, (pp. 159-167).

Sprinkle, J., & Karsai, G. (2004, June). A domain-specific visual language for domain model evolution. *Journal of Visual Languages and Computing, 15*(3-4), 291-307.

Sun Microsystems. (2000). *Java remote method invocation specification*, revision 1.5, JDK 1.2, Oct. 1998.

Sun Microsystems. (2001). *Enterprise JavaBeans specification*. Retrieved March 7, 2008, from java.sun.com/products/ejb/docs.html

Szyperski, C. (2002). *Component software: Beyond object-oriented programming*. Addison-Wesley.

Timesys. (2002). *Predictable performance for dynamic load and overload*. Retrieved March 7, 2008, from www.timesys.com/prodserv/whitepaper/Predictable_Performance_1_0.pdf

Tolvanen, J.P., & Kelly, S. (2005). Defining domain-specific modeling languages to automate product derivation: Collected experiences. In *Proceeding of the 9th Software Product Line Conference*, Rennes, France, (pp. 198-209).

Vizhanyo, A., Agrawal, A., & Shi, F. (2004, October). Towards generation of efficient transformations. In *Proceeding of the ACM International Conference on Generative Programming and Component Engineering*, Vancouver, Canada, (pp. 298-316).

White, J., Schmidt, D.C., & Gokhale, A. (2005, October). Simplifying autonomic enterprise Java Bean applications via model-driven development: A case study. In *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica, (pp. 601-615).

Wind River Systems. (1998). *VxWorks 5.3.* Retrieved March 7, 2008, from ww.wrs.com/products/html/vxworks.html

Zandy, V. C., & Miller, B. P. (2002, September). Reliable network connections. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, (pp. 95-106).

ADDITIONAL READING

Batory, D. (2006). Multilevel models in model-driven engineering, product lines, and metaprogramming. *IBM Systems Journal*, *45*(3), 527-540.

Bézivin, J. (2001, July 29-August 03). From object composition to model transformation with the MDA. In *Proceedings of the 39th international Conference and Exhibition on Technology of Object-oriented Languages and Systems*, Washington, DC, (p. 350).

Bézivin, J. (2005). On the unification power of models. *Software and Systems Modeling, 4*(2), 171-188.

Billig, A., Busse, S., Leicher, A., & Süß, J. G. (2004, October 18-22). Platform independent model transformation based on triple. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, New York, (Vol. 78, pp. 493-511). New York: Springer-Verlag.

Bosch, J. (2000). *Design and use of software architectures: Adopting and evolving a product-line approach.* ACM Press/Addison-Wesley.

Buck, J., Ha, S., Lee, E.A., & Messerschmitt, D.G. (1991). Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal on Computer Simulation, 4*, 155-182.

Czarnecki, K., Antkiewicz, M., & Kim, C. H. (2006, December). Multi-level customization in application engineering. *Communications of the ACM, Special Issue on Software-Product Line Engineering.*

Czarnecki, K., & Helsen, S. (2003, October). Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA*, Anaheim, CA.

Durán, A., Bernárdez, B., Genero, M., & Piattini, M. (2004, October). Empirically driven use case metamodel evolution. In *Proceedings of 7th International Conference Unified Modeling Language: Modeling Languages and Applications*, Lisbon, Portugal, (pp. 1-11).

Fairbanks, G., Garlan, D., & Scherlis, W. (2006, October). Design fragments make using frameworks easier. *ACM SIGPLAN Notices, 4*(10).

Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). *Software factories: Assembling applications with patterns, models, frameworks, and tools.* John Wiley & Sons.

Gokhale, A., Balasubramanian, K., Balasubramanian, J., Krishna, A., Edwards, G.T., Deng, G., et al. (2007). Model driven middleware: A new paradigm for deploying and provisioning distributed real-time and embedded applications. In M. Aksit (Ed.), *Elsevier Journal of Science of Computer Programming: Special Issue on Model Driven Architecture.*

Gokhale, A., Schmidt, D. C., Natarajan, B., & Wang, N. (2002, October). Applying model-integrated computing to component middleware and enterprise applications. *The Communications of the ACM Special Issue on Enterprise Components, Service and Business Rules, 45.*

Gore, P., Schmidt, D.C., Gill, C., & Pyarali, I. (2004, May). The design and performance of a real-time notification service. In *Proceedings of the 10th Real-time Technology and Application Symposium* (pp. 112-120). Toronto, CA.

Gray, J., Bapty Neema, T., & Tuck, J. (2001, October). Handling crosscutting constraints in domain-specific modeling. *The Communications of the ACM*.

Johnson, R.E. (1997, October). Frameworks = (components + patterns). *Communications of the ACM*, *40*(10), 39-42.

Ledeczi, A., Bakay, A., Maroti, M., Volgysei, P., Nordstrom, G., Sprinkle, J., & Karsai, G. (2001, November). Composing domain-specific design environments. *IEEE Computer,* 44-51.

Lenz, G., & Wienands, C. (2006, July 6). *Practical software factories in .NET.* Apress.

Pohl, K., Bockle, G., & Linden, F., van der. (2005). *Software product line engineering.* Berlin, Heidelberg, New York: Springer-Verlag.

Schmidt, D.C., Stal, M., Rohert, H., & Buschmann, F. (2000). *Pattern-oriented software architecture: Concurrent and networked objects.* John Wiley & Sons.

Sztipanovits, J., & Karsai, G. (1997, April). Model-integrated computing. *IEEE Computer*, 110-112.

Tourwé, T., & Mens, T. (2003, September). Automated support for framework-based software evolution. In *Proceedings of the International Conference on Software Maintenance*, (p. 148). Washington, DC: IEEE Computer Society.

This work was previously published in Designing Software-Intensive Systems: Methods and Principles, edited by P. Tiako, pp. 102-132, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 3.11
Reverse Engineering from an XML Document into an Extended DTD Graph

Herbert Shiu
City University of Hong Kong, Hong Kong

Joseph Fong
City University of Hong Kong, Hong Kong

ABSTRACT

The extensible markup language (XML) has become a standard for persistent storage and data interchange via the Internet due to its openness, self-descriptiveness, and flexibility. This article proposes a systematic approach to reverse engineer arbitrary XML documents to their conceptual schema, extended DTD graphs, which are DTD graphs with data semantics. The proposed approach not only determines the structure of the XML document, but also derives candidate data semantics from the XML element instances by treating each XML element instance as a record in a table of a relational database. One application of the determined data semantics is to verify the linkages among elements. Implicit and explicit referential linkages are among XML elements modeled by the parent-children structure and ID/IDREF(S), respectively. As a result, an arbitrary XML document can be reverse engineered into its conceptual schema in an extended DTD graph format.

INTRODUCTION

As the extensible markup language (XML; Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2004) has become the standard document format, the chance that users have to deal with XML documents with different structures is increasing. If the schema of the XML documents in a document type definition (DTD; Bosak, 1998) is given or derived from the XML documents right away (Kay, 1999; Moh, Lim, & Ng, 2000), it is easier

to study the contents of the XML documents. However, the formats of these schemas are hard to read and not very user friendly.

XML has been the common format for storing and transferring data between software applications and even business parties as most software applications can generate or handle XML documents. For example, a common scenario is that XML documents are generated and based on the data stored in a relational database; there have been various approaches for doing so (Fernandez, Morishima, & Suciu, 2001; Thiran, Estiévenart, Hainaut, & Houben, 2004). The sizes of XML documents that are generated based on the data stored in databases can be very large. Most probably, these documents are stored in a persistent storage for backup purposes as XML is the ideal format that can be processed by any software applications in the future.

In order to handle the above scenario, it is possible to treat XML element instances in an XML document as individual entities, and the relationships from the different XML element types can be determined by reverse engineering them for their conceptual models, such as extended DTD graphs with data semantics. As such, users can have a better understanding of the contents of the XML document and further operations with the XML document become possible, such as storing and querying (Deutsch, Fernandez, & Suciu, 1999; Florescu & Kossmann, 1999; Kanne & Moerkotte, 2000).

This article proposes several algorithms that analyze XML documents for their conceptual schema. Two main categories of XML documents exist: data centric and narrative. As the contents of narrative XML documents, such as DocBook (Stayton, 2008) documents, are mainly unstructured and their vocabulary is basically static, the necessity of handling them as structured contents and reverse engineering them into conceptual models is far less than that of handling data-centric ones. Therefore, this article will concentrate on data-centric XML documents.

Referential Integrity in XML Documents

XML natively supports one referential integrity mechanism, which is the `ID/IDREF(S)` type of attribute linkages. In every XML document, the value of an `ID` type attribute appears at most once and the value of the `IDREF(S)` attribute must refer to one `ID` type attribute value. An `IDREF(S)` type attribute can refer to any XML element in the same document, and each XML element can define at most one `ID` type attribute. Due to the nature of `ID/IDREF(S)` type attributes in XML documents, relationships among different XML element types can be realized and it is possible to use them to implement data semantics.

This article will discuss the various data semantics and the possible ways to implement them. The algorithms presented are based on observations of the common XML document structures:

1. Due to the nested structure of an XML document (the relationship between a parent element and its child elements), the child elements implicitly refer to their parent element.
2. For an `IDREF` or `IDREFS` type attribute, the defining element is referred to the element(s) with an `ID` type attribute by the referred value. Such linkages are similar to the foreign keys in a relational database. The two associated element types are considered to be linked by an explicit linkage.
3. As an `IDREFS` type attribute can refer to more than one element, there is a one-to-many cardinality from the referring element type and the referred element type(s).

The schema of an XML document can restrict the order of the XML elements, which may be significant; the order depends on the intentions of the original XML document designer. For example,

Table 1. Two equivalent XML documents that can represent the same data

DTD	XML Document
<!ELEMENT couples (husband*,wife *,couple*)> <!ELEMENT husband EMPTY> <!ELEMENT wife EMPTY> <!ATTLIST husband hid ID #REQUIRED name CDATA #REQUIRED> <!ATTLIST wife wid ID #REQUIRED name CDATA #REQUIRED> <!ATTLIST couple hid IDREF #REQUIRED wid IDREF #REQUIRED>	<?xml version=”1.0”?> <couples> <husband hid=”A123456” name=”Peter”/> <husband hid=”B234567” name=”John”/> <wife wid=”X123456” name=”Amy”/> <wife wid=”Y234567” name=”Bonnie”/> <couple hid=”A123456” wid=”X123456”/> <couple hid=”B234567” wid=”Y234567”/> </couples>
<!ELEMENT couples (husband,wife)*> <!ELEMENT husband EMPTY> <!ELEMENT wife EMPTY> <!ATTLIST husband hid ID #REQUIRED name CDATA #REQUIRED> <!ATTLIST wife wid ID #REQUIRED name CDATA #REQUIRED>	<?xml version=”1.0”?> <couples> <husband hid=”A123456” name=”Peter”/> <wife wid=”X123456” name=”Amy”/> <husband hid=”B234567” name=”John”/> <wife wid=”Y234567” name=”Bonnie”/> </couples>

two XML documents with their corresponding DTDs are shown in Table 1.

The two XML documents shown in Table 1 are storing the same data, which are the data of two couples. For the first one, its couple elements use the two `IDREF` type attributes to denote the corresponding `husband` and `wife` elements. However, the use of `ID/IDREF` cannot ensure that a particular `husband` or `wife` element is referred by one `couple` element only. For the second XML document, the DTD restricts the `husband` and `wife` elements to exist as a pair. Furthermore, the use of `ID` type attributes `hid` and `wid` ensures any husband and wife element instance must exist in the document at most once.

Extended DTD Graph

As XML element instances are treated as individual entities, the relationships of the element

types are therefore related not only to the structure of the XML document, but also to the linkages of the different types. As such, DTD cannot clearly indicate the relationships.

An extended DTD graph for XML is proposed to add data semantics into a DTD graph so that the data semantics can be clearly identified, which is an excellent way of presenting the structure of an XML document. As such, in order to visualize the data semantics determined based on the XML document with its optional schema, it will provide the notations to be used for presenting the various data semantics. This article uses the authors' notations of the extended DTD graph for presenting the structure and the data semantics of the elements:

1. The vertexes as squares are drawn on the graph for elements, and vertexes as circles are drawn for the occurrence operators (?, +, and *) and selection operator (|).
2. Attributes and simple elements are omitted from the graph as they specify a particular attribute of their defining and parent elements, respectively.
3. Data semantics, other than one-to-one and one-to-many cardinality relations, are presented in the graph as arrows pointing from the referring element to the referred element with suitable descriptions as legends.

Based on the above criteria, it is possible to consider the ELEMENT declarations only for constructing the extended DTD graph. Three types of ELEMENT declarations can be identified as follows:

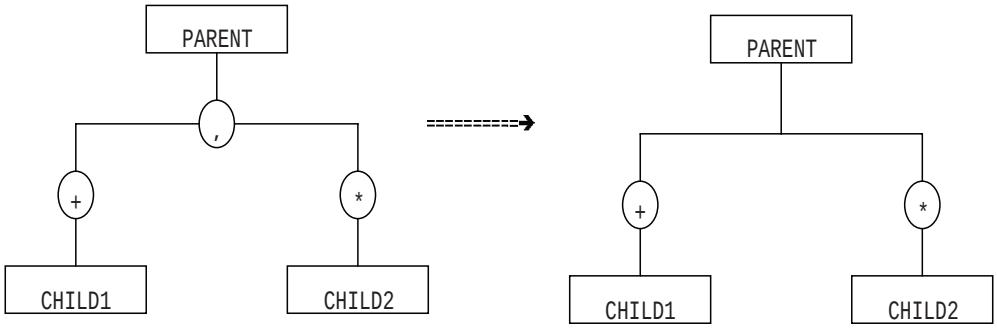
1. An ELEMENT declaration defines subelements only.
2. An ELEMENT declaration involves subelements and #PCDATA as its contents.
3. An ELEMENT declaration defines #PCDATA as its contents only.

The above three types correspond to the following three examples.

```
<!ELEMENT PARENT (CHILD1+, CHILD2*)>
<!ELEMENT MIXED_ELEMENT (#PCDATA | CHILD1
    | CHILD2)*>
<!ELEMENT SIMPLE_ELEMENT (#PCDATA)>
```

For each ELEMENT declaration of the first type, the content model expression can be tokenized as individual elements, occurrence indicators, and sequence separators (,), and represented as a tree structure with the element name as the root node. For example, the first example above can be visualized as a tree diagram. In Figure 1, the sequence (,) is implied in the diagram.

Figure 1. A sample extended DTD graph



DTDs mostly contain more than one `ELEMENT` declaration, but each element type can only appear once. Therefore, to construct the complete DTD graph for a DTD, the tree structures of all `ELEMENT` declarations in a DTD are constructed first and are eventually merged by replacing each subelement node in a tree with the tree structure of that element. Such merging is repeated until there is only one tree structure or all subelements have been replaced with their corresponding tree structures.

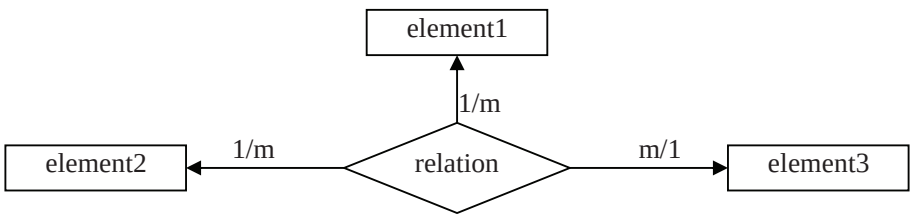
Cardinality and Participation

Element types are visualized as rectangles in the graph and a cardinality relationship is presented as an arrow pointing from the referring element type to the referred element type, with a double line and single line for total participation and partial participation, respectively. The cardinality types, including one to one (`1/1`), one to many (`1/m`), many to one (`m/1`), and many to many (`m/m`), are shown as legends of the arrows. If the cardinality relationship is implemented as explicit `ID/IDREF(S)` linkages, the name of the `ID` type attribute of the referring element is appended to the legend, such as `1/m (parent_id)`. To identify explicit linkages from implicit linkages, cardinality relationships due to `ID/IDREF(S)` type attributes are shown as arrows with curved lines. Table 2 presents the eight possible combinations of arrows and legends.

Table 2. The arrows illustrating various cardinalities with participation types

Participation / Cardinality		
One to one	1/1	1/1
One to many	1/m	1/m
Many to one	m/1	m/1
Many to many	m/m	m/m

Figure 2. A sample diagram with an n-ary relationship



N-ary Relationship

An *n*-ary relationship is implemented as a particular element type involved in more than two binary relationships. To represent such a relationship, a diamond-shaped vertex is used for such element types. Figure 2 presents a sample diagram with an *n*-ary relationship.

Aggregation Relationship

An aggregation relationship denotes that the involved element types must exist as a unity. In Figure 2, an aggregation exists as the defining characteristic of mandatory participation between parent and child elements. As such, a rectangle is to be drawn enclosing all involved element types.

RELATED WORK

In order to have a complete picture of the reasons behind the algorithms for determining various data semantics, this article explains the existing approaches of constructing XML documents, especially those exported from relational databases.

The Determination of XML Schema

There is some existing work concerning the extraction of schema, such as DTD, from XML documents (Chidlovskii, 2001; Min, Ahn, & Chung, 2003). The outputs of these algorithms are the schemas that can validate the XML documents. However, the derived schemas provide no semantic interpretation other than the containment structures of the XML documents. The algorithms proposed in this article concern the determination of data semantics from the XML element instances rather than simply XML schema among XML elements. Compared to the approach proposed by Goldman and Widom (1997) that directly manipulates semistructured databases, such as an XML document, the algorithm proposed here provides the user with a clear picture of the data semantics from the XML element instances before further manipulating them.

The Determination of Data Semantics from XML Documents

One approach exists that can reverse engineer data semantics from XML documents (Fong & Wong, 2004), but the algorithm maps some predefined templates of document structures to data semantics, and the algorithm can only be implemented with DOM (World Wide Web Consortium [W3C], 2003), which needs to read the entire XML document to the memory and therefore is inappropriate for huge documents. The methodology presented in this article, however, determines basic candidate data semantics from arbitrary XML documents with SAX (W3C, 2004), which is applicable to XML documents of any size. Some of the determined data semantics may not be the intentions of the original writer and need user supervision for verification.

The Implementation of Inheritance among XML Elements

Schema for object-oriented XML (SOX; W3C, 2005) introduced the idea of element and attribute inheritance, which enables an element to extend another element so that the derived element can have all attributes defined by the base element with its own new attributes.

Due to the limitations and low extensibility of DTD (Sahuguet, 2000), XML schema definition (XSD; Sperberg, 2000) is becoming the popular replacement schema of DTD. Unlike DTD, XSD is an XML document itself and can define more restrictive constraints and clear definitions of the XML documents to be validated. In other words, the set of capabilities for defining the structures and data types of XSD are the superset of that

of DTD. As such, there has been research and software for converting DTD to XSD (Mello & Heuser, 2001; W3C, 2000).

There are other alternative schemas, such as RELAX NG (*RELAX NG*, 2003) and Schematron (Shanmugasundaram et al., 2008). Lee and Chu (2000a) evaluated six common XML schemas, including DTD and XSD. As the others are not as popular as DTD and XSD, they are not discussed in this article.

By constructing a graph by placing vertexes for elements—and the elements that are involved in a parent-child relation, which is defined by `ELEMENT` declaration in DTD, are connected with edges—it is possible to derive a graphical representation of the DTD that is commonly known as a DTD graph. Up to now, there has been no formal standard for DTD graphs, and various researchers are using their own conventions, as in Klettke, Schneider, and Heuer (2002), Shanmugasundaram et al. (2008), Lu, Sun, Atay, and Fotouhi (2003), and Böttcher and Steinmetz (2003); the graph introduced in Funderburk, Kiernan, Shanmugasundaram, Shekita, and Wei (2002) is the first one that was denoted as a DTD graph.

There is a graphical representation of XSD (Fong and Cheung, 2005) that derives an XML conceptual schema of an XML tree model from an XML schema of XSD. Its approach is different from this article's approach, which derives an extended DTD graph from an XML document.

As the conventions of most graphs for presenting the structure of an XML document are applicable to different schema languages, the graph is also known as a semantic graph (An, Borgida, & Mylopoulos, 2005). Some researchers proposed other graphical representations of XML schemas, such as the use of UML (unified modeling language; Booch, Christerson, Fuchs, & Koistinen, 1999).

The Application of Extended DTD Graphs

A data graph is a DTD in a graph. Zhao and Siau (2007) described DTD as a good common data model when the majority of data sources are XML sources for the interoperability between relational databases and XML databases. Reverse engineering an XML document into a DTD graph

Table 3. A comparison between the proposed and other existing approaches

	Proposed Approach	**Other Approaches**
Input	XML document with optional schema	XML document
Output	Conceptual schema with data semantics	Schema without data semantics
Completeness	All common data semantics can be determined.	Schemas that can validate the XML document can be derived.
User friendliness	Algorithms can be implemented with a user-friendly GUI (graphical user interface), such as the prototype.	Commercial products exist that provide a user-friendly GUI.
Performance	Good	Not available as no mathematical proofs were provided

is similar to data mining an XML document into a data tree (Zhang, Liu, Ling, Buckner, & Tija, 2006). The former is a database schema while the latter is internal data in tree structure. Trujillo and Luján-Mora (2004) demonstrated that a DTD can be used to define the correct structure and content of an XML document representing the main conceptual multidimensional model for data warehouses.

Compared to the approach proposed by Goldman and Widom (1997) that directly manipulates semistructured databases such as an XML document, the algorithm proposed in this article enables the user to have a clear picture of the data semantics from the XML element instances before further manipulating them. Table 3 provides a comparison between the proposed algorithms and other existing approaches.

REVERSE-ENGINEERING METHODOLOGY

There are basically two different definitions in a DTD: `ELEMENT` and `ATTLIST`. Each `ATTLIST` definition defines the attributes of a particular element, whereas `ELEMENT` defines its possible containments. Each `ELEMENT` definition can be represented in a tree structure with the element name as the root and its child subelements as leaves; there must be another `ELEMENT` definition for each of the child elements.

It is not mandatory to define the `ELEMENT` declaration prior to all its child elements, and it is actually uncertain which element is the root element of the corresponding XML documents. The root element of the XML document is defined by the `DOCTYPE` declaration before the root-element start tag.

Implementations of Various Data Semantics in XML

The following subsections provide all possible implementations of various data semantics, some of which are consistent with those proposed by other researchers (Lee, 2000; Lee, Mani, & Chu, 2003).

Cardinalities

One-to-many cardinalities can be realized by both explicit and implicit referential linkages. Through implicit referential linkages, a parent element can have child elements of the same type, such as

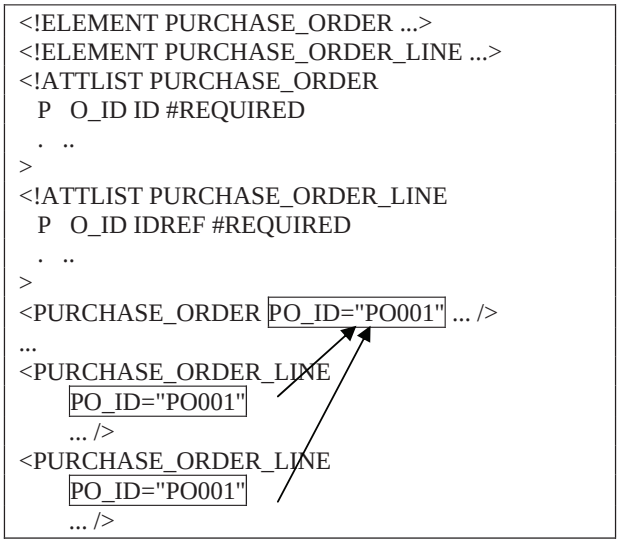
```
<PURCHASE_ORDER>
  <PURCHASE_ORDER_LINE .../>
  <PURCHASE_ORDER_LINE .../>
</PURCHASE_ORDER>.
```

The parent element `PURCHASE_ORDER` and the child elements `PURCHASE_ORDER_LINE` are implicitly in a one-to-many relationship. If the occurrences of child element `PURCHASE_ORDER_LINE` are at most one for all `PURCHASE_ORDER` elements, they are in a one-to-one relationship instead.

If the schema of the XML document is given, it can specify the `ID/IDREF(S)` type attributes. If an XML element defines an `IDREF` attribute and all such elements refer to the same element type, there is a one-to-many relationship between the referred and referring XML elements. For example, sample DTD and XML documents are shown in Figure 3.

For explicit referential linkages, to determine if the cardinality is one to one or one to many, it is necessary to scan the entire XML document. An XML element type may be involved in more than one one-to-many relationship. In other words, all elements of such XML element types define more than one linkage. For example, if an XML element type defines an `IDREF(S)` type attribute,

Figure 3. A many-to-one cardinality implemented by an IDREF type attribute



all elements of such XML element type actually define two linkages: one implicit linkage by the nested structure and one explicit linkage by the `IDREF(S)` type attribute. If the two linkages are both one-to-many relationships, the two referred element types by such a referring element type can be considered to be in a many-to-many relationship. For example, the XML document in Figure 4 illustrates a many-to-many relationship.

For an XML element type that defines two linkages and hence two one-to-many relationships, the two referred XML element types can be considered to be in a many-to-many relationship.

The linkages from the XML elements in an XML document are identified by the referring element name, linkage name, and the referred element name. The algorithm shown in Figure 6 is used to determine the linkages in Table 4.

Figure 5 illustrates the meanings of the four attributes.

There are eight XML elements in the document and there is only one implicit linkage from them. The values of the above four linkage attributes for such implicit linkage are given in Table 5.

According to the combination of the values of the four attributes, it is possible to determine the cardinality data semantics for the involved elements. The rules are shown in Table 6.

The algorithm is composed of two passes of parsing of the same XML document. The first pass assigns a synthetic element identity to each XML element in the document and determines all `ID` type attribute values and their corresponding element types. For the second pass, the XML document is traversed again and the linkages of each XML element are investigated and their attributes are stored. Finally, the stored linkage attributes are consolidated to give the four linkage attributes mentioned above and in Table 4.

The algorithm shown in Figure 6 can determine whether the XML document is valid, in particular,

Figure 4. A many-to-many cardinality implemented by an element type with two IDREF type attributes

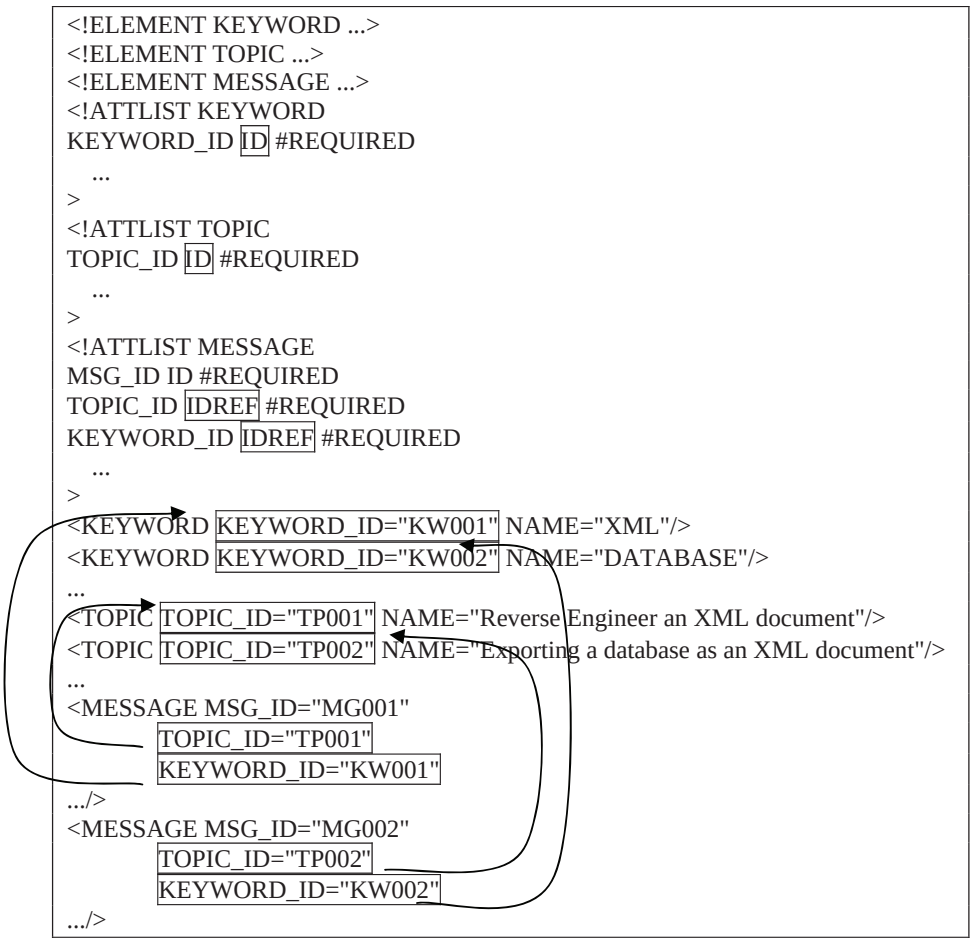


Table 4. The attributes and their sources for determining data semantics

Attribute	Description	Value
MaxReferring	The maximum number of referred elements referred by a single referring element	Get from ReferringInfo with key (RGE, RDE, L)
MaxReferred	The maximum number of referring elements that are referring to the same referred element with the same linkage type	Get from ReferredInfo with key (RGE, RDE, L)
SumReferring	The number of referring elements that possess the linkage	Get from ReferringInfo with key (RGE, RDE, L)
NumberElements	The number of referring elements in the document	Get from ElementNameCount with key RGE

Figure 5. MaxReferring, MaxReferred, SumReferring, and NumberElements example

```
<?xml version="1.0"?>
<message id="ID1" ... >
  <message id="ID2" ... >
    <message id="ID3" ... >
    <message id="ID4" ... >
    <message id="ID5" ... >
  </message>
  <message id="ID6" ... >
    <message id="ID7" ... >
    <message id="ID8" ... >
  </message>
</message>
```

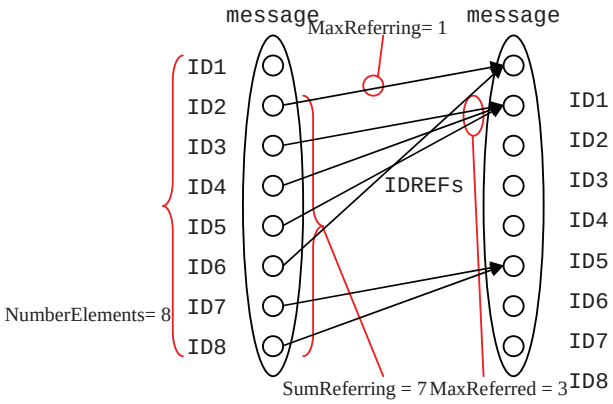


Table 5. Descriptions of variables in reverse-engineering algorithms

Attribute Name	Value	Explanations
MaxReferring	1	All linkages are implicit and each child element has one implicit parent element only.
MaxReferred	3	The root message element with attribute ID value ID1 is referred by two subelements (with attribute ID values ID2 and ID6). The message element with attribute ID value ID2 is referred by three subelements (with attribute ID values ID3, ID4, and ID5). The message element with attribute ID value ID6 is referred by two subelements (with attribute ID values ID7 and ID8). Therefore, the value of MND is 3.
SumReferring	7	Except the root message element with attribute ID value ID1, all other message elements define such linkages. The value of NL is therefore 7.
NumberElements	8	There are eight message elements.

whether a nonexisting ID value is referred by an IDREF(S) type attribute. If the XML document is valid, three tables can be obtained: ReferringInfo, ReferredInfo, and ElementNameCount. The key for the former two tables is the composite key (RGE, RDE, L), that is, the referring element name, the referred element name, and the linkage name, whereas the key for ElementNameCount is

Table 6. Matrix for determining cardinality and participation based on the determined linkage attributes

		Participation	
		Total	Partial
Cardinality	One to one	MaxReferring= 1 MaxReferred = 1 SumReferring= Number-Elements	MaxReferring = 1 MaxReferred= 1 SumReferring < Number-Elements
	One to many	MaxReferring = 1 MaxReferred > 1 SumReferring = Number-Elements	MaxReferring = 1 MaxReferred > 1 SumReferring < Number-Elements
	Many to one	MaxReferring > 1 MaxReferred = 1 SumReferring = Number-Elements	MaxReferring > 1 MaxReferred = 1 SumReferring < Number-Elements
	Many to many	MaxReferring > 1 MaxReferred > 1 SumReferring = Number-Elements	MaxReferring > 1 MaxReferred > 1 SumReferring < Number-Elements

Figure 6. The algorithm for determining linkage information by traversing the XML document

Variable name	Definition
EID	The current element ID. While processing the XML document sequentially, the EID determines the ID to be assigned to individual element encountered.
E	The current element to be handled.
A	An attribute of the current element to be handled.
AV	The attribute value of attribute *A*.
L	A linkage of the current element. It can be an implicit linkage with its parent element or an explicit linkage with an IDREF(S) type attribute. For a non-root element without IDREF(S) attribute, the element has only one implicit linkage to its parent element. Otherwise, the element can have more than one linkage, one implicit linkage and at least one explicit linkages.
L_{value}	The Element ID of the linkage *L* for the current element *E*. For example, if *L* is an implicit linkage, L_{value} is the element ID of the parent element of *E*. Otherwise, L_{value} is the attribute value of IDREF value and the value should be an ID type attribute of an element in the same document.
NG	The number of referring element of the same element name is referring to the same referred element with the same link.
RGE	The referring element of a link.
RDE	The referred element by a link.

continued on following page

Figure 6. continued

```
Pass One:
Let EID = 1;
Repeat until all XML document elements are read
    Let E be the current element to be processed
    If ∃ record in Table_ElementNameCount where ElementName = element name of E
        Get record (ElementName, NumberElement) from Table_ElementNameCount
        Increment NumberElement by 1;
        Update (ElementName, NumberElement) into Table_ElementNameCount;
    Else
        Add (ElementName, 1) into Set_ElementNameCount;
    End If
    Add (EID, ElementName) into Set_ElementIDName;
    If there exists ID type attribute A of element E with attribute value AV
        Add (AV, ElementName) into Set_ElementIDName;
    End If
    Increment EID by 1;
    Navigate to the next element E in the XML document

Pass Two:
Repeat until all XML document elements are read
    Let RGE is the current element to be handled
    For each linkage, L, of RGE
        For each linkage value, L_value of linkage L of RGE
            Get record (EID,ElementName) from Table_ElementIDName
                    where primary key value is L_value
            If no such record exist in Table_ElementIDName
                XML document is invalid
            Else
                Let RDE = ElementName of the record obtained from Table_ElementIDName
            End If

            Get record (RGE, RDE, L, L_value, ND) from Table_RawReferredInfo
                    for primary key (RGE, RDE, L, L_value);
            If record exists
                Increment ND of the record by 1;
                Update the record to Table_RawReferredInfo;
            Else
                Add record (RGE, RDE, L, L_value, 1) to the Table_RawReferredInfo;
            End If

            For each referred element type, RDE
                Let NG = number of RDE referred by this linkage, L;
                Get record (RGE, RDE, L,MaxReferring, SumReferring)
                        from the Table_ReferringInfo for primary key (RGE, RDE, L);
                If record exists
                        If NG > MaxReferring from the record
                            Update MaxReferring of the record to be NG
                        End If
                        Increment SumReferring of the record by 1;
                        Upda    te the record to the Table_ReferringInfo;
                Else
                        Add record (RGE, RDE, L, NG, 1) to the Table_ReferringInfo;
                End If
            End For
        End For
    End For
    Navigator to the next element RGE in the XML document

Consolidate the records with same combination of (RGE, RDE, L) in table RawReferredInfo;

 let MaxReferred = maximum of the ND values of all records;
 Add record (RGE, RDE, L, MaxReferred) to the table ReferredInfo;
```

simply the element name. With three such tables, it is possible to derive the linkage attributes as shown in Table 4.

The complete algorithm is presented in Figure 6, which is followed by a list of definitions for the variables to be used.

The operation in Figure 6 can be represented by the following in SQL (structured query language):

```
SELECT
    RGE, RDE, L,
    ReferringInfo.MaxReferring,
    ReferredInfo.MaxReferred,
    ReferringInfo.SumReferring,
    ElementNameCount.NumberElements
FROM
    ReferringInfo
    INNER JOIN ReferredInfo
    ON ReferringInfo.RGE = ReferredInfo.
    RGE
    AND ReferringInfo.RDE = ReferredInfo.
    RDE
    AND ReferringInfo.L = ReferredInfo.L
    INNER JOIN ElementNameCount
    ON ReferringInfo.RGE = ElementNam-
    eCount.E.
```

Once the four attributes of a linkage are determined, the data semantics can be determined by using the matrix shown in Table 6. According to the determined one-to-one and one-to-many relationships, it is then possible to consolidate the related ones into many-to-many and *n*-ary relationships.

As mentioned above, if an XML element type defines two linkages that are determined to be many-to-one cardinalities, the two referred XML element types are considered to be in a many-to-many relationship. Similarly, if an XML element type defines more than two linkages that are determined to be many-to-one cardinalities, the referred XML element types are considered to be in an *n*-ary relationship. Therefore, based on the one-to-many cardinalities determined by the previous algorithm, the many-to-many and *n*-ary relationships can be determined, and the algorithm is shown in Figure 7.

The many-to-one relationship to be considered should be those implemented by explicit linkages, that is, those defined by `ID/IDREF(S)` linkages. Otherwise, an element type exhibits implicitly a one-to-many relationship due to the nested structure and defines a many-to-one relationship that will be considered to be a many-to-many relationship even though the two referred elements are actually not related at all.

Participation

Participation concerns whether all instances of a particular element type are involved in a relationship with the corresponding element type.

Figure 7. The algorithm for determining many-to-many and n-ary relationships

```
Get referring XML element types from one-to-many cardinalities;
For each referring XML element T_referring type
        Get referred XML element types, S_referred referred by T_referring via explicit linkages;
    If the size of the set S_referred = 2
        XML element types in S_referred = many-to-many relationship with T_referring;
    Else
        If size of S_referred > 2
    XML element types in S_referred = n-ary relationship with T_referring ;
```

For implicit referential linkage by a parent-child relation, such as the following DTD ELEMENT declaration,

```
<!ELEMENT PARENT (CHILD*)>,
```

where there are no other ELEMENT declarations that define CHILD as their child elements, all CHILD element instances must appear as the child element of a PARENT element, and hence the participation can be considered to be total as all instances of CHILD must be involved in the one-to-many cardinality relation with PARENT. If no schema is provided, and if all instances of an element type always appear as the child elements of the same parent element type, the participation is also considered to be total.

For explicit referential linkage by ID/IDREF(S) attributes, if all instances of an element type use the same attribute with values referring instances of the same element type, the relationship is considered to be total participation. Otherwise, the relation is considered to be partial. The DTD of the XML document can only identify the ID/IDREF(S) type attributes and cannot restrict the referring and referred element types. As such, actually parsing the XML document is required to determine the type of participation.

Aggregation

An aggregation means that the creation of a whole part of an element depends on the existence of its component subelements. An aggregation relation is signified by the scenario that elements of different types are considered to be a single entity and all constituting elements must exist together. An XML document by itself does not provide any facility to enforce such a constraint. At best, the schema can hint at the correlations of the existence of the elements in the corresponding XML document.

Implicit referential linkage by an aggregation relationship is shown in the following DTD ELEMENT declaration:

```
<!ELEMENT AGGREGATION (COMPONENT1, COMPONENT2,….COMPONENTN)+>.
```

For example, the following ELEMENT declaration can restrict the existence of the elements enrollment, student, and course:

```
<!ELEMENT enrollment (student, course)+>.
```

Furthermore, no student or course elements exist in the document that are not subelements of an enrollment element. For example, if there is another ELEMENT declaration in the same DTD, such as

```
<!ELEMENT student _ list (student*)>,
```

student elements can exist in the document as the subelements of a student _ list element. As such, the coexistence relationship of enrollment, student, and course elements no longer holds.

Such a coexistence relationship specified in the schema can be extended to more than one nested level. For example, if the existence of a course element must be accompanied by a lecturer element and a tutor element, that is,

```
<!ELEMENT course (lecturer, tutor)+>,
```

then the elements enrollment, student, course, lecturer, and tutor must exist as a whole. All these elements are considered as being in an aggregation relationship. From another perspective, an aggregation relationship is actually composed of two one-to-one cardinality relations (course-lecturer and course-tutor) that are both total participation.

An exceptional case is if the subelements are actually the attribute of the parent element, such

as in the first example, where it is inappropriate to consider that the involved elements are in an aggregation relationship. As a result, user supervision is needed in the process.

Based on the DTD of the XML document, it is possible to determine the aggregation relationships from the elements. As the requirements of an aggregation relationship are the coexistence of the involved elements and the insignificance of the order of the subelements for a parent element, the nested structure of the elements should first be simplified with the algorithm presented in Figure 8, where T is an aggregation tree.

The determination of aggregation relationships is separated into two parts. The first part discovers the pair of parent and child elements that must coexist. Once the pairs are determined, the second part of the algorithm treats each pair as a path from the parent element to the child element in a subtree, and these subtrees are merged to form a bigger tree. Eventually, the nodes in each tree must coexist in an aggregation relationship. The second part is straightforward except for a tricky point: If a child element is found to be a nonroot node of a particular subtree, it implies that such an element can have more than one parent element, and the aggregation relation that includes such element must start with the parent element.

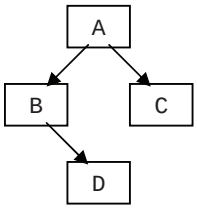
An example is this list of `ELEMENT` declarations in the DTD:

Figure 8. The algorithm for determining aggregation relationships

```
Let Set_temporary = empty;
For each ELEMENT declaration for element E_parent
    For each child element, element_child
        If element_child = mandatory and non-repeatable
            Add an aggregation relation (E_parent, E_child) to Set_temporary;

Let Set_aggregation and Set_root = empty;
For each relation R (E_parent, E_child) in Set_temporary
    If (∃ tree, T, in Set_aggregation) ∧ (E_parent is a node in T) ∧ (E_child is not a node in T)
        Add a path E_parent to E_child to T;
    Else
        (∃ tree, T, in Set_aggregation) ∧ (E_child is a node of T) ∧ (E_parent is not a node)
        If (E_child = root node) ∧ (E_child not in Set_root of T)
            A dd the path E_parent to E_child to T;
        Else
            Add E_childto Set_root
            Remove the sub-tree starting with E_child from T;
            If ∃ sub-tree starting with E_child in multiple nodes
                Add sub-tree to Set_aggregation;
            Else
                ∃ tree T_i with a node for E_parent and T_j with E_child as root node;

        Merge trees T_i and T_j with a path from node for E_parent in T_i to root of T_j
    Else
        ¬∃ sub-tree in Set_aggregation with node for either E_parent and E_child;
        Add a new tree with a path E_parent to E_child to Set_aggregation;
```

```
<!ELEMENT A (B, C)>
<!ELEMENT B (D)>
<!ELEMENT C (D)>
<!ELEMENT D (E, F)>.
```

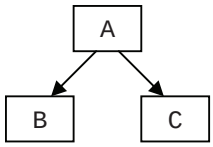
The determined pairs of raw aggregation relations are (A, B), (A, C), (B, D), (C, D), (D, E), and (D, F).



Relations to be added

(C, D), (D, E) and (D, F)

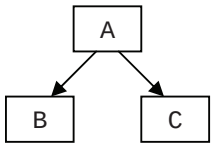
When adding the path (C, D) to the subtree, as D is not a root node, D should be removed from the subtree and is considered to be an individual subtree with D as the single node.

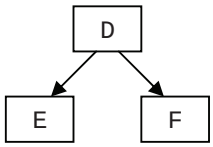


Relations to be added

(D, E) and (D, F)

After the path (D, E) and (D, F) is added to the subtree with node D as the root node, two subtrees are obtained.





As such, the elements A, B, and C and the elements D, E, and F are considered as being two individual aggregation relationships.

Figure 9. test.xml and test.dtd

```
<?xml version="1.0"?>
<test>
   <element1 id="id1"/>
   <element1 id="id2"/>
   <element2 id="id3"/>
   <element2 id="id4"/>
   <element3 id="id5" idref1="id1" idref2="id3"/>
   <element3 id="id6" idref1="id2" idref2="id4"/>
   <element3 id="id7" idref1="id1" idref2="id4"/>
   <element3 id="id8" idref1="id2" idref2="id3"/>
</test>
```

```
<!ELEMENT test (element1*,element2*,element3*)>
<!ELEMENT element1 EMPTY>
<!ELEMENT element2 EMPTY>
<!ELEMENT element3 EMPTY>

<!ATTLIST element1
        id ID #REQUIRED>
<!ATTLIST element2
        id ID #REQUIRED>
<!ATTLIST element3
        id ID #REQUIRED
        idref1 IDREF #REQUIRED
        idref2 IDREF #REQUIRED>
```

CASE STUDY AND PROTOTYPE

To illustrate the applicability and correctness of the algorithms mentioned in this article, a prototype was built that implements the algorithms proposed. For actually drawing the DTD graph, the algorithm proposed by Shiren, Xiujun, Zhongshi, and Bing (2001) is used to define the layout of the vertexes on the graph. With such a prototype, a sample XML document with its DTD file as shown in Figure 9 is provided for the prototype.

For this case study, both `ID/IDREF` type attributes are considered and the minimum number of common attributes is one. All elements with at least one attribute are sorted in ascending order of the lengths of their attribute lists. Therefore, the order of the elements to be processed is element1, element2, and element3.

According to the DTD of the XML document, only one `ELEMENT` declaration is used for constructing the extended DTD graph as the contents of other element types are `EMPTY`.

Figure 10. The determined data semantics

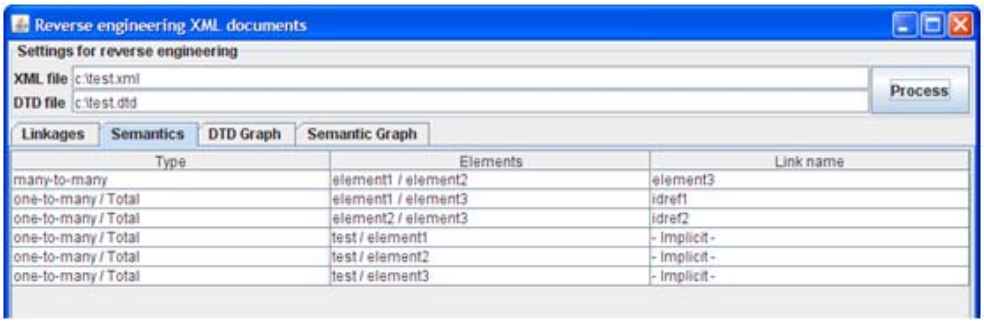
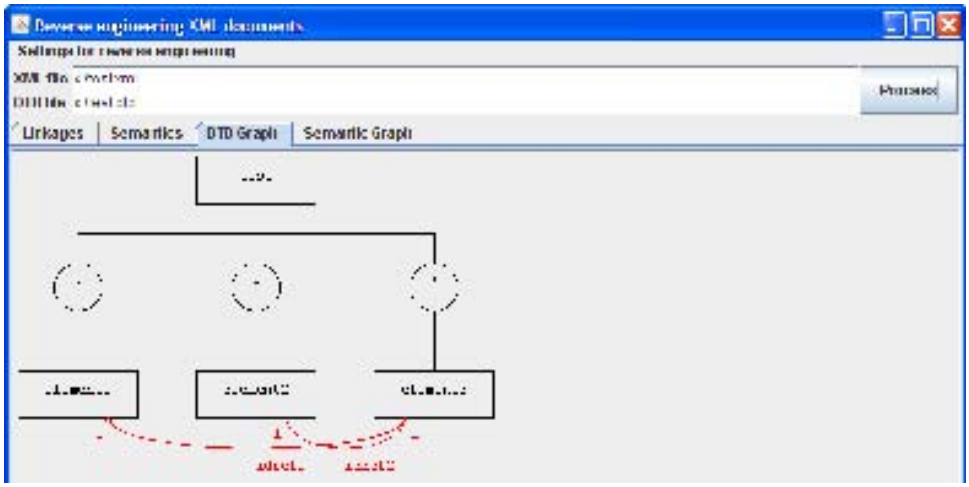


Figure 11. Extended DTD graph based on the DTD and the determined cardinality references



```
<!ELEMENT test (element1*,
    element2*,element3*)>
```

Therefore, only those explicit one-to-many relationships are to be added to the graph, and the graph will become the one shown in Figures 10 and 11. The detailed derivation of the reverse engineering can be referred to in Shiu (2006).

CONCLUSION

In order to make use of an XML document, software developers and end users must have a thorough understanding of the contents in the document, especially in historical and large XML documents. Sometimes, the schemas of XML documents are missing and the documents cannot be opened to be inspected on the screen due to their huge size. Therefore, it is necessary to determine as much information as possible regarding the relationships from the elements in the document.

By reverse engineering the XML document with DTD, all explicit linkages can be determined, and the resultant DTD graph can be used to verify the correctness of `ID/IDREF(S)` linkages as any incorrect `IDREF(S)` linkage will be indicated as an extra cardinality and shown in the extended DTD graph. This article provides algorithms to help the users to understand the relationships from the elements by reverse engineering data semantics from the XML document, including the following.

1. Cardinality relationships
2. Participation relationships
3. *n*-ary relationships
4. Aggregation relationships
5. Many-to-many relationships (a special case of cardinality relationships)

In summary, to visualize the determined data semantics, a new extended DTD graph is proposed. XML documents natively support one-to-one, one-to-many, and participation data semantics. With a corresponding schema such as DTD, the `ID` and `IDREFS` attributes of the elements can be identified, and many-to-many, *n*-ary, and aggregation relationships can also be determined.

ACKNOWLEDGMENT

This article is funded by Strategic Research Grant No. 7002325 of City University of Hong Kong.

REFERENCES

An, Y., Borgida, A., & Mylopoulos, J. (2005). Constructing complex semantic mappings between XML data and ontologies. In *International Semantic Web Conference ISWC 2005* (pp. 6-20).

Booch, G., Christerson, M., Fuchs, M., & Koistinen, J. (1999). *UML for XML schema mapping specification.* Retrieved from http://xml.coverpages.org/fuchs-uml_xmlschema33.pdf

Bosak, J., Bray, T., Connolly, D., Maler, E., Nicol, G., Sperberg-McQueen, C. M., et al. (1998). *Guide to the W3C XML specification (XMLspec) DTD, version 2.1.* Retrieved from http://www.w3.org/XML/1998/06/xmlspec-report-v21.htm

Böttcher, S., & Steinmetz, R. (2003). A DTD graph based XPath query subsumption test. In *Xsym 2003* (pp. 85-99).

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2004). *Extensible markup language (XML) 1.0* (3rd ed.). Retrieved from http://www.w3.org/TR/2004/REC-xml-20040204

Chidlovskii, B. (2001). Schema extraction from XML data: A grammatical inference approach. In *KRDB'01 Workshop (Knowledge Representation and Databases).*

Deutsch, A., Fernandez, M., & Suciu, D. (1999). *Storing semi-structured data with STORED*. Paper presented at the SIGMOD Conference, Philadelphia.

Fernandez, M., Morishima, A., & Suciu, D. (2001). Publishing relational data in XML: The SilkRoute approach. *IEEE Data Engineering Bulletin, 24*(2), 12-19.

Florescu, D., & Kossmann, D. (1999). Storing and querying XML data using an RDBMS. *Bulletin of the Technical Committee on Data Engineering, 22*(3), 27-34.

Fong, J., & Cheung, S. K. (2005). Translating relational schema into XML schema definition with data semantic preservation and XSD graph. *Information and Software Technology, 47*(7), 437-462.

Fong, J., & Wong, H. K. (2004). XTOPO: An XML-based technology for information highway on the Internet. *Journal of Database Management, 15*(3), 18-44.

Funderburk, J. E., Kiernan, G., Shanmugasundaram, J., Shekita, E., & Wei, C. (2002). XTABLES: Bridging relational technology and XML. *IBM Systems Journal, 41*(4).

Goldman, R., & Widom, J. (1997). DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases.*

Kanne, C. C., & Moerkotte, G. (2000). Efficient storage of XML data. In *Proceedings of ICDE* (p. 198).

Kay, M. (1999). *DTDGenerator: A tool to generate XML DTDs.* Retrieved from http://users.breathe.com/mhkay/saxon/dtdgen.html

Klettke, M., Schneider, L., & Heuer, A. (2002). Metrics for XML document collections. In A. Chaudri & R. Unland (Eds.), *XMLDM Workshop*, Prague, Czech Republic (pp. 162-176).

Koike, Y. (2001). A conversion tool from DTD to XML schema. Retrieved from http://www.w3.org/2000/04/schema_hack

Lee, D. W., & Chu, W. W. (2000a). Comparative analysis of six XML schema languages. *SIGMOD Records, 29*(3).

Lee, D. W., & Chu, W. W. (2000b). Constraints-preserving transformation from {XML} document type definition to relational schema. In *International Conference on Conceptual Modeling: The Entity Relationship Approach* (pp 323-338).

Lee, D. W., Mani, M., & Chu, W. W. (2003). Schema conversion methods between XML and relational models. In *Knowledge Transformation for the Semantic Web.*

Lu, S., Sun, Y., Atay, M., & Fotouhi, F. (2003). A new inlining algorithm for mapping XML DTDs to relational schemas. In *Proceedings of the First International Workshop on XML Schema and Data Management, in conjunction with the 22nd ACM International Conference on Conceptual Modeling (ER2003).*

Mello, R., & Heuser, C. (2001). A rule-based conversion of a {DTD} to a conceptual schema. In *Lecture notes in computer science* (Vol. 2224).

Min, J. K., Ahn, J. Y., & Chung, C. W. (2003). Efficient extraction of schemas for XML documents. *Information Processing Letters, 85*(1).

Moh, C., Lim, E., & Ng, W. (2000). DTD-Miner: A tool for mining DTD from XML documents. In *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce.*

RELAX NG. (2003). Retrieved from http://www.relaxng.org

Sahuguet, A. (2000). Everything you ever wanted to know about DTDs, but were afraid to ask. In *WebDB-2000*.

Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, E., Naughton, J., et al. (2008). *Schematron*. Retrieved from http://www.schematron.com

Shiren, Y., Xiujun, G., Zhongzhi, S., & Bing, W. (2001). Tree's drawing algorithm and visualizing method. In *CAD/Graphics'2001.*

Shiu, H. (2006). *Reverse engineering data semantics from arbitrary XML document.* Unpublished master's thesis, City University of Hong Kong, Hong Kong, China.

Sperberg-McQueen, C., & Thompson, H. (2000). *W3C XML schema*. Retrieved from http://www.w3.org/XML/Schema

Stayton, B. (2008). *DocBook*. Retrieved from http://www.docbook.org

Tatarinov, I. (2001). A general technique for querying XML documents using a relational database system. *SIGMOD Record, 30*(3), 261-270.

Thiran, P. H., Estiévenart, F., Hainaut, J. L., & Houben, G. J. (2004). Exporting databases in XML: A conceptual and generic approach. In *Proceedings of CAiSE Workshops (WISM'04).*

Trujillo, J., & Luján-Mora, S. (2004). Applying UML and XML for designing and interchanging information for data warehouses and OLAP applications. *Journal of Database Management, 15*(1), 41-72.

World Wide Web Consortium (W3C). (1998). *Schema for object-oriented XML*. Retrieved from http://www.w3.org/TR/1998/NOTE-SOX-19980930

World Wide Web Consortium (W3C). (2003). *Document object model DOM*. Retrieved from http://www.w3.org/DOM

World Wide Web Consortium (W3C). (2004). *Simple API for XML, SAX*. Retrieved from http://www.saxproject.org

Zhang, J., Liu, H., Ling, T., Bruckner, R., & Tija, A. (2006). A framework for efficient association rule mining in XML data. *Journal of Database Management, 17*(3), 19-40.

Zhao, L., & Siau, K. (2007). Information mediation using metamodels: An approach using XML and common warehouse metamodel. *Journal of Database Management, 18*(3), 69-82.

This work was previously published in the Journal of Database Management, edited by K. Siau, Volume 19, Issue 4, pp. 62-80, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 3.12
LOGIC–Minimiser:
A Software Tool to Enhance Teaching and Learning Minimization of Boolean Expressions

Nurul I. Sarkar
Auckland University of Technology, New Zealand

Khaleel I. Petrus
University of Southern Queensland, Australia

ABSTRACT

Boolean algebra, minimization of Boolean expressions, and logic gates are often included as subjects in electronics, computer science, information technology, and engineering courses as computer hardware and digital systems are a fundamental component of IT systems today. We believe that students learn minimization of Boolean expressions better if they are given interactive practical learning activities that illustrate theoretical concepts. This chapter describes the development and use of a software tool (named LOGIC-Minimiser) as an aid to enhance teaching and learning minimization of Boolean expressions.

LEARNING OBJECTIVES

After completing this chapter, you will be able to:

- List and describe three main features of LOGIC-Minimiser.
- Explain how LOGIC-Minimiser can be used in the classroom to enhance teaching and learning Boolean expression minimization.
- Describe the Q-M algorithm for the minimization of Boolean expressions.
- Define the following key terms: Boolean expression, SOP, logic gate, logic minimization, and K-maps.

INTRODUCTION

It is often difficult to motivate students to learn minimization of Boolean expressions because students find the subject rather abstract and technical. A software tool (named LOGIC-Minimiser) has been developed that gives students a hands-on learning experience in minimizing Boolean expressions. LOGIC-Minimiser was developed in C language under MS Windows and is suitable for classroom use in introductory Boolean algebra courses. Based on user input (i.e., logic expression), the system displays the sum of product (SOP) functions as well as minimized logic gate diagrams. Test results demonstrate the successful implementation of LOGIC-Minimiser, and the simplicity of the user interface makes it a useful teaching and learning tool for both students and instructors.

This chapter describes the development of LOGIC-Minimiser and its usefulness as an aid to teaching and learning minimization of Boolean expressions. The chapter concludes with a discussion of the strengths and weaknesses of LOGIC-Minimiser and its future development.

BACKGROUND AND MOTIVATION

Boolean algebra, minimization of Boolean expressions, and logic gates are essential concepts included in electronics, computer science, information technology, and engineering. These concepts play a fundamental role in computer hardware and digital systems design. We believe that it is extremely important to incorporate practical demonstrations into these courses to illustrate theoretical concepts and therefore provide an opportunity for hands-on experience. These demonstrations will significantly enhance student learning about Boolean expression minimization.

In fact, very little material has been designed and made available for public access to supplement the teaching of Boolean expression minimization. This is revealed by searches of the Computer Science Teaching Center Web site (http://www.cstc.org/) and the SIGCSE Education Links page (http://sigcse.org/topics/) on the Special Interest Group on Computer Science Education Web site. We strongly believe, as do many others (Bem & Petelczyc, 2003; Hacker & Sitte, 2004; Ibbett, 2002; Leva, 2003; Shelburne, 2003; Williams, Klenke, & Aylor, 2003), that students learn more effectively from courses that provide for active involvement in hands-on learning activities.

Boolean expression minimization is one of the most challenging subjects to teach and learn in a meaningful way because students find the topic full of technical jargon, dry in delivery, and quite boring. Sarkar, Petrus, and Hossain (2001) have developed LOGIC-Minimiser in C under MS Windows to give students an interactive, hands-on learning experience in minimization of Boolean expressions. LOGIC-Minimiser can be used by a teacher in the classroom as a demonstration to enhance the traditional lecture environment at an introductory level. Also, students can use the system in completing tutorials on Boolean expression minimization and to verify (interactively and visually) the results of in-class tasks and exercises on Boolean expression minimization. LOGIC-Minimiser can be used either in the classroom or at home as an aid to enhance teaching and learning Boolean expression minimization.

Minimization of Boolean expressions using traditional methods such as truth tables, Boolean algebra, and K-maps can be very tedious and is not well-suited for expressions involving more than six variables. A more useful approach, the Quine-McCluskey (Q-M) algorithm, also called tabular method, is an attractive solution for minimizing complex Boolean expressions involving variables of any length. Moreover, the algorithm lends itself to a fast and easy machine implementation.

The remainder of the chapter is organized as follows. First we examine various open source software tools suitable for logic-gate design and minimization. We then describe LOGIC-Mini-

miser in teaching and learning contexts. Then, software implementation of LOGIC-Minimiser is discussed, and the educational benefits of the software are highlighted. An example of a classroom plan and LOGIC-Minimiser in practice is discussed. Test results which verify the successful implementation of LOGIC-Minimiser are presented, followed by a conclusion and future research directions.

RELATED WORK

A detailed discussion of digital systems design and minimization of Boolean expressions in general can be found in Green (1985), Greenfield (1977), Mano (1984), and Tanenbaum (1999). The Quine-McCluskey algorithm is described extensively in the computer hardware and digital logic design literature (Carothers, 2003; Costa, 2004; Hideout, 2003; Hintz, 2003). Grimsey (2000) examined the strengths and weaknesses of various methods of minimizing Boolean expressions, including truth tables, Boolean algebra, and Karnaugh maps (K-maps).

A variety of open source and commercial software tools exist for modelling and simulation of logic circuit design and Boolean expression minimization. These powerful tools can have steep learning curves; while they may be good for doing in-depth performance modelling of computer hardware and logic design, they often simulate a hardware environment in far more detail than is necessary for a simple introduction to the subject.

Lockwood (2003) presented a program for the implementation of the Q-M algorithm. However, it is of limited use as a teaching and learning tool because of its text-based interface that is not user-friendly. Leathrum (2003) described another text-based menu-driven program for the Q-M algorithm, but the user interface is rather difficult to use. Costa (2004) developed a package called "bfunc" for Boolean functions minimization. It is an MS-DOS-based program and is considered an alternative to the K-map method of simplifying Boolean functions. Burch (2002) proposed a tool named Logisim, a graphical system for logic circuit design and simulation which is suitable for classroom use. Logisim is a Java application and can be run on both Windows and Unix workstations. While Logisim is an excellent tool for building a variety of complex combinational circuits, but it is not suitable for logic gate minimization. Other tools such as Digital Works 3.0 (2001) and LogicWorks (1999) are similar to Logisim in that they provide a graphical toolbox interface for composing and simulating logic circuits. LOGIC-Minimiser, which we describe in the next section, has its own unique features, including simplicity and ease of use either in the classroom or at home, to enhance teaching and learning Boolean expression minimization.

ARCHITECTURE OF LOGIC-MINIMISER

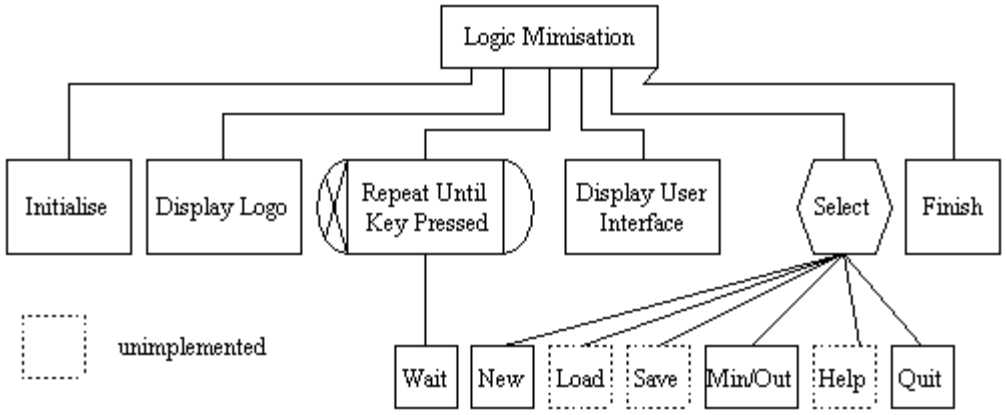
Figure 1 shows the structured diagram of LOGIC-Minimiser. The main features of LOGIC-Minimiser are briefly described.

- **New**: This feature allows users to enter a new set of variables for minimization.
- **Min/Out**: This feature allows users to view a minimized sum of product (SOP) expression and logic circuit diagram.
- **Quit**: This feature allows users to exit from the program at any time.

The following three features have not been implemented yet and are considered as future work.

- **Load**: This feature will allow users to view existing data (i.e., minimized minterms) for further analysis and modifications.

Figure 1. Structured diagram of LOGIC-Minimiser



- **Save**: This feature will allow users to store outputs on disk for later use and further modifications.
- **Help**: This feature will provide help on various topic related to minimization of Boolean expressions.

SOFTWARE IMPLEMENTATION

The Q-M algorithm is used to reduce a Boolean expression to its simplest form. It is designed particularly for use with problems containing six variables or more but can be used equally well for a smaller number of variables. The algorithm is based on repeated applications of the distributed law and the fact that XOR (NOT X) is always true. The Q-M method is a systematic way of selecting the pairs to be used for simplification. The main steps in the Q-M algorithm are summarized below:

1. Representing all addends as sums of minterms
2. Grouping the minterms that have the same number of ones
3. Merging the terms that differ in only one bit (this is done in several steps)
4. In order to find the irredundant cover we use the min-cover algorithm:
   a. Find all distinct minterms.
   b. Find all essential prime implicants.
   c. Find all the minterms that are covered by the essential prime implicants.
   d. Remove all minterms and prime implicants found in (a)-(c).
   e. Choose that prime implicant that covers most of the remaining minterms.
   f. Repeat (d) until all minterms have been covered.

A structured analysis and design has been employed to design the package. C programming language under MS Windows has been used in the implementation.

USEFULNESS AND BENEFITS OF LOGIC-MINIMISER

For simplicity and ease of use, it has been decided to implement LOGIC-Minimiser with a menu-driven, keyboard-based interface with a few menu options. The interface is self-explanatory, which makes the package well-suited for both students and teachers for classroom use. Therefore, the package can be an integral part of a 2-hour session for teaching and learning the Q-M method for logic gate minimization. An in-class task will be given to the students to produce a minimized logic diagram on paper. After a prescribed period of time (for example, 20 minutes), LOGIC-Minimiser will be introduced to the students on a step-by-step basis to verify their solution and learn more about minimization of Boolean expressions.

LOGIC-Minimiser provides the following main benefits:

- **Hands-on**: It facilitates an interactive, hands-on introduction to minimization of Boolean expressions.
- **Modelling**: It provides a simple and easy way to develop a variety of SOP functions and models. Students can experiment with minterms of various sizes and develop a sound knowledge and understanding of Boolean expression minimization.
- **Ease of use**: The use of a menu-driven interface makes LOGIC-Minimiser easy to use and a user-friendly tool. The software can be easily installed and run on any PC operating under MS Windows.
- **Economical/usefulness**: It enhances face-to-face teaching with online learning and can be used either in the classroom or at home to provide hands-on experience.
- **Robustness**: It was tested on various PCs across campuses and was found to be robust.
- **Challenging**: It provides an environment for students to test their knowledge on Boolean expression minimization.

EXAMPLE OF CLASSROOM PLAN

In this section we present a detailed lesson plan (2-hour session) which can be used in teaching and learning minimization of Boolean expressions using LOGIC-Minimiser. The learning outcomes focus on learning the Q-M algorithm as well as use of the software tool for verifying results of Boolean expression minimization. The lesson plan incorporates a number of resources and classroom activities, including revision of Boolean expressions, brainstorming, teaching, example, worksheet, demonstration of software package, and use of the package to verify the worksheet exercises.

Lesson Plan

Table 1 lists the learning outcomes, resources, and various activities that can be conducted in the classroom in teaching minimization of Boolean expressions effectively. It can be used for a 2-hour lecture session on the minimization of Boolean expressions. The lesson plan includes a guided worksheet (see Table 2) suitable for classroom use.

HOW TO USE THE SYSTEM

LOGIC-Minimiser is easy to use and can be run from any PC operating under MS-DOS/Windows. To run the package, the user can either double-click on “newqm.exe” or type “newqm” at the DOS prompt. The main steps of using this package (from Windows) are summarized below:

Table 1. Lesson plan (2-hour session with 10-minute break)

<table>
<tr><td colspan="2">By the end of this session students will be able to:
• Outline steps in minimization of Boolean expressions using Q-M algorithm.
• Use LOGIC-Minimiser to verify the minimization of Boolean expressions.</td></tr>
<tr><td>Resources required</td><td>• LOGIC-Minimiser
• Data Show
• Computer Laboratory
• Whiteboard
• Worksheets</td></tr>
<tr><td>Time (minutes)</td><td>Activity</td></tr>
<tr><td>10</td><td>Quickly review of Boolean expressions</td></tr>
<tr><td>5</td><td>Brain storming (ask the class what they know about Boolean expression minimization)</td></tr>
<tr><td>15</td><td>Explain Q-M algorithm</td></tr>
<tr><td>5</td><td>LOGIC-Minimiser demonstration</td></tr>
<tr><td>15</td><td>Solve workout/example problems</td></tr>
<tr><td>10</td><td>Break</td></tr>
<tr><td>20</td><td>Worksheet Exercises (ask the class to work in pairs and solve worksheet exercises)</td></tr>
<tr><td>20</td><td>Use LOGIC-Minimiser and verify results of minimization (worksheet exercises)</td></tr>
<tr><td>10</td><td>Conclusion and checking learning outcomes</td></tr>
<tr><td colspan="2">Review:
What went well:

What could be improved:</td></tr>
</table>

- **Run**: Double-click on "newqm.exe."
- **Entering minterms**: Select the New option to enter a new set of minterms. The user will be prompted for the number of variables to be used. After entering the appropriate number of variables, a matrix of cells with index numbers will appear on the screen. At this point the user can enter each minterm by selecting a cell by pressing the Enter key on the keyboard.
- **Accepting data**: When a set of minterms has been entered, press the F8 key to accept.
- **Display diagram**: The minimized logic-gate diagram can be seen on the screen in graphics mode. The user can zoom the diagram using the F1, F2, and F3 keys for 100%, 50%, and 20% scaling, respectively. To go back to main menu, press the F10 key.
- **Display minterms and output**: Select the Min/Out option from the main menu to see

Table 2. Boolean expression minimization worksheet

Consider the minimization of following logical function:

$F = A.B.C.D + A.B.C.\overline{D} + A.B.\overline{C}.D + A.\overline{B}.C.D + \overline{A}.B.C.D + \overline{A}.B.C.\overline{D}$

The above function can be written as:

$F(A,B,C,D) = \sum(0,1,2,4,8,9)$

Where 0, 1, 2, 4, 8, 9 are the decimal values of the minterms.
It is required to apply the minimization on this Boolean function with the use of Quine-McCluckey's algorithm, firstly by hand and then verify the solution using LOGIC-Minimiser.

Solution:

1. Write the first list as:

	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
4	0	1	0	0
8	1	0	0	0
9	1	0	0	1

2. Deduce the second list (You have to complete this list)

	A	B	C	D
0,1	0	0	0	-
0,2				
0,4				
0,8				
1,9				
8,9	1	0	0	-

3. Third list (You have to do it all yourself)

	A	B	C	D
0,8,1,9				

4. Now build the chart which relates minterms with the prime implicates as shown:

	0	2	4	8	9
$\overline{A}.\overline{B}.\overline{C}$	X	X			

5. Now use the LOGIC-Minimiser to cross check your solution.

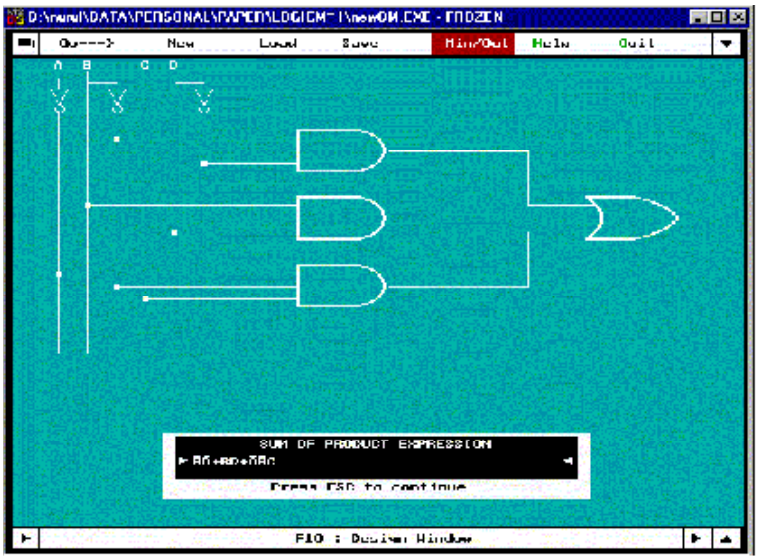
the list of minterms (that have been entered) and the minimized output expression.

- **Exit from the program**: Select the Quit option from the main menu to exit from the program at any time.

TEST RESULTS

To evaluate the performance of LOGIC-Minimiser, the software has been installed on various PCs and tested with various Boolean expres-

Figure 2. Example of four-variable minimization with minimized output expression and logic gate diagram



sions, each involving a different number of input variables. Then the test results were validated manually. Figure 2 shows a sample test result for four-variable (A, B, C, and D) Boolean expression minimization. The following minterms were entered from the keyboard: [0,2,3, 5,7,8,10,13,15], and the software produced the simplified logic-gate diagram as well as the output expression, as shown in Figure 2.

EVALUATION

An earlier version of LOGIC-Minimiser had been presented at the National Advisory Committee on Computing Qualifications conference in Napier, New Zealand (Sarkar & Petrus, 2001). The discussion during the conference presentation was quite encouraging, and many staff members from various polytechnic institutions expressed their interest in using the package in their classes.

To assess the educational value of LOGIC-Minimiser, we administered a survey to the students of the introductory digital logic subject. The survey was repeated for two consecutive years. Overall results revealed that the majority of students found the package useful and user-friendly, with an overall rating of 4 out of 5.

The questionnaire also posed five open-ended questions: (1) How well did you understand minimization of Boolean expressions before entering this course? (2) How easy did you find the software package to use? (3) How well did the package help in understanding the minimization of Boolean expressions? (4) Would you like to have more software tools of this kind as part of your course? (5) Would you prefer to learn minimization of Boolean expressions in a hybrid mode (i.e.,

minimization by hand and verification of results by software tool)?

CONCLUDING REMARKS

A software tool (LOGIC-Minimiser) has been developed that can be used in the classroom to enhance the teaching and learning of various aspects of Boolean expression minimization. LOGIC-Minimiser is easy to use and can be run from any computer operating under MS-DOS and MS Windows. It was tested on various PCs and was found to be robust. Many staff members from various polytechnic institutions expressed their interest in using the software in the classroom.

Currently, the system minimizes Boolean expressions involving variables of size 8, which is adequate for demonstration purposes. LOGIC-Minimiser can easily be upgraded to accommodate variables of any length. User options such as New, Min/Out, and Quit have been implemented. More options such as Save, Load, and Help are still under development, and incorporation of a mouse-based user interface is also suggested for future work.

LOGIC-Minimiser is available free of cost to faculty interested in using it to supplement their teaching. More information about LOGIC-Minimiser can be obtained by contacting the first author.

SUMMARY

Boolean algebra, minimization of Boolean expressions, and logic gates are essential concepts included in electronics, computer science, information technology, and engineering. These concepts play a fundamental role in computer hardware and digital systems design. We believe that it is extremely important to incorporate practical demonstrations into these courses to illustrate theoretical concepts and therefore provide an opportunity for hands-on experience. These demonstrations will significantly enhance student learning about Boolean expression minimization. This chapter described the development and use of LOGIC-Minimiser as an aid to enhance teaching and learning Boolean expression minimization. It was tested on various PCs and was found to be robust.

REVIEW QUESTIONS

1. List and describe three main features of LOGIC-Minimiser.
2. Discuss the usefulness of LOGIC-Minimiser in teaching and learning contexts.
3. Describe the main steps in the Q-M algorithm for the minimization of Boolean expressions.
4. Define the following key terms: Boolean expression, logic gate, logic, minterms, and K-maps.
5. Explain how LOGIC-Minimiser can be used in the classroom for demonstration.
6. List and describe further enhancements to LOGIC-Minimiser.

REFERENCES

Anonymous. (2006). Digital Works. Retrieved January 5, 2006, from http://www.spsu.edu/cs/faculty/bbrown/circuits/howto.html

Bem, E. Z., & Petelczyc, L. (2003, February 19-23). *MiniMIPS: A simulation project for the computer architecture laboratory.* Paper presented at the Proceedings of the 34th Technical Symposium on Computer Science Education (SIGCSE'03), Reno, NV (pp. 64-68).

Burch, C. (2002). Logisim: A graphical system for logic circuit design and simulation. *Journal*

of Educational and Resources in Computing, 2(1), 5-16.

Carothers, J. D. (2003). *Quine-McCluskey algorithm*. Retrieved September 20, 2004, from http://www.ece.arizona.edu/~csdl/474aslide4

Costa, A. (2004). *Boolean functions simplification (logic minimization).* Retrieved December 27, 2004, from http://www.dei.isep.ipp.pt/~acc/bfunc

Green, D. C. (1985). *Digital techniques and systems* (2nd ed.). Longman.

Greenfield, J. D. (1977). *Practical digital design using ICs.* Wiley.

Grimsey, G. (2000). The truth, the whole truth, and/or nothing but the truth. *Journal of Applied Computing & Information Technology, 4*(1), 42-52.

Hacker, C., & Sitte, R. (2004). Interactive teaching of elementary digital logic design with WinLogiLab. *IEEE Transactions on Education, 47*(2), 196-203.

Hideout, G. (2003). *The Quine-McCluskey method of logic reduction*. Retrieved September 20, 2004, from http://www.geekhideout.com/qmm.shtml

Hintz, K. (2003). *Quine-McCluskey method*. Retrieved September 20, from *http://www.cpe.gmu.edu/courses/ece331/lectures/331_8/sld001.htm*

Ibbett, R. N. (2002, June 24-26). *WWW visualization of computer architecture simulations.* Paper presented at the 7th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE), Aarhus, Denmark (pp. 247).

Leathrum, J. F. (2003). *Quine McCluskey tabular minimization method.* Retrieved September 20, 2004, from http://www.ece.odu.edu/~leathrum/ECE241_284/support/quine.html

Leva, A. (2003). A hands-on experimental laboratory for undergraduate courses in automatic control. *IEEE Transactions on Education, 46*(2), 263-272.

Lockwood, J. W. (2003). *Quine-McClusky algorithm; computational techniques; cygwin freeware (GPL) tools.* Retrieved September 20, 2004, from http://www.arl.wustl.edu/~lockwood/class/coe460/

LogicWorks. (1999). *Capilano Computing Systems Ltd.* Retrieved January 10, 2006, from http://www.logicworks4.com

Mano, M. (1984). *Digital design*. Prentice Hall.

Sarkar, N., & Petrus, K. (2001, July 2-5). *Logic gate minimization demonstration.* Paper presented at the 14th annual conference of the National Advisory Committee on Computing Qualifications (NACCQ), Napier, New Zealand (p. 456).

Sarkar, N., Petrus, K., & Hossain, H. (2001, July 2-5). *Software implementation of the Quine-McCluskey algorithm for logic gate minimization*. Paper presented at the 14th annual conference of the National Advisory Committee on Computing Qualifications (NACCQ), Napier, New Zealand (pp. 375-378).

Shelburne, B. (2003). *Teaching computer organization using a PDP-8 simulator.* Paper presented at the SIGCSE'03 Technical Symposium on Computer Science Education (pp. 69-73).

Tanenbaum, A. S. (1999). *Structured computer organization* (4th ed.). Prentice Hall.

Williams, R. D., Klenke, R. H., & Aylor, J. H. (2003). Teaching computer design using virtual prototyping. *IEEE Transactions on Education, 46*(2), 296-301.

KEY TERMS AND DEFINITIONS

Boolean Expression: An expression which results in a Boolean (binary or TRUE/FALSE) value. For example 4 > 3 is a Boolean expression. All expressions that contain relational operators like >, <, and so forth are Boolean. Logical gates and their combinations are used to implement physical representations of Boolean expressions.

K-Maps: This term refers to Karnaugh maps, a logical minimization method based on graphical representation of Boolean functions in which each row in the truth table of the Boolean function is represented as a box. Unlike the truth table, K- map values of input must be ordered such that the values of adjacent columns vary by one single bit.

Logic Gate: An electronic device (based on transistors) used for implementing logical functions. The inputs and outputs of the gate are Boolean (i.e., binary) values. Gates can be used to implement various Boolean functions. NOT gates take one input and have one output. The AND, NAND, OR, and NOR gates may take two or more inputs and have one output. XOR gates take two inputs and have one output. Logical functions are all combinational functions, that is, their output depends on the input. Gates can also be used to implement latches and flip-flops which have an internal state and are used to implement sequential logical systems.

Logic Minimization: Simplification of Boolean expressions with the aim of reducing the number of logical gates. This is done by reducing the number of minterms into a number of prime implicants in which as many variables as possible are eliminated. The tabular method makes repeated use of the rule $\bar{A} + A = 1$.

LOGIC-Minimiser: Software package developed at the Auckland University of Technology to enhance teaching and learning minimization of Boolean expressions. The package was implemented in C programming language.

Minterms: This term refers to the product of Boolean variables. These variables can appear either as themselves or their inverses. A minterm corresponds to exactly one row in the truth table of the Boolean function. If we have four variables A, B, C, and D, then a minterm can be something like A.B.C.D or .B.C.D, and so forth.

Quine-McCluskey Algorithm: Table-based reduction method for simplification of Boolean expressions. This method is quite versatile as compared with other algorithms. It can handle any number of inputs and can easily be implemented on machines. The method starts from the truth table of the Boolean function.

Sum of Product (SOP): A two-level expression which represents a sum of minterms of a logical function. It is two-level because it is implemented by two layers of logic gates. The first level represents the product of Boolean variables of the logical function and the second level represents summing the products with OR operator.

This work was previously published in Tools for Teaching Computer Networking and Hardware Concepts, edited by N. Sarker, pp. 303-318, copyright 2006 by Information Science Publishing (an imprint of IGI Global).

Chapter 3.13
Assisting Learners to Dynamically Adjust Learning Processes through Software Agents

Weidong Pan
University of Technology, Sydney, Australia

Igor Hawrysiewycz
University of Technology, Sydney, Australia

ABSTRACT

To make online learning more productive, software agent technology has been applied to provide services for learners in order to assist them to construct knowledge in constructivist ways. This paper is focused on the application of software agents in assisting learners to dynamically adjust learning processes. Unlike pedagogical agents, the agents in this application do not hold domain knowledge but simply assist learners to get through learning processes by a variety of supportive services. They assist learners to develop personalized preferred learning plans and to guide them to dynamically adjust learning toward their goals. In this article, the online learning process is first investigated, and an approach to assisting learners to dynamically adjust learning is outlined. Then, the structure of the UOL (unit of learning) database that provides links between a practical learning scenario and the required services is explored. A multi-agent architecture for realizing the services is configured, and the roles of the involved agents are described. After that, the related agent algorithms for guiding learners to dynamically adjust learning are described.

INTRODUCTION

Constructivist learning is being recognized by more and more people as a productive learning method. Although there are diverse constructivist paradigms, they share commonly epistemologi-

cal assumptions for learning (Fosnot, 1996). The fundamental epistemological assumption is that knowledge cannot be transmitted to learners but must be individually constructed and socially co-constructed by learners (Jonassen, 1999). Because constructivist learning focuses on actively constructing meaningful understandings of the study theme, it can generate more significant outcomes than other methods such as the objectivist ones (Wilson, Teslow, & Osman-Jourchoux, 1995).

According to constructivist theories for learning, learners are *active* knowledge-constructors, whereas teachers are cognitive guides who provide guidance and scaffolds to support the construction (Mayer, 1999). Unfortunately, most current online instructional systems have not really taken such roles. Mostly, they just simply deliver online course materials over the Internet without providing effective guidance on how to use these materials to construct knowledge. As a result, learners only passively receive information from the presented materials. They have not been engaged in *actively* constructing meaningful understandings of the study theme. This research is aimed by applying software agents into online learning to *actively* assist learners to construct knowledge by using constructivist methods.

The research into software agents has been a rapidly developing area of research. Already a lot of agent-based systems have been proposed, ranging from comparatively small systems such as e-mail filters to large, complex, mission-critical systems such as air-traffic control (Jennings, Sycara, & Wooldridge, 1998). In particular, pedagogical agents have been developed to take the role of a virtual tutor, a virtual learning partner, and so forth. The agents we are developing facilitate online learning through comprehensive applications of the properties agents exhibit (e.g., autonomy, learning, cooperation, reactivity, goal-driven, etc.). They work together cooperatively in order to facilitate effective knowledge construction for individual learners. They assist learners to construct knowledge not through understanding the academic content of subjects but rather through providing a wide range of services. These services include (1) providing access to appropriate learning resources and learning strategies; (2) fostering meaningful interactions with content, teachers, and fellow learners; (3) supporting personalized learning for individual learners; (4) promoting collaborative learning among learners in groups; and (5) aiding to evaluate learning achievements in a timely and accurate manner (Pan & Hawryszkiewycz, 2004a).

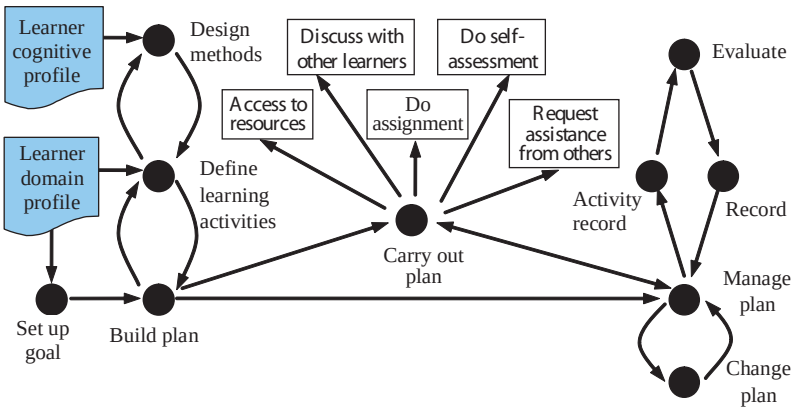
This article is focused on showing how software agents are applied to assist learners to dynamically adjust learning processes. This includes (1) guiding them to develop personalized preferred learning plans that satisfy their learning needs and that match their particular learning styles; and (2) assisting them in the alignment of learning plans according to the real progress in learning. The purpose of such services is to engage learners in knowledge construction and to promote its success through providing guidance for learners to solve the problems pertaining to learning strategies that they often encounter in online learning.

DYNAMICALLY ADJUSTING THE LEARNING PROCESSES

Online Learning and Learning Plans

Online learning takes place in many environments rather than just at educational institutions. Learners in education institutions usually follow a particular instruction program. Other learners, however, build their knowledge through a continuous and guided process of identifying learning project goals, discussing and trying ideas, and evaluating learning outputs. Such a process, as shown in Figure 1, is initiated and driven by a learning goal. After a goal is constituted based

Figure 1. An online learning process



on a project or a case study, learners go through a guided process in order to reach it. The first step is to *build a plan* in order to achieve the goal. This includes defining the learning activities to be taken and designing the methods to conduct these activities based on their particular cognitive features and learning history. Then, learners *carry out the plan* to construct their own understanding of the study theme. They follow the learning steps in sequence, as defined in the plan. Each learning step is a particular learning activity (e.g., accessing learning resources, discussing with others, doing assignments, doing self-assessments, requesting assistance from others, etc.). As the learning proceeds, learners *manage the plan* in order to align learning toward their goal. They record the learning activities that they conducted, evaluate their outcomes, and then revise the current plan based on the evaluation. The updated plan immediately will affect the learning process; the relevant activities or sequences will be aligned. The learning, based on the updated plan, will be evaluated again that further results in a plan revision. An online learning process evolves in this way, until the evaluation shows that the learning goal has been achieved.

General Framework for Supporting Online Learning

As previously described, the major challenges for learners to take part in online learning include (1) building an appropriate learning plan to achieve their learning goal; and (2) timely and accurately adjusting the learning plan toward the goal, based on the practical learning progress. It will significantly benefit learners to continue their pursuits for the goal, if the online instructional system can provide assistance for them to tackle these challenges. This is because *not* all learners are equally capable of adequately addressing these challenges on their own (Large, 1996). Some may lack the necessary *prior* knowledge or abilities to determine independently the needed learning activities and to choose a proper method to conduct them. Some may have no idea how to evaluate the outcomes of learning and to vary plans according to learning progress.

What degree of assistance is suitable for helping learners to deal with these problems? Is it suitable to take full control of the learning by the online instructional system? Most intelligent tutoring systems (ITS) adopt this mode. They *predesign* all learning routes for learners based on a variety of learner models, expert models, and tutorial models. Learners only can *follow* these routes precisely. The problem is that these models cannot possibly specify all possible ways in which learners may go about trying to solve a problem (Jonassen, 2000). This is because learners never learn in the same way due to their different backgrounds, interests, styles, motivations, capabilities, and so forth. It is even truer for online learning, because most online learners are adults. As a result, learners in those systems often are forced down the *preset* learning routes that do not suit them or even limit the development of their cognitive abilities.

In order to provide personalized learning experiences for individual learners and to make the learning process optimal, learners must assume responsibility for some decisions in the process (Kay, 2001). *Active learning* must be encouraged, because learners construct knowledge only by active learning (Akhras & Self, 2000). Therefore, an innovative strategy have been applied in our work where learners are encouraged to actively construct knowledge, and meanwhile, the system provides them with services to shape and to scaffold the learning process. Those services are aimed directly at solving the problems that emerge in the learning process. They are customized to the personalized needs of learners according to their unique learning characteristics.

In the proposed instructional system, learners are not imposed to take any learning activities. They thus can independently develop and explore their own learning plans for the study themes and actively construct meaningful understandings of the themes. Their autonomy in learning has been sufficiently supported and encouraged. Meanwhile, the system offers suggestions or advice for directing them to develop learning plans and to revise learning plans while they have trouble with these things. This contrasts with most current online instructional systems that just present course materials and leave learners to determine how to achieve their learning goals. This also contrasts with those ITS where learners only can *passively* follow a preset learning plan chosen by the system.

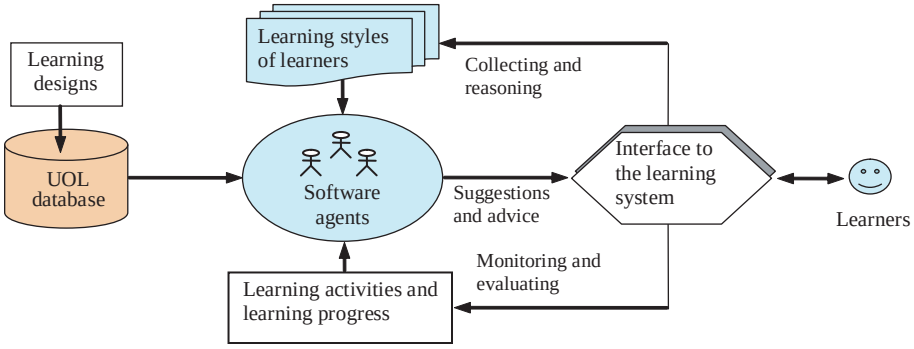
Approach for Dynamically Adjusting Learning

In order for the online instructional system to provide services to meet an individual learner's *just-in-time* needs or even *just-for-me* needs in online learning, knowledge about the learning activities being conducted, the practical learning progress, and the learner's learning styles is necessary. It thus requires combining learning content and learning evaluation with the service components together. A UOL (unit of learning) database is being used to provide the connection.

A UOL is a learning unit that satisfies one or more learning objectives. It may correspond to a course, a module, or even a single learning activity such as a discussion to elaborate on some topic. The UOL database contains a collection of UOLs, each of which is a carefully designed learning scenario where not only the learning content and evaluation methods are specified, but also the relevant learning activities, conduct sequences, and supportive services for various types of learners are defined, as well. The structure of the UOL database will be explored in detail in the next section.

Furthermore, the customized services suggest the requirements for the dynamic interaction and communication between different components in the system. Those require communications between distributed components, sensing and monitoring of the environment, and autonomous operations. The application of software agents is quite appropriate to realize these services, since

Figure 2. The overall architecture of the online learning system



software agents can act as human agents on behalf of humans. They can easily perform sequences of complex operations based on the messages they receive, their own internal beliefs, and the overall goals and objectives (Garro & Palopoli, 2002). Therefore, software agents have been developed to implement the service components. They are responsible for providing suggestions or advice according to the real learning progress and learner styles.

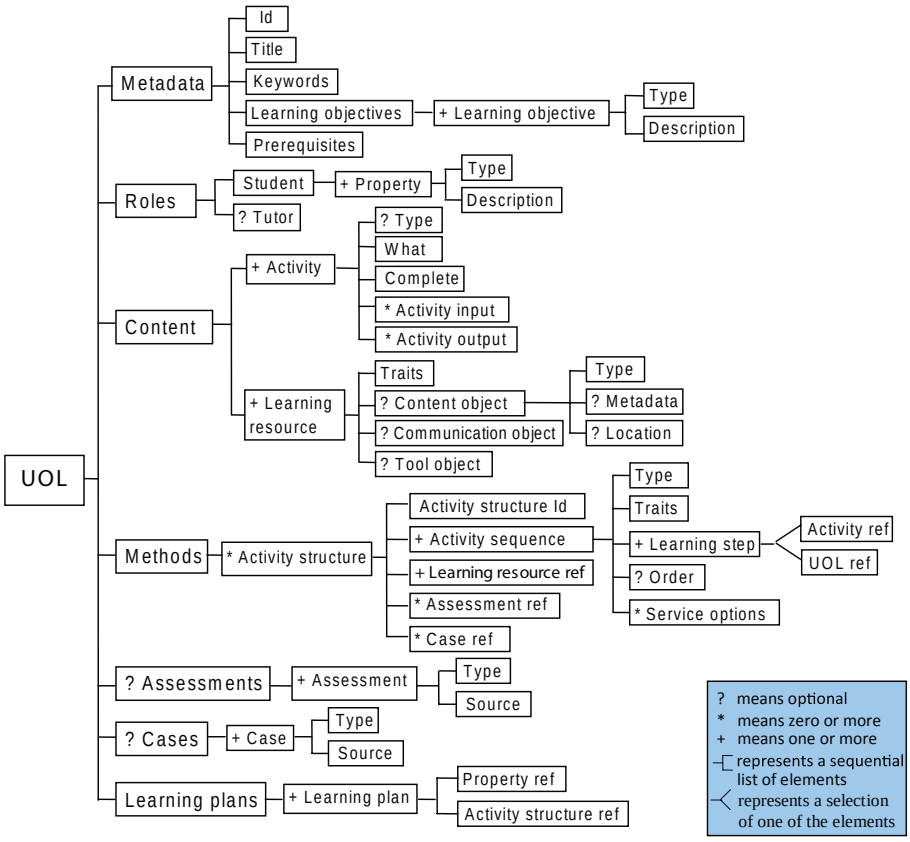
Figure 2 depicts the overall system architecture. The agents work independently of learners, observe and monitor the learning of individual learners, and make suggestions and advise them, when necessary. Any change taking place in the learning environment made by the learner is detected by the agents. The learning progress is evaluated through evaluating the detected events. The learner profiles are built and updated timely through collecting the detected events and inducing from them. The multi-agent architecture takes a practical learning scenario and the learner styles as input and generates a suggestion or advice for individual learners. The suggestion is about what the learner should do next, which mainly is based on the knowledge extracted from the UOL database by matching the input to the learning scenario in the UOL database.

THE UOL DATABASE

The purpose of developing a UOL database is to link the practical learning scenarios with the corresponding services for different learners so that agents can, based on the links, determine an appropriate service to assist individual learners to learn while a scenario appears. Building upon the earlier work in the descriptions of learning activities, particularly the educational modeling language (EML) (Koper, 2001), we are developing mechanisms to describe and specify learning processes so that individual learning can be promoted by the services provided by the agents. The primary means is to extend and to adapt the object parameters of the EML in ways that they can be combined flexibly to describe the arrangements of learning activities and the services to support personalized learning (Pan & Hawryszkiewycz, 2004b).

As shown in Figure 3, a UOL describes the learning activities for a particular objective and

Figure 3. The major components of a UOL



the corresponding supportive services using seven composite fields. Every field contains more elementary fields, constructing a complex architecture. The major fields and their roles in implementing the services to facilitate knowledge construction of learners are outlined below.

The *metadata* field is for providing the meta-information of the UOL, including the title, the prerequisites, the learning objectives, and so forth. Each objective has a brief description and a corresponding category (e.g., skill, knowledge, etc.). The *keywords* field is to store the keywords extracted from the objective descriptions that are used to match the learning goals of the learner.

The *roles* field is for specifying the intended users of the UOL. Its *property* field contains the description of the learning characteristics of every class of the intended learners, which enables the software agents to provide services according to the unique learning characteristics of individual learners.

The *content* field is for storing the description of all the learning resources and all the learning activities relating to the UOL. Every *activity* field describes a learning activity. Its *complete* field specifies the progress status of learning while the activity is completed. Its *activity output* field specifies the artifact files that the activity

will produce, which is used for evaluating the outcomes of learning. Every *learning resource* field describes a resource available for the UOL. Its *traits* field specifies the specific features of the learning resource; *content object* indicates the medium type of the resource and its exact location; *communication object* represents the requirements for the communication facilities; and *tool object* specifies the prerequisite tools and facilities for using the resource.

The *methods* field is for defining all the dynamics of the learning process in order to achieve the objectives of the UOL. They are categorized based on the learning characteristics of their targeted learners and divided into different groups accordingly. Each group is put into an *activity structure* field. This implies that an *activity structure* field stores all of the possible methods suitable for a particular category of learners to achieve the learning objectives defined in the UOL. Within an *activity structure*, there are multiple *activity sequence* fields, each of which defines a particular *learning flow* in which a sequence of learning activities is performed during the learning process. This enables the software agents to assist a learner with optional activity sequences that suit his or her learning characteristics. Each activity sequence can be associated with more than one learning resource, ensuring that an activity sequence can be conducted by using different learning resources. Each activity sequence can be associated with multiple assessment approaches and multiple related case materials, ensuring an activity sequence can be evaluated through alternative methods and be scaffolded by alternative case materials, respectively.

Within the description of a particular activity sequence, a learning step can be a reference to a learning activity defined in the UOL or a reference to another UOL. In the latter case, another UOL is referenced in order to implement a learning step of the UOL, constructing a hierarchical architecture of a module or a subject. The *type* field is to specify the pedagogical methodology category of the activity sequence (e.g., knowledge acquisition, problem-based, project-based, learning by designing, discovery learning, etc.). *Service options* declare the computer-supported collaborative tools that are required in the learning activities (e.g., learning spaces, discussion forums, etc.). *Traits* indicate the particular features of the sequence.

The *assessments* field and the *cases* field are respectively for describing the assessment approaches and the case materials, related to the UOL. The *type* specifies the category, and *source* indicates where to find it.

The *learning plans* field provides the link between a learning method and its targeted learner category. It is composed of a number of *learning plan* fields, each of which contains a pair of *property ref* and *activity structure ref*. The former is an ID of the property defined in the *roles* field, whereas the latter is an ID of the activity structure defined in the *methods* field. This enables the software agents to specify the learning methods based on the category of the learner. This field is crucial to determine dynamically a specific learning mode for individual learners and to provide associated supportive services for different learning needs.

THE MULTI-AGENT ARCHITECTURE

Agents and Their Roles

The agents that we have developed to realize the services to facilitate knowledge construction of learners are a series of individual agents with specific expertise, which forms a multi-agent architecture. In this article, we only concentrate on the agents that are involved in assisting learners to dynamically adjust their learning processes. The agents responsible for this task are described briefly as follows:

- **Personal Assistant Agent:** Helps individual learners with their learning activities. It continuously observes the behaviors of its owner learner in order to maintain a profile. The profile covers many facets that can be relevant in the learning process. The assistant agent updates the profile as the learning proceeds and provides this information to other agents when being requested.
- **Planning Agent:** An agent responsible for assisting learners to develop personalized preferred learning plans for a particular goal. Its tasks include designing learning activities and their sequences, determining evaluation methods, and so forth. All of these are accomplished based on the practical learning scenarios and the unique learning characteristics of individual learners.
- **Managing Agent:** Responsible for managing the learning for a particular UOL. Each UOL has an agent, whose task is to provide personal assistance for individual learners to learn the unit and to manage the learning based on the plan being adopted. It delegates work to learning activities according to the plan. The managing agent keeps track of the progress in the learning activity for the UOL and provides assistance to learners in revising the learning plan accordingly. It includes monitoring the submission of the artifact file for the UOL, evaluating the artifact file or asking a field expert to evaluate it and then receiving the evaluation result from the expert.

Working Modes of the Agents

The agents use a hybrid architecture that combines a reactive reasoning and a BDI-based proactive reasoning (Wooldridge, 2002). They autonomously monitor the learning environment, observe and react to the events in the environment. When a learner sets up a learning goal, the personal assistant agent for the learner perceives the event. It requests the planning agent for a learning plan for that goal. The planning agent, based on the goal and the learning characteristics of the learner provided by the assistant agent, designs learning plans for that goal and provides them to the learner for making decisions. While the learner starts learning a unit, a managing agent is created for assisting the completion of the learning activities in that unit. The managing agent manages the learning of the unit by following the plan being adopted. Based on the sequence of the learning activities scheduled in the plan, it delegates a learning activity and creates a learning space for the learning activity and an agent for it, as well. The activity agent further creates action agents to assist in conducting related actions in the activity and reports the progress to the managing agent. The managing agent delegates another learning activity after it receives the report from the activity agent on the completion of the activity with which it is associated. It reports the progress of the plan to the planning agent, and the latter revises the plan accordingly.

IMPLEMENTATION APPROACHES FOR ADJUSTING LEARNING PROCESSES

Developing Personalized Preferred Learning Plans

Assisting learners to develop personalized preferred learning plans is realized through advising them on several plans that they can follow to achieve their goals. These plans are the methods to conduct learning, including relevant learning activities to be taken and their sequence. They are extracted from the UOL database, based on the practical learning scenario and the specific learning characteristics of the learner. As shown in Figure 4, the agent first determines a UOL by matching an individual learner's goals to the objectives of a UOL in the UOL database. Then,

Figure 4. Assisting learners to develop personalized preferred learning plans

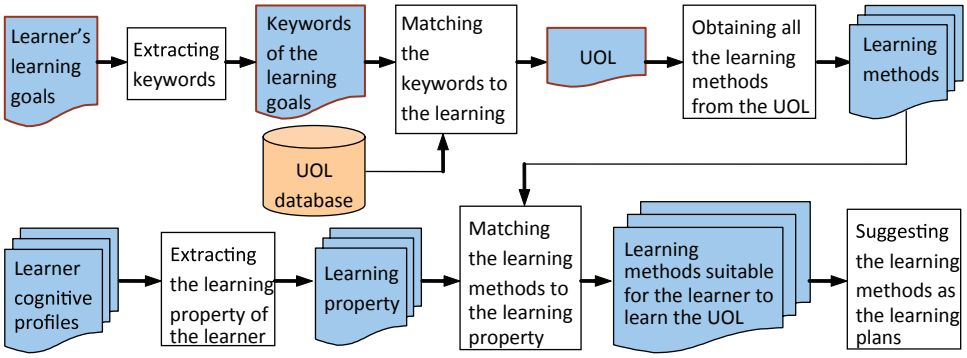
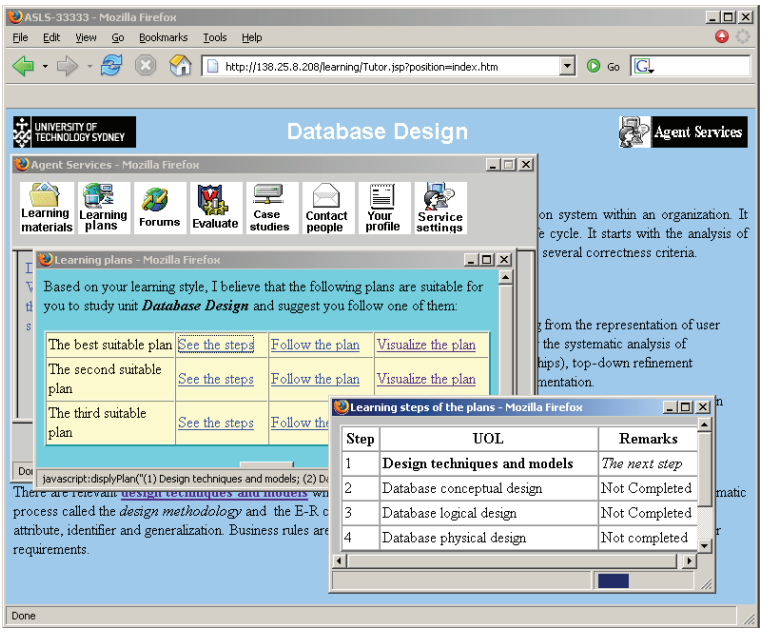


Figure 5. Presenting learners with appropriate learning plans



it captures the learning methods for the UOL from the database. Next, it further determines the ones from them that are suitable for the learner according to the fit degree of the method for him. Finally, it presents these methods as the recommended learning plans.

The following technique is used by the agent to identify if a particular learning method is suitable for a learner and to measure the fit degree. The learning property of a learner, stored in his or her cognitive profile, is characterized by a set $P = \{p_1, p_2, \ldots, p_n\}$, where p_i is one of his or her preferred styles in learning (e.g., like to study together with other learners, like to work through concrete experiences, etc.). Every learning method for a UOL stored in the UOL database has a set $M = \{m_1, m_2, \ldots, m_m\}$ to describe its traits, where m_i is a style that the method can accommodate.

The agent recognizes if a method is suitable for a learner by comparing set *M* with *P*. The fit degree is calculated by summing the numbers where the learner's favored styles are met by a method; namely, $V_{fit} = \sum (p_i \text{ in } M)$. A method is recognized as an appropriate one for a learner, if its V_{fit} is larger than a designated threshold value. A method is considered as the optimal one, if it has a larger V_{fit} than other methods.

In order to promote active learning, the agents do not force a learner to accept any of the learning plans that they have designed for him or her. Instead, they present these plans as a suggestion for the learner to make his or her own decisions. Figure 5 shows a typical scene, where the planning agent, based on the cognitive profile and domain profile of a learner, already has generated three plans for the learner to learn the *database design* unit and is presenting the three plans for the learner in a popup window. The learner is free to accept any one of them or to reject them, building his or her own plan. If the learner wants to accept a plan recommended by the agent, he or she only needs to click on *Follow the plan* in the plan line. The learner then is forwarded to the learning step defined in the plan. The learner also may want to look at the detailed learning steps of a plan before making a decision. If so, he or she can click *See the steps* in the plan line. If the learner wants to inspect and to analyze a plan in depth, he or she can click on *Visualize the plan* in the plan line. The learner will be presented with an interactive visual interface, where he or she can inspect the learning steps and view their hierarchical architecture.

Dynamically Revising Learning Plans

Guiding learners to align the learning toward their goals is implemented through managing individual learning plans. The agents perform this work with the aid of two lists; namely, activity list and checklist. The activity list stores the UOLs that a learner has started learning but has not completed yet; whereas the checklist stores the UOLs that a learner has completed. The major components of these lists are shown in Table 1 and Table 2. When a learner initiates the learning for a UOL, the UOL, its learning goal (i.e. the goal UOL), and the adopted learning plan for achieving the goal are added to the activity list. When the learner has completed the learning for a UOL, the UOL will be moved to the checklist. The two lists are updated dynamically by the agents as the learning proceeds.

The requirements for aligning learning recognized by the agents take place mainly in two scenarios: (1) when a learner starts learning a new UOL but has not completed all the UOLs planned to learn prior to the one he or she is going to study; and (2) when a learner is not able to achieve the objectives of the UOL. Figure 6 depicts the recognition of the first scenario. The agent first retrieves the learning plan from the activity list that the learner is adopting for the UOL goal. It then compares the checklist against the learning tasks scheduled in the plan to see if the checklist contains all of the UOLs planned to be learned prior to the one to be studied. If it does not, then the agent will suggest adjusting the learning plan. The second scenario is recognized by keeping track of the execution of the UOLs in the activity list and by evaluating the artifacts submitted by the learner for those UOLs. If a learner fails to submit the desired artifacts for a long time after starting a UOL, the agent will suggest adjusting the learning plan. If the evaluation of an artifact indicates that a learner is not able to achieve the objectives of a UOL, the agent also will suggest adjusting.

As to the approaches for adjusting learning, the agent usually suggests two kinds of adjustments: (1) keep the learning plans being carried out unchanged and select another UOL to learn; or (2) revise one of the plans being carried out. The agent can generate a suggestion for the first kind of adjustment by examining the current plan and the

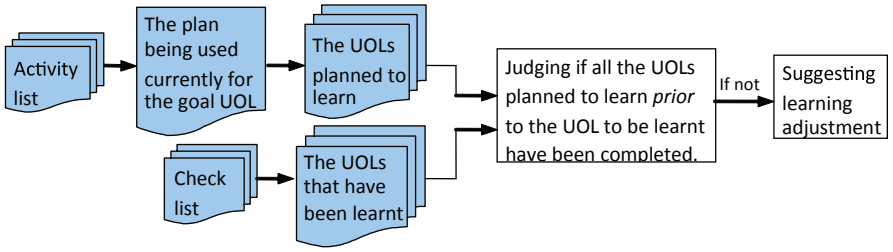
Table 1. The major components of the activity list

Field	Meaning
username	the user name of a learner
uol	the ID of a UOL the learner has started learning
goal_uol	the ID of the goal UOL for the UOL
plan	the ID of the plan for achieve the goal UOL
status	the status of learning for the UOL by using the plan, e.g. proceed or suspended
remarks	the plan is selected by agent or designed by learner

Table 2. The major components of the check list

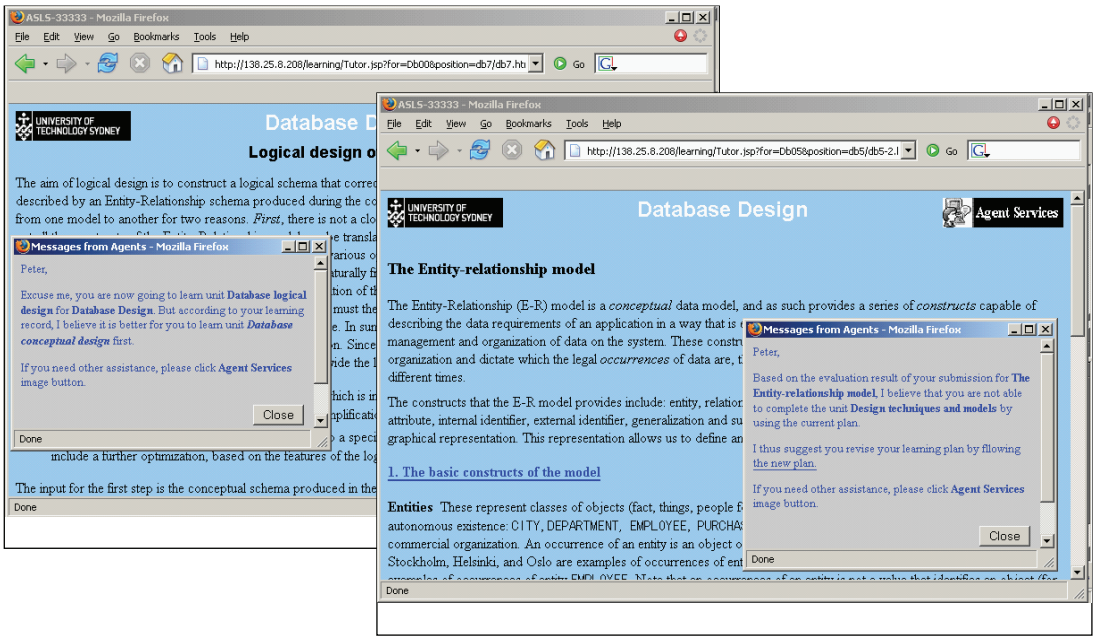
Field	Meaning
username	the user name of a learner
uol	the ID of a UOL that the learner has completed
goal_uol	the ID of the goal UOL for the UOL
remarks	the plan is selected by agent or designed by learner

Figure 6. Recognizing the requirement for learning adjustment



checklist, since it can identify the UOL that should be learned next by comparing the learning tasks scheduled in the current plan against the ones in the checklist. It is a challenging task to generate a suggestion for the second kind of adjustment, as it needs a complicated search procedure. As can be seen, a UOL with a larger grain (e.g., a subject or a chapter) may have a lot of learning paths with a complicated hierarchical structure. Accordingly, there are very complicated relationships between the plans being carried out and the ones that can be adopted after some alignment. To search for a suitable plan in the hierarchical architecture, the agent has to examine the plans from the current

Figure 7. Example suggestions by agents for learning alignment



UOL to higher level UOLs level by level. That is, the agent first checks if there is any other plan to reach the current UOL's goal, and, if not, it will further check to see if there is any possible plan suitable for the goal of the higher level UOLs. In this way, the agent checks the plans for the UOLs level by level, until it finds a suitable plan or attains another reasonable result.

Figure 7 shows two examples of the suggestions generated by the agents for learning alignment. The one on the left shows the agent suggesting that the learner study another UOL first, because it is scheduled to be learned prior to the one that the learner is going to study, based on the plan being adopted. The one on the right corresponds to changing the current learning plan, because the evaluation illustrates that the learner is not able to achieve the objectives of the UOL.

SUMMARY AND FURTHER WORK

A prototype of the multi-agent architecture for facilitating knowledge construction of learners has been developed, and the services related to assisting learners to dynamically adjust learning processes have been implemented. This research has illustrated that software agents are effective in assisting learners to dynamically adjust learning processes. They can efficiently solve the problems that learners face in online learning and can promote deeper cognitive engagement for them. Our future work involves further refining and extending of the UOL database and the agent services so that they can assist learners in adjusting learning process in different ways. An empirical validation of the system also would be carried out in order to demonstrate the usefulness of the system to a wide audience.

REFERENCES

Akhras, F. N., & Self, J. A. (2000). System intelligence in constructivist learning. *International Journal of Artificial Intelligence in Education, 11.* 348-376.

Fosnot, C. (1996). *Constructivism: Theory, perspectives, and practice*. New York: Teachers College Press.

Garro, A., & Palopoli, L. (2002). An XML multi-agent system for e-learning and skill management. *Proceedings of the Third International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002)*, Erfurt, Thuringia, Germany (pp. 8-10).

IMS. (2004). *IMS learning design specification V1.0*. Retrieved October 4, 2004, from http://www.imsglobal.org/learningdesign/index.cfm

Jennings, N., Sycara, K., & Wooldridge, M. (1998) A roadmap of agent research and development. *Autonomous Agents and Multi-Agent System, 1*, 7-38.

Jonassen, D. (1999). Constructivist learning environments on the Web: Engaging students in meaningful learning. *Proceedings of the Educational Technology Conference (EdTech 99)*, Singapore.

Jonassen, D. (2000). Computer as mindtools for schools: Engaging critical thinking. Columbus, OH: Prentice-Hall.

Kay, J. (2001). Learner control. *User Modeling and User-Adapted Interaction, 11*(1). 111-127.

Koper, R. (2001). *Modeling units of study from a pedagogical perspective: The pedagogical model behind EML*. Retrieved October 12, 2004, from http://eml.ou.nl

Large, A. (1996). Hypertext instructional programs and learner control: A research review. *Education for Information, 14*, 95-105.

Mayer, R. E. (1999). Designing instruction for constructivist learning. In C. M. Reigeluth (Ed.), *Instructional design theories and models: A new paradigm of instructional theory* (pp. 141-159). Mahwah, NJ: Lawrence Erlbaum Associates, Publishers.

Pan, W., & Hawryszkiewycz, I. (2004a). To develop constructivist learning environments on the Web using software agent technology. *Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2004)*, Kauai, Hawaii (pp. 236-241).

Pan, W., & Hawryszkiewycz, I. (2004b). A method of defining learning processes. *Proceedings of the 21st ASCILITE Conference*, Perth, Australia, 734-742.

Wilson, B. G., Teslow, J., & Osman-Jourchoux, R. (1995). The impact of constructivism (and postmodernism) on instructional design fundamentals. In B. B. Seels (Ed.), *Instructional design fundamentals: A review and reconsideration*. Englewood Cliffs, NJ: Educational Technology Publications.

Wooldridge, M. (2002). An introduction to multiagent systems. John Wiley & Sons.

This work was previously published in the International Journal of Intelligent Information Technologies, edited by V. Sugumaran, Volume 2, Issue 2, pp. 1-15, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 3.14
Integrating Software Engineering and Costing Aspects within Project Management Tools

Roy Gelbard
Bar-Ilan University, Israel

Jeffrey Kantor
Bar-Ilan University, Israel, & University of Windsor, Canada

Liran Edelist
Bar-Ilan University, Israel

INTRODUCTION

Currently, there is no integration among CASE tools (computer aided software engineering, also named AMD tools, analysis modeling and design), costing tools, and project management (PM) tools. Not only are there no integrated tools, but there is also no conceptual integration among software engineering (SE) aspects and accounting-costing aspects of software projects within PM tools. PM tools, as well as costing tools are used not only for tracking and controlling an ongoing software project, but also at the very beginning stages of the project, in which critical estimations concerning budget and time frame are made. In order to have a firm, robust, and accurate planning, project planning should be based directly upon raw SE components-objects, that is, upon analysis and design components-objects.

According to the Standish Group CHAOS Report 2003, each year in the USA there are approximately 175,000 projects in IT Application Development which spends $250 billion. Among these, 31.1% of projects will be cancelled, 52.7% of projects will cost 189% of their original estimates, only 52% of required features and functions make

it to the released product, and Time overruns 82%. In financial terms $55 Billion dollars is wasted in these projects (Madpat, 2005).

Budget overrun indicates cost management problems, although this area is defined by the project management integration (PMI), as one of the nine core activities of projects management. Costing difficulties result from both implementation limitations of costing solutions in complex and changing requirements as well as the technological environment. Risk management is also defined by the PMI as one of the nine core areas of project management; but there is also no integration between PM tools and SE tools in light of the need for risk management.

According to Maciaszek and Liong (2005), success of a software project depends on five software engineering areas that are related to each other: the development of the life cycle of the software, processes management, the model's configuration and language, and SE tools and project planning. The combining between formal tools of SE and PM processes in the different stages has been proved by research as holding a positive contribution to the efficacy of the project and as an improver of the adherence to costs, technical requirements, and the schedules that were allocated to the project (Barker & Verma, 2003).

This study proposes and prototypes a model that integrates these three aspects of software projects by automatically mapping SE objects and accounting–costing objects into PM objects. To validate the feasibility of the model and without loss of generality, it is demonstrated using former research platform focused on conversion of data flow diagrams (DFD), which are actually full enterprise set of use cases diagrams reflecting entire system-software project into Gantt charts.

BACKGROUND

CASE and PM Tools

CASE/AMD tools support the analysis, design, construction, and implementation stages of the information system life cycle (ISLC) (Barker & Longman, 1992; Pendharkar, Subramanian, & Rodger, 2005; Sommerville, 2004). Commercial tools, such as IBM–Rational XDE, are covering main stages of ISLC; the "Requisite-Pro" module, for instance, is designated to the stage of requirement definition, "Rose" module to the analysis and design stage, and "Test-Studio" module to the testing stage.

Although PM tools support management and control along the ISLC, there is hardly any integration between CASE tools and PM tools. Thus, ISLC modeling approaches, such as the functional approach (e.g., DFD, ERD, STD), as well the object-oriented approach (e.g., use cases, activity diagrams, STD), even when automated, are used mainly in the early analysis stage primarily for visual documentation. The "database of specifications," laboriously elicited and gathered during the creation of modeling diagrams, is hardly ever applied again for project management purposes, even though this information is valuable for project managers who are involved in the construction and implementation stages. In fact, due to lack of integration along the ISLC, the specifications database is often either overlooked altogether or collected again as if their creation earlier never took place. Moreover, standard methods for system analysis and development usually make no reference to methods for project management. Accounting and costing parameters, which are reviewed at the next chapter, are not represented not at SE tools or at PM tools, and handled in totally separated systems.

One conclusion that emerges from a thorough review of software engineering and project management areas is that SE tools are much more heterogeneous than PM tools. Gantt and Pert charts have become dominant project management modeling tools (Fox & Spence, 1998; Hughes & Cotterell, 2002) and are currently included in standard PM software such as Microsoft Project, PS-Next, and others. A survey of 1,000 project managers has found that 48.4% use MS Project, 8.5% use MS Excel, and the rest use Gantt/Pert-based tools from other vendors. The average satisfaction from PM tools in this survey was 3.7 on a scale of 1 to 5. Another survey reveals that only 10% of 240 project managers do not use PM tools at all, down from 33% in 1996. Moreover, more than 50% use Gantt/Pert-based project management software to manage every project, independently of its application domain and characteristics.

In contrast, the following two commercial CASE software packages demonstrate the heterogeneity of tools in the area of software engineering. Oracle's *Designer* supports functional hierarchy analysis based on Barker and Longman (1992) methodology, and IBM-Rational offers XDE-*Rose*, a modeling tool based on the unified modeling language (UML) only. PM tools thus seem more standardized and mature than CASE tools. This could be the reason why 71% of 397 software engineers surveyed in 20 European countries employ PM tools while only about 26% utilize CASE tools, despite similar levels of training (Domges & Pohl, 1998).

Although CASE tools, including those mentioned above, support teamwork, none contain elements that take into consideration teamwork planning, time planning, dependencies, resources allocation, cost estimation, or risk management. Moreover, none include Gantt or Pert models or offer built-in interfaces to PM tools. Methodologies and models for managing software projects have yet to make it from the idea to the product phase, despite persistent improvements in automated tools for requirement definition, systems modeling, and software engineering. The failure to transform project management theory to practice in the context of software development is especially troubling since more than 50% of such projects do not succeed (Madpat, 2005; Reel, 1999). In addition to the lack of integration between SE tools and PM tools managers in charge of software projects usually refrain from basing managerial judgement on data about requirements and functional characteristics of the specific development project (Reel, 1999).

With decades of systems development behind us, there is quite a consensus today with respect to the critical success factors (CSF) of system development projects and agile methodologies, there is still a need to introduce effective concepts, methods, measures, and tools for better control of software projects. All these observations lead one to conclude that assembling a repository of system requirements and system components, complete as it might be, does not guarantee effective planning of teamwork, scheduling of tasks, and controlling deviations between planned milestones and actual progress.

Against this background, the questions to consider are:

I. Is the gap between SE tools, costing methods, and PM tools is bridgeable?
II. Can SE components, collected by CASE tools, become directly available for the use of cost estimation, risk management, and directly integrated within PM without being subjectively interpreted or biased?
III. Is there a way to improve software modeling and engineering by introducing a managerial perspective in addition to the technical perspectives?

Our preliminary answers to those questions are "yes." This study proposes and prototypes a model that integrates these three aspects of software projects by automatically mapping SE objects and accounting–costing objects into PM objects. We have engaged in symmetry–isomorphism research with respect to distinct methodologies for software engineering and project management. Since Gantt chart is a technique for visual description of networks, the ability to convert DFD model or hierarchical use-case model to a network format is at the basis of our symmetry–isomorphism research. It is our intention in this article to demonstrate, based on this research, a possible integration scheme and provide more robust answers to the above questions. Given the wealth of CASE and PM tools, this work refrains from developing yet another one, but prototyping an integrated platform built-up of common CASE tool, costing models, and common PM tools. We show that combining these sets of capabilities can create the desired synergy where the whole is greater than the sum of its parts.

Costing Aspects and Methods

Detailed costing information is expected to include all types of costs that are required for manufacturing a product-software or providing a service. Data based on financial systems, which contain costs, derived from the income statement and the estimation of the company's capital and assets, enclosed the historical execution data and future estimations and forecasts (Roztocki & Needy, 1999). Williams (2004) supports the integration approach according to the conception that a modern accounting system is supposed to supply a framework for strategic management of the company's resources. In order to realize this conception, Williams proposes a multidimensional construct that clusters information from the company's systems on customers base, activity areas, and more for the purpose of forming an accounting system that facilitates planning, improvement and control, analysis and regulation of resources, and enhancement of profitability. Such a system is based on integrative information from a number of systems or from the arrays DW (data warehouse), BI (business intelligent) in five areas: costs, assets, quality/service, time, and outputs. The pioneers of the combining of financial and operational information are Cooper and Kaplan who developed the method of activity based costing (ABC) at the end of the 1980s. Cooper and Kaplan (1998) suggest in light of the technological development of information systems to define the integration between operational and financial systems for the purpose of building an accurate costing model.

In light of the above, establishment of integration conception required the definition not only of an enterprise costing model but also the definition of interfacing between the different areas and systems, that is, interface between SE aspects tools, financial aspects tools, and PM tools. Cost management is a term used for a wide description of short-term and long-term managerial activities involved in planning and controlling of costs (Horngren, Foster, & Datar, 2000). Table 1 presents variety aspects of costing model in a technological projects environment.

Costs analysis within the framework of technological environment must be carried out with the understanding of the project life cycle. Kerzner (2000) portrays the distribution of the project's cost over the project's life cycle:

- 5% - Conceptualization
- 10% - Feasibility study
- 15% - Preliminary planning
- 20% - Detail planning
- 40% - Execution
- 10% - Testing and commissioning

Tasks in each of these stages are described under the work breakdown structure (WBS). The WBS represents the required activities for the project's management in a hierarchical structure.

For each component of the WBS, an evaluation of direct and indirect (overhead) costs must be included. Direct costs are divided to work's cost (usually work hours multiple hourly rate) and direct costs that are not work payment such as travel, materials, and so forth. It is recommended that these costs will include managerial reserve as well (Jurison, 1999).

A reinforcement of the need to include the project's tasks (or the WBS components) in a costing model is intensified in the light of the cost estimations that are founded on work hours' evaluation. It has been argued (Ooi & Soh, 2003) that according to traditional approaches of software costing (time-based estimations), there may be a bending towards time planning without linking it to the specific task and the role player that performs it. Therefore, it is suggested to include the detailing of the tasks (Ooi & Soh, 2003) and/or an elaborate planning of the various project's resources as part of the costing model. The advantages of the resources' cost analysis throughout activities/tasks are more detailed information for managers, monitoring abilities, analysis of resources' cost and allocation, and a more accurate ability of overhead allocation (Jahangir, 2003; Kinsella, 2002; Ooi & Soh, 2003; Raz & Elnathan, 1999).

Indirect costs (overhead costs) include all types of costs that cannot be attributed directly to a specific task in the project marketing and sales expenses, office supplies, buildings' cost, professional services, information systems, computerization infrastructure, and the like. These costs are only occasionally incorporated in the project planning, but they carry great influence on the profitability of the portfolio and the projects' pricing decisions (Horngren et al., 2000). These costs are described as one of the "major headaches" (Kerzner, 2000). However,

Table 1. Aspects of costing model in a technological projects environment

	Aspect	Description	Difficulties
1.	**Planning**	Costs estimation of the project and for each resource in the projects portfolio	Defining direct and indirect resources and their costs
2.	**Controlling**	Costs analysis for each project and executed task	Attributing in-reality-costs to each project's task
3.	**Timeline**	Costs analysis over different time periods in planning and execution	Evaluating capacities of resources consumption over specified time periods
4.	**Tasks**	Identification and costing of project's tasks (WBS items)	Matching the costs to each of the project's components
5.	**Overhead Allocation**	A precise allocation of indirect costs	Determining the indirect cost generators in project's tasks
6.	**Risk management**	The inclusion of risk element and its value as part of the costing	Estimating risk on the basis of risk factors in the different tasks
7.	**Scenarios**	The ability to analyze alternative modes of action and costs	Defining assumptions and alternatives to the mode of cost's calculation
8.	**Profitability Analysis**	The understanding of the profit that derives from each of the projects and the whole projects portfolio	The inclusion of all the cost factors in the model

in this context, it has been argued that the ability to control costs is largely dependent on the monitoring of these costs.

Table 2 summarizes costing methods according to financial and engineering literature. The table also presents the common evaluation of model compatibility in light of entire costing aspects.

- **Analogy:** Cost estimation based on previous experience, using case-based reasoning (CBR) techniques. The accuracy of this method ranges from –10% to +25% (Kerzner, 2000).
- **Parametric:** Cost estimation based on heuristics and thumb's rules (Jahangir, 2003). Similar to the analogy estimation method, a parametrical model is also based on accumulation of historical data of project costs. On the basis of these data, a mathematical model is defined for the prediction of costs (Kinsella, 2002). The level of accuracy of a parametrical model ranges on a wide scope of –25% to +75% (Kerzner, 2000).
- **Function points:** A method that was first introduced in 1979 by Albrecht. Its objective is to assess the software system's size while using the user's requirements without direct dependence on the technological realization (Hale & Smith, 2001). The function points method is calculated in three steps using the quantity and complexity of the functional components and the system attributes (Kemerer, 1993).
- **COCOMO (constructive cost model):** The model was first introduced in 1981 and since then several modifications were made in order to suit fourth generation languages, decrease in hardware costs, increase in quality assurance (QA) levels, advanced and agile development methods. The current version, COCOMO 2.0 (Boehm, Clark,

Table 2. Costing methods according to financial and engineering literature

			Planning	**Controlling**	**Time Line**	**Task Resolution**	**Overhead Allocation**	**Risk Management**	**Scenarios**	**Profitability Analysis**
Software Eng.	Top Down	Analogy	√ *	P*	X	X	P*	X	X	X
		Parametric	√ *	P*	X	X	P*	P*	√	X
	Bottom Up	Function Points	√	X	X	√	P*	√*	√	X
		COCOMO II	√	P	X	√	P*	√	√	X
Costing		Target Costing	√	P	P	P*	P	X	P	√
		Standard Costing	√	P	√*	√*	P	X	√	P
		ABC	√	P	P*	X	√	X	√	√

*√ - Good compatibility; X - No compatibility; P - Partial compatibility; * - Adjustments are required*

Horowitz, Madachy, Sclby, & Westland, 1995), is not based upon line of codes but on four submodels that match a spiral approach of software system development that are applied according to the stage of the life cycle (the application-composition model, early design model, reuse model, and post-architecture model).

- **Target costing:** Suits engineering framework in which there are several engineering activities simultaneously and is utilized as a means for costs strategic management. The idea behind the method is that a product's cost must be based on the sum that can be received for it in the market, and in other words, the development cost should be the basis for the quantity and mode of investment in the development rather than the development's outcome.
- **Standard costing:** Ascertains the cost framework while employing the amount of direct cost components and a standard price that was set for this unit. We shall formulate it concisely as:

$$TotalCost = \sum_{i-1}^{n} Qty_i * StdP_i \cdot$$

It should be accentuated that the standard price does not solely include the direct price of the component (price per working hour) and is intended to contain the meaning of the cost or the consumption of indirect resources (rent, computerization, etc.). In the calculation of the standard price, it is customary to rely on known performance data from the past (Horngren et al., 2000).

- **Activity based costing (ABC):** Considered one of the advanced models for predicting costs while incorporating managerial decisions. The model was developed in the 1980s and its main innovation is in the addition of nonfinancial elements to the costing model. The model is widely used in a variety of industries such as agronomy, banking (Kao & Lee, 2001), and medicine. In the projects area, there is not much literature that discusses the application of ABC; however, there are a few studies that help to understand the method. These studies include the description of the method for software developing and assimilation (Ooi & Soh, 2003), the portrayal of the mode in which ABC can be taken on in projects (Raz & Elnathan, 1999), the implementation of ABC in favor of IT cost analysis in the organization, and a recommendation to include this model in the project management body of knowledge, or PMBOK (Kinsella, 2002).

THE INTEGRATED MODEL

The integrated model is based upon former research (Gelbard, Pliskin, & Spiegler, 2002) that has mapped data flow diagrams, which are actually comprehensive enterprise sets of use cases diagrams reflecting entire system-software project into Gantt charts. Current research is focused on the following extensions:

- Extending the database schema, used as the integrated system repository, in a flexible way enabling the addition of any costing parameter to each of the DFD/use case/SE components.
- Adding specific manipulations and outputs in order to support presentation of costing aspects.

Mapping DFD/Use Case Objects into Gantt Objects

As suggested in former research (Gelbard et al., 2002), data flow diagrams as well as use case diagrams can be mapped into Gantt charts based on the following conversions:

1. Each of the external entities are represented once only for input (if they produce input) and once only for output (if they produce output).
2. Each read only (RO) data store and each read/write (R/W) data store are represented once only for input and once only for output.
3. Each basic flow appears once only in every Gantt diagram.
4. Each basic process appears once only in every Gantt diagram.
5. OR connections between flows are not represented in the absence of parallels in Gantt diagrams. Logical connection traits between flows can, however, be included within basic process characteristics, thus maintaining mapping completeness.
6. A general process is represented by means of a summary task, that is, a grouping of activities and flows under a general name.
7. General flows are those that connect between summary tasks.

Example: Mapping of Hierarchical DFD

DFD as well as use case methodology enable hierarchical analysis of systems. The hierarchical description is achieved by "blowing up" general processes-usages into dedicated diagrams. Those dedicated diagrams represent lower level descriptions and can be composed of basic pro-

Figure 1. DFD-0

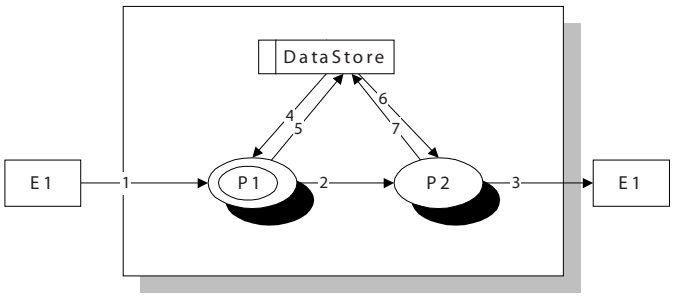
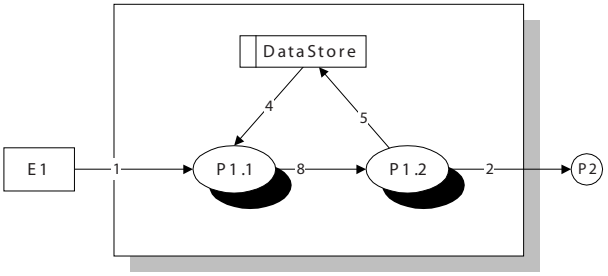


Figure 2. DFD-1 ("Blow-up" of General Process P1)



cesses, general processes, and of "includes" and "extends" use cases. The hierarchical description can be halted when there are no more general processes at the lowest-level diagrams. Except for the root level, identified as "DFD-0," each diagram in the DFD hierarchy is identified by the respective general function, and the same can be defined for use case diagrams. For the sake of simplicity, the DFD used in the examples below contains only the following objects: basic and general processes, flows, external entities, and data stores. Processes are symbolized by ellipses and denoted by **P#**, entities by rectangles and **E#**, and flows by arrows and **#**.

Figures 1 and 2 demonstrate a hierarchical DFD. Figure 1 describes the root level with Basic Process P2 and General Process P1 (a general process is depicted by concentric ellipses). Figure 2 describes a lower level description of the General Process P1.

A composite DFD, made up from both DFD-0 (Figure 1) and DFD-1 (Figure 2), is shown in Figure 3. A composite DFD, not a typical or common representation of a hierarchical DFD, is included here because of its similarity to Gantt diagram representation, where summary tasks and subtasks can be displayed on the same diagram. A summary task represents, in the mapping model, a general process, while each subtask represents a component at the respective hierarchical DFD level. As can be seen in the fourth section, Figure 6 depicts the Gantt diagram corresponding to DFD-0 in Figure 1, and Figure 7 depicts the mapping of the composite DFD in Figure 3 into a Gantt diagram.

Adding Costing Aspects

In order to enable location of any costing parameter to each of the DFD/use case models, we have used predefined symbols for each costing aspect. Noting a symbol with a numerical parameter beside it, in the "Label" input box at any "engineering" object dialog box, as shown in Figure 4, inserts relevant costing value (in light of the relevant costing aspect) to the relevant software component. A parser, running over the CASE repository, recognizes those predefined symbols and allocates the costing data in the integrated repository for further manipulation and interchanges. Overheads (indirect costs) inputs are made as labels of the entire project (the DFD-0 object itself).

Figure 3. A composite DFD representing both DFD-0 and DFD-1

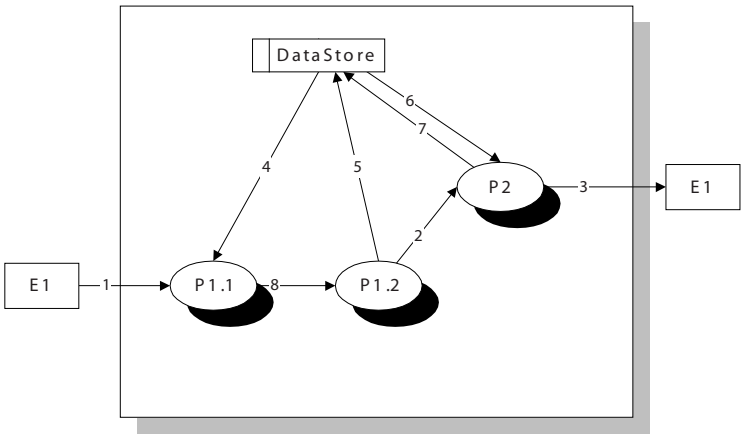
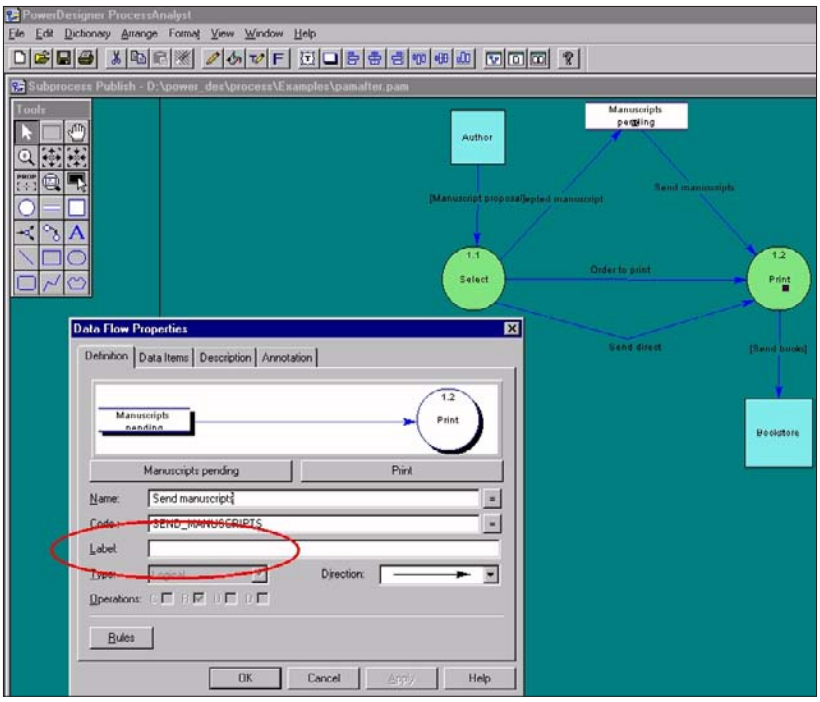


Figure 4. The "Label" input box used to insert costing aspects and values



The Integrated Repository

As mentioned above, the database schema, used as the integrated system repository, was extended in a flexible way so it is possible to add any costing parameter to each of the DFD/use case/SE components. Figure 5 illustrates the integrated database schema, which supports engineering objects (DFD/use cases), PM objects (WBS and ascription of dependencies), and costing objects (costing aspects and values). The integrated repository contains three main components: SE components, PM components, and costing components.

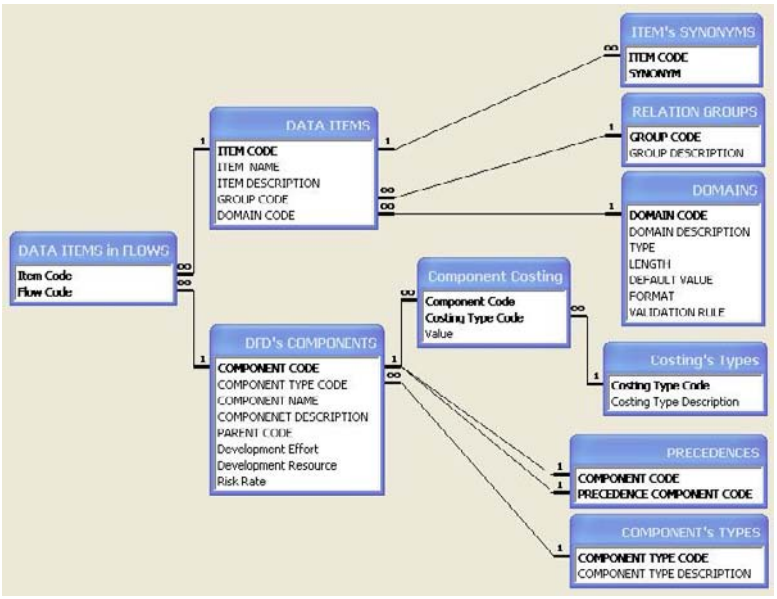
The SE component is based upon the following tables:

- Objects Dictionary = Tables: [DFD's COMPONENTS], and [COMPONENT's TYPES].
- Data Items Dictionary = Tables: [DATA ITEMS], [ITEM SYNONYMS], [DOMAINS], and [RELATION GROUPS].
- Ascription of Data Items to Basic Flows = Table: [DATA ITEMS in FLOWS].

The PM component is based upon the following tables:

- Ascription of dependencies between SE objects = Tables: [DFD's COMPONENTS], and [PRECEDENCES].

Figure 5. Database schema of the integrated repository



The costing component is based upon the following tables:

- Ascription of costing aspects and values to each SE object = Tables: [DFD's COMPONENTS], [Component Costing], and [Costing's Types]

In this way, those components enable representing and manipulating of DFD objects, costing aspects, and Gantt objects. The [PRECEDENCES] table stands for many-to-many network relationships as required for a Gantt representation.

In Figure 5, rectangles represent database tables, with the table name contained in the blue header and the primary key bolded. Lines between rectangles represent the database constraints (foreign key) and indicate the cardinality (one-to-many) of the relation. To distinguish between the various DFD/use case components, each component, in the [DFD's COMPONENTS] table, is attributed to a component type, defined in the [COMPONENT TYPES] table. The [Cocting's Types] table stands for the variety of costing aspects, while the [Component Costing] represents many-to-many relations between SE component-object and costing aspects. The specific value of costing aspect of the relevant SE component-object is stored in this table (in the [Value] field).

THE PROTOTYPE

Data flow diagrams where constructed by using the CASE tool "Power Designer; Process Analyst," and it is worth mentioning that the same can be done for use cases using the same tool. Costing aspects and values were referenced to each SE

component (see Figure 4). A parser while running over the CASE repository inserts accordingly the relevant records to the integrated database schema, as described in the third section and illustrated in Figure 5. Then, the prototype activates MS Projects and MS Excel, creating project Gantt charts (by MS Project), and project summation (by MS Excel). The following figures illustrate two kinds of outputs: "classic" PM outputs and costing outputs. "Classic" PM outputs relate to Gantt chart, while costing outputs relate to project summations.

"Classic" PM Outputs

DFD models shown in Figures 1, 2, and 3 were mapped into MS Project Gantt charts. Figure 6 displays the Gantt chart corresponding to DFD-0 in Figure 1, and Figure 7 displays the mapping of the composite DFD in Figure 3 into a Gantt chart. The Summary Task P1 in Figures 6 and 7 (line 10) corresponds to General Process P1, while subtasks of P1 are represented only in Figure 7. The P1 Summary Task has a distinct symbol with emphases at the edges.

Double clicking on line 10 "blows up" the summary task and displays its subtasks. Lines 11, 12, and 13 in the Gantt chart of Figure 6 are hidden because they are related to a lower level in the DFD hierarchy, DFD-1 (Figure 2). Lines 11, 12, and 13 in Figure 7 represent the DFD objects that are shown in DFD-1 (Figure 2).

Figure 6. Ms Project Gantt chart representation of Figure 1 DFD's objects

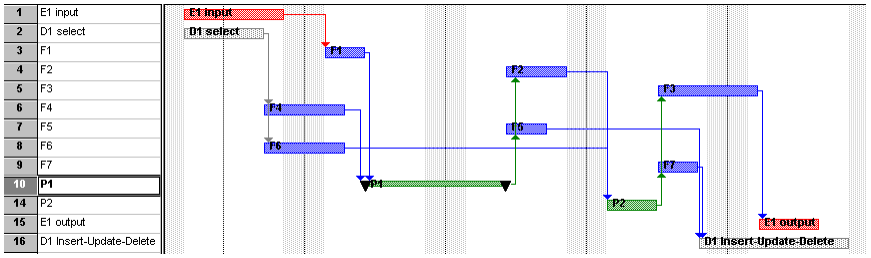
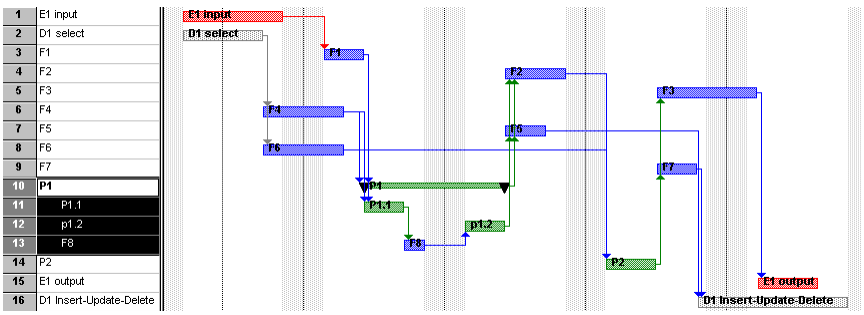


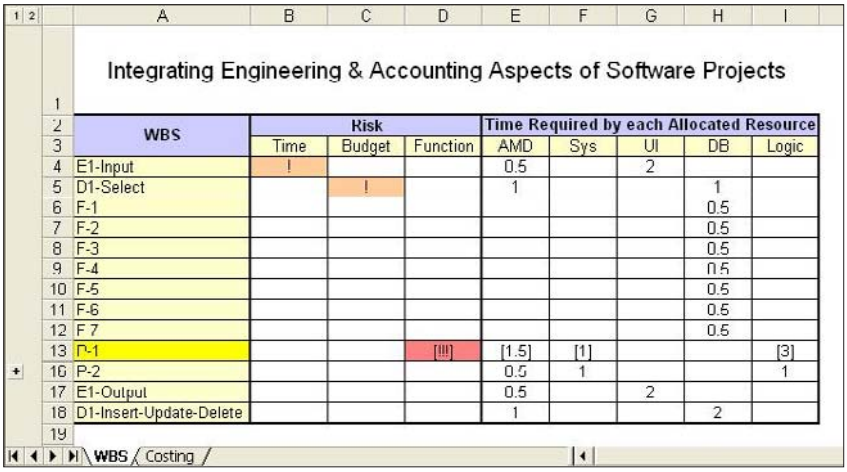
Figure 7. Ms Project Gantt chart representation of Figure 3 DFD's objects



Costing Outputs

Costing aspects of risk and direct costs where constructed by using the "Label" input box (see Figure 4). A parser, while running over the CASE repository, inserts accordingly the relevant records to the integrated database schema and then activates the Visual Basic Excel module, which

Figure 8. Summary of risks and resources

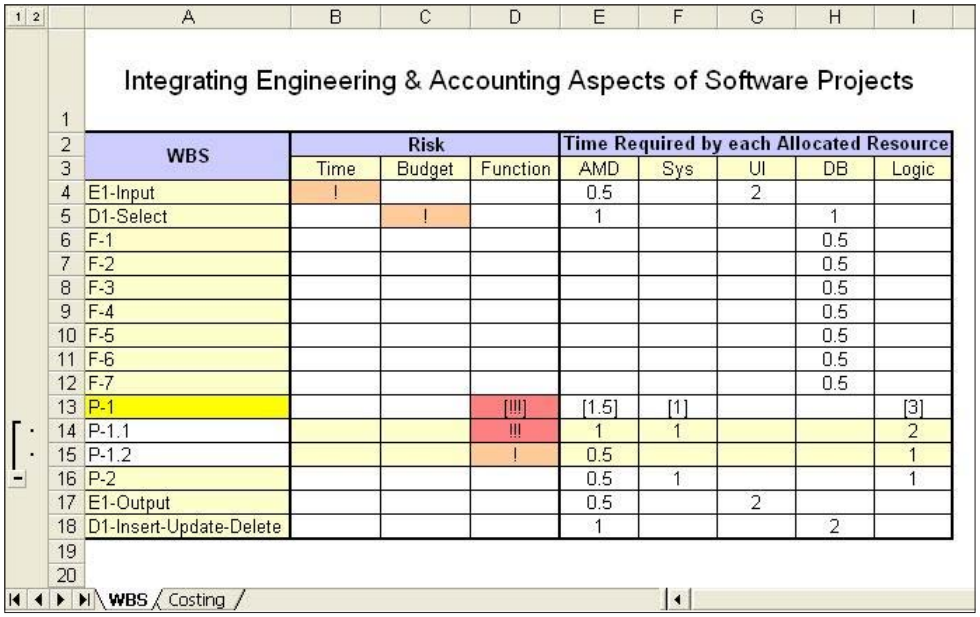


Integrating Engineering & Accounting Aspects of Software Projects

WBS	Risk			Time Required by each Allocated Resource				
	Time	Budget	Function	AMD	Sys	UI	DB	Logic
E1-Input	!			0.5		2		
D1-Select		!		1			1	
F-1							0.5	
F-2							0.5	
F-3							0.5	
F-4							0.5	
F-5							0.5	
F-6							0.5	
F-7							0.5	
P-1			[!!!]	[1.5]	[1]			[3]
P-2				0.5	1			1
E1-Output				0.5		2		
D1-Insert-Update-Delete				1			2	

WBS / Costing

Figure 9. "Blown-up" of risks related to Process "P-1"



Integrating Engineering & Accounting Aspects of Software Projects

WBS	Risk			Time Required by each Allocated Resource				
	Time	Budget	Function	AMD	Sys	UI	DB	Logic
E1-Input	!			0.5		2		
D1-Select		!		1			1	
F-1							0.5	
F-2							0.5	
F-3							0.5	
F-4							0.5	
F-5							0.5	
F-6							0.5	
F-7							0.5	
P-1			[!!!]	[1.5]	[1]			[3]
P-1.1			!!!	1	1			2
P-1.2			!	0.5				1
P-2				0.5	1			1
E1-Output				0.5		2		
D1-Insert-Update-Delete				1			2	

WBS / Costing

has created a project summation accordingly. Hierarchical presentations were applied using built-in features of MS Excel.

Figures 8 and 9 illustrate hierarchical presentations of the same DFD models shown in Figures 1, 2, and 3. Figure 8 displays the summation corresponding to DFD-0 in Figure 1, and Figure 9 displays the summation of the composite DFD in Figure 3. The Summary Task P1 in Figures 8 and 9 (line 13) corresponds to General Process P1, while subtasks of P1 are represented only in Figure 9. The P1 Summary Task has a distinct background with "+" sign beside it.

Risk was assigned separately to time, budget, and functional aspects of each SE component-object. Direct costs are presented in units of months. Further calculation can be made according to parameters of salary rates and overheads, presented in the "costing" sheet.

Double clicking on the "+" sign beside line 13 "blows up" the summary task and displays its subtasks. Lines 14 and 15, in the Excel spreadsheet of Figure 8, are hidden because they are related to a lower level in the DFD hierarchy, DFD-1 (Figure 2). Lines 14 and 15 in Figure 9 represent the DFD objects that are shown in DFD-1 (Figure 2).

DISCUSSION AND CONCLUSION

Current research demonstrates the derivation of PM objects and project risk and costing evaluation, directly on the basis of raw SE components-objects. By this, we provide an integrating layer, which combines standard PM tools with common system analysis and design tools and costing aspects and models. Applying such an integrating layer in software development projects enables improved risk and cost estimations at the very beginning stages of a software project, as well as better monitoring and control over various topics uniquely related to software projects.

The integration model presented in this study has the following advantages:

1. The integrated model enables risk, effort, direct, and indirect cost estimation for software development to be an integral part of conventional analysis and design methodologies. This is due to the possibility of deriving assessments *directly from raw data stored at the repository of a CASE tool*, as opposed to relying on aggregates or other secondary sources.
2. The integrated model enables extension of the basic development time assessment beyond the mere aspects of time and direct cost *to include also risk, overheads, and any accounting-costing parameter.*
3. The integrated model bases the estimation process on the *very common management tools*, for example, MS Project and MS Gantt.
4. Use of Gantt charts enables *dynamic* control of the estimation, based on reports regarding *actual* progress. This provides for *projected-to-actual* comparisons of cost and *moment-to-moment* updates of CPM calculations, as opposed to the *static* control methods customary in the field of software development.
5. Use of Gantt charts allows *"drilling down,"* into the *system code design,* including master routines and system major service routines. This differs from current methods used for software development, which are limited primarily to the area of *functionality.*
6. The potential of a detailed drill-down concerning system code design provides for engaging and *integrating* the technical team (development managers) as early as the analysis stage. This results in a more *reliable* and *accurate* estimation base for the entire system development project.

It is noteworthy that conversion of a DFD model as well as of a use-case model into a Gantt chart is actually a representation of a knowledge model as a semantic network. We have reason to believe that

the integration of SE aspects and costing aspects within PM tools, modeled and prototyped in this study, is not limited to the DFD/use-case approach or to the described costing methods, but can be applied to any "network-based" software engineering modeling, as well as to any additional costing method. The fact that DFD conversion into Gantt chart is actually a representation of a knowledge model as a semantic network opens opportunities for commercialization for practitioners.

In sum, this study showed the feasibility and validity of converting SE objects and the risk and the cost we ascribe to those software components into PM objects. The integration of common CASE tools with costing models and standard PM tools can potentially improve estimation, planning, and control of software development projects in terms of cost, time, and risk management. In software projects, where so many things may go out of control, any theoretical as well as practical novelty is required in order to gain additional progress.

REFERENCES

Armour, P. (2002). Ten Un-Myths of Project Estimation. *Communications of the ACM, 45*(11).

Barker, R., & Longman, C. (1992). *CASE method: Function and process modeling.* Addison-Wesley.

Barker, B.G., & Verma, D. (2003). System engineering effectiveness: A complexity point paradigm for software intensive systems. *Engineering Management Journal, 15*(3), 29.

Becker, S.A., & Bostelman, M.L. (1999, May/June). Aligning strategic and project measurement systems. *IEEE Software,* pp. 46-51.

Boegh, J., Depanfilis, S., Kitchenham, B., & Pasquini, A. (1999, March/April). A method for software quality planning, control, and evaluation. *IEEE Software,* pp. 69-77.

Boehm, B.W., Clark, B.K., Horowitz, E., Madachy, R., Sclby, R.W., & Westland, C. (1995). Cost models for future software processes: COCOMO 2.0. *Annals of Software Engineering, 1,* 57-94.

Cooper, R., & Kaplan, R.S. (1998, July/August). The promise and peril of integrated cost systems. *Harvard Business Review, 76*(4), 109-119.

Dasher, G.T. (2003). The interface between systems engineering and program management. *Engineering Management Journal, 15*(3), 11-21.

Datar, S.M, Foster, G., & Horngren, C. (2000). *Cost accounting* (10th ed.). Prentice Hall.

Domges, G., & Pohl, K. (1998). Adapting traceability environments to project-specific needs. *Communication of the ACM, 41*(12), 54-62.

Dutta, S., Lee, M., & Van Wassenhove, L. (1999, May/June). Software engineering in Europe: A study of best practices. *IEEE Software,* pp. 82-89.

Fox, T.L., & Spence, J.W. (1998, September). Tools of the trade: A survey of project management tools. *Project Management Journal*, pp. 20-27.

Gelbard, R., Pliskin, N., & Spiegler, I. (2002). Integrating systems analysis and project management tools. *International Journal of Project Management, 20,* 461-468.

Geri, N., & Ronen, B. (2005, April). Relevance lost: The rise and fall of activity-based costing. *Human Systems Management*, pp. 1-12

Grady, R.B. (1994). Successfully applying software metrics. *Computer, 27*(9), 18-25.

Hale, J., & Smith, R. (2001). An empirical study using task assignment patterns to improve the accuracy of software effort estimation. *IEEE Transactions on Software Engineering, 27*(3), 264-271.

Horngren, C., Foster, G., & Datar, S. (2000). *Cost accounting.* Prentice Hall.

Hughes, B., & Cotterell, M. (2002). *Software project management* (3rd ed.). McGraw-Hill.

Jahangir, M. (2003). Costing R&D projects: A bottom-up framework. *Cost Engineering, 45*(2), 12-20.

Jurison, J. (1999). Software project management: The manager's view. *Communications of the AIS, 2*(17).

Kaindle, H., & Carroll, J.M. (1999). Symbolic modeling in practice. *Communications of the ACM, 42*(1), 28-30.

Kao, J., & Lee, T. (2001). Application of simulation technique to activity-based costing of agricultural systems: A case study. *Agricultural Systems, 67,* 71-82.

Kemerer, C.F. (1993). Reliability of function points measurement: A field experiment. *Communications of the ACM, 36*(2), 85-98.

Kerzner, H. (2000). *Project management* (7th ed.). John Wiley & Sons.

Kinsella, S.M. (2002). Activity-based costing: Does it warrant inclusion in a guide to the PMBOK? *Project Management Journal, 33*(2), 49-56.

Maciaszek, L.A., & Liong, B.L. (2005). Practical software engineering: A case-study approach. Addison-Wesley.

Madpat, S. (2005). Bridging organizational strategy and projects: An OPM3 insiders perspective. *PMI Journal, 5*(10), 16-20.

Ooi, G., & Soh, C. (2003). Developing an activity-based costing approach for system development and implementation. *The DATA BASE for Advances in Information Systems, 34*(3), 54-71.

Pendharkar, P.C., Subramanian, G.H., & Rodger, J.A. (2005). A probabilistic model for predicting software development effort. *IEEE Transactins on Software Engineering, 31*(7), 615-624.

Rajkumar, R., & Rush, C. (2000). Analysis of cost estimating processes used within a concurrent engineering environment throughout a product life cycle. In *Proceedings of the 7th ISPE International Conference on Concurrent Engineering* (pp. 58-67).

Raz, T., & Elnathan, D. (1999). Activity based costing for projects. *International Journal of Project Management, 17*(1), 61-67.

Reel, J.S. (1999, May/June). Critical success factors in software projects. *IEEE Software*, pp. 18-23.

Roztocki, N., & Needy, K.L. (1999). Integrating activity-based costing and economic value added in manufacturing. *Engineering Management Journal, 11*(2), 17-22.

Sommerville, I. (2004). *Software engineering* (7th ed.). Addison-Wesley.

Wouters, M., & Davila, A. (2004). Designing cost-competitive technology products through cost management. *Accounting Horizons, 18*(1), 13-26.

KEY TERMS

Activity Based Costing (ABC): A cost prediction model that has greatly improved the ability to predict a proper allocation of indirect costs among several activities and thereafter between many products. Before using this model, one has to appreciate and understand the overall business (including its production and marketing). The model is not always applicable: a cost-benefit analysis is necessary before a final decision is made.

CASE/AMD Tools: Software tools (Computer Aided Software Engineering), also named AMD tools (Analysis Modeling and Design), to assist entire System Life Cycle (SLC), including analysis phase, design phase, testing phase, and even

maintenance phase. CASE/AMD tools support the fuctional analysis phase using visual modeling notations, which can automatically convert into code.

COCOMO (Constructive Cost Model): An estimation method used to assess the human effort required for software development, which was first introduced in 1981, and since then several modifications were made in order to suit fourth generation languages, decrease in hardware costs, increase in QA levels, and advanced and agile development methods. The current version, COCOMO 2.0 (Boehm et al., 1995), is not based upon line of codes but on four submodels that match a spiral approach of software system development that are applied according to the stage of system life cycle.

Function Points: An estimation method used to assess the human effort required for software development, which was first introduced in 1979. Method objective is to assess the software system's size while using the user's requirements without direct dependence on the technological realization. It is calculated in three steps using the quantity and complexity of the functional components and the system attributes.

Gantt Chart: A popular type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates (ES, EF, LS, LF) of entire project elements. Project elements comprise the work breakdown structure (WBS) of the project. Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "Today" line.

Modeling Languages: Visual-graphical notations used to express functionality, processes, structures, behavior, as well as technical aspects of a system. Modeling languages are defined by a consistent set of rules, which are used for interpretation of those graphical symbols. Among modeling languages, in the software domain, there are data flow diagrams, used for hierarchical functionality decomposing; process diagrams, used for business processes modeling; entity relation diagrams, used for data structure modeling; and state transition diagrams, used for behavior modeling.

Target Costing: A cost prediction model that suits an engineering framework in which there are several engineering activities which take place simultaneously. It is utilized as a means for costs strategic management. The idea behind the method is that a product's cost must be based on the sum that can be received for it in the market. It is therefore the development costs which should be the basis for the quantity and mode of investment in the development rather then the development's outcome.

UML (Unified Modeling Language): A standardized visual-graphical notation gathering together diverse modeling diagrams, which are required in order to define entire software system aspects. UML is officially defined at the Object Management Group (OMG) by the UML metamodel, a meta-object facility metamodel (MOF).

This work was previously published in Encyclopedia of Information Communication Technology, edited by A. Cartelli & M. Palma, pp. 443-456, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 3.15
Developing Enjoyable Second Language Learning Software Tools:
A Computer Game Paradigm

Chee Siang Ang
City University, UK

Panayiotis Zaphiris
City University, UK

ABSTRACT

This chapter attempts to examine computer game theories—ludology and narratology—that explain computer games as play activities and storytelling media. Founded on this theoretical explanation, a game model that incorporates gameplay and narratives is presented. From the model, two aspects of learning in the game environment are identified: gameplay-oriented and narrative-oriented. It is believed that playing computer games involves at least one of these types of learning; thus, this game's nature can be used in designing engaging educational software. In addition, based on Malone's theoretical framework on motivational heuristics, there are two methods of applying computer games in language learning: extrinsic and intrinsic, depending on the integration of game designs and learning materials. Then, two cases of language-learning games are scrutinized, using the game model, in order to demonstrate the use of computer games in language learning.

INTRODUCTION

In one of his most influential texts about computer games, *The Art of Computer Game Design*, Chris Crawford (1982) states that schools, but not games, are the untested fad and violator of tradition in education. Game playing is a vital educational function for any creature capable of learning. Hence, games are the most ancient and time-honored vehicle for education. Crawford

explores the reasons for people playing games and asserts that the fundamental motivation of game playing is to learn. He also cites an example to support his view by observing the behavior of lion cubs near their mother: the cubs crouch in the grass, creeping slowly toward a butterfly and pouncing on it. The beasts are apparently playing some sort of game and having fun. However, the game is also how lions learn to hunt their prey without being injured. They are learning by doing, with minimum risks. This observation is true not only for animals. Since the dawn of human history, games have been used in the teaching and learning process. Board games, for example, are believed to be the earliest games, and they were battle simulations designed to instruct the young (Murray, 1978).

The ability of computer games to spark interest among players can hardly be denied, and some educators have started to see the capability of these highly engaging games. People play games voluntarily, without asking for external rewards. Besides, the use of computer games in learning is parallel with Piaget's constructivism, in which knowledge is constructed instead of being transmitted. A lot of game-based learning projects have been carried out with an emphasis on this pedagogical epistemology. Nevertheless, most of these projects are centered in science education and mathematics. Not much theoretical work has been done on language learning, although computer games have long been used in this area. This is due to the fact that computer games are too varied and intricate to indicate a clear function in language education. Furthermore, what counts as a game is rather loosely defined. Therefore, a proper study of computer game theories would throw light upon this issue.

This chapter is structured as follows: First, we review the theoretical parts of computer games, which include ludology and narratology. Then, a theoretical model of game is proposed. The next section explains two kinds of learning that occur when playing computer games based on the model. We outline two methods of integrating game designs with language learning. Next, an analysis of two cases of language-learning games is presented. Then, we discuss the future direction of this study; and the final section concludes the chapter.

THEORETICAL REVIEW ON GAMING

Although the use of computer games in learning is gaining attention among educators, there is still a lack of theoretical understanding of the game itself in most studies. Recent literature reveals that the research of computer games falls into two major principles: ludology and narratology. Ludology focuses on the study of computer games as play and game activities, while narratology focuses on the study of computer games as stories. The views between ludologists and narratologists are generally contradictory; the former argues that the pleasure of playing games lies in the gameplay, while the latter treats narrative as the fundamental enjoyment players are experiencing during the play session. In computer games, gameplay is referred to as activities conducted within a framework of agreed rules that directly or indirectly contribute to achieving goals (Lindley, 2002). A narrative is an account of something that happens to someone (Barrett, 1997). It consists of a series of events, from the background setting to the completion of the game. In other words, gameplay is the actions taken by the players, while narratives are an account of these actions. In this section, several kinds of game rules are explicated to better comprehend gameplay. The narrative mechanisms of the game are also scrutinized.

Gameplay and Ludology

The term ludology first appeared in the text of computer game research in 1999. Gonzalo Frasca (1999) points out in his paper, *Ludology Meets*

Narratology: Similitude and Differences Between (Video) Games and Narrative, that another dimension has been almost ignored when studying computer games: to analyze them as games. He proposes the term "ludology" to refer to the discipline that studies game and play activities as opposed to narratives, and asserts that ludology should be independent from the medium that supports the activity. Frasca (2001) identifies two kinds of game: ludus and paidea. Ludus refers to the games whose result defines a winner and a loser, while paidea refers to the games whose result does not (Frasca, 2001). Based on this difference, Frasca introduces two types of game rules: paidea rules and ludus rules. Paidea rules are established to play the game as paidea, while ludus rules are established to win or lose the game. In chess, for example, paidea rules describe how each token moves, while ludus rules state a condition to end the match. It is noticed that we can easily switch from paidea to ludus and vice versa. In SimCity (see Figure 1) — a paidea game in which no explicit ludus rules are defined — players can engage in paidea by playing with the buildings. Once they establish a goal, say, to build a city with a population of 10,000, they immediately switch to a ludus activity. Not only can we have several paidea rules, we can also have several ludus rules. In chess, we can define the winner by counting the amount and value of each player's remaining tokens. Table 1 shows some examples of paidea and ludus rules in computer games.

Besides rules, ludologists are also keen on understanding gameplay. Jesper Juul (2002) proposes two types of gameplay based on the relationship between the rules of a game and the actual game sessions played. First is emergent gameplay, where a number of simple rules are combined to form an interesting variation of gameplay. In a game of emergence, the game structure is primitive and defined by a set of simple paidea rules with usually only one ludus rule. Chess, for instance, is a game of emergence. It has a set of paidea rules that define how each piece moves, and one ludus rule, which is to take the opponent's King. Driven towards this explicitly stated ludus rule, players might construct more ludus rules (such as to take the Knight) and plan for complicated strategies to achieve the goal. Second is progressive gameplay,

Figure 1. SimCity 4000 (courtesy Electronic Arts, 2003)

Table 1. Paidea rules and ludus rules

	Paidea rules	Ludus rules
SimCity (Paidea game)	If the crime rate is high, the population becomes low	Nil
Tetris (Ludus game)	If the blocks fill a layer, the layer is cleared	To keep the level of block as low as possible

Figure 2. Myst III (courtesy UbiSoft Entertainment, 2001)

where separate challenges are introduced serially for the player to solve. Most adventure games, like Myst (see Figure 2), fall into this category. Players are introduced, one after another, to ludus rules of goals to be achieved that lead to the attainment of the ultimate game goal.

Narratives and Narratology

For narratologists, the advent of the computer signifies the birth of a new medium for storytelling. The capability of computers is not limited to performing calculations; computers are a new medium to represent human activities and present narratives in an unprecedented way. In most modern computer games, players can naturalize their actions as the solving of a familiar type of problem (Ryan, 1994). In Myst III, the player needs to track down the villain; in Super Mario Bros. 3 (see Figure 3), the player is trying to save Princess Toadstool; and in SimCity, the player plays the role of the mayor and plans for city development.

Marie Ryan (2001) tries to understand narratives in computer games, and she proposes a definition of narrative based on mental images: A narrative is defined as a mental image, or cognitive construct, that can be activated by various types of signs. This image consists of a world (setting) populated by intelligent agents (characters). These agents participate in actions and happenings (events, plot) that cause global changes in the narrative world. Several useful terms are recognized in this definition: world, character and action, and we would like to know to what extent these exist in computer games. First, a game has a spatial representation, whether real or abstract. Espen Aarseth (1998) has claimed in the article *Allegories of Space* that computer games are essentially concerned with spatial representation and negotiation. Myst, for example, has a rich description of space represented with high-quality pre-rendered 3D images. The player recognizes the space immediately after entering the game world, and knows what and how they should act because it resembles a real social setting. Pong, on the other hand, represents an abstract space, which might not have a referent in the real world. Players might construct their own mental image about the game, though most players would probably relate Pong with table tennis. The space in Pong is symbolic, while in Myst it is narrative.

Figure 3. Super Mario Bros. 3 (courtesy Nintendo, 1988)

One thing in common is that these two worlds operate within a strict set of rules that define the mechanism of the worlds. Second, most computer games feature explicit characters who will interact with the world or the player. In Myst, the characters are descriptive, and players can interact with them as if they are real humans, although the interaction is limited to several chosen aspects. In Pong, even though it does not have an explicit character, the player is apparently playing against an opponent or intelligent agent. The character is not depicted graphically in the game world, but the existence cannot be overlooked. Third, all games involve active actions and reactions of the players. Games are usually discerned from linear narratives by the existence of interaction: the reciprocal actions between players and games. These actions include not only the action of the player, but also the autonomous actions of the characters in the game world.

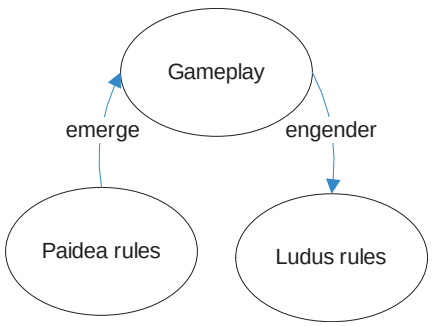
A GAME MODEL OF GAMEPLAY AND NARRATIVES

In this section, we propose a game model of gameplay and narratives that attempts to unify the view of ludologists and narratologists, thus advocating the study of computer games that comprises both gameplay and narratives. From the ludological perspective, we know that game rules are significant in understanding the semantics and structure of the game. Though they constitute a very important part of computer games, rules are not always the only thing one needs to learn in order to play. Game playing is more than simply memorizing the rules. Having learned the rules merely establishes the ability to play, and successful play does not necessarily require learning all the rules (Lindley, 2002). We need to understand something more complex that can arise from the rules: the gameplay. Examining the two types of rules and gameplay closely, it has been found that gameplay emerges from and must conform to the paidea rules that describe the semantic of the game. In addition, gameplay is oriented towards the ludus rules that describe the structure of the game. This relation is described in Figure 4.

Usually, paidea rules are fixed and predefined by the game designer. The player cannot breach paidea rules and their planning of strategies should conform to these rules. If the paidea rules state that the game character can only move forward and backward, the player can never move it upward or downward. Ludus rules are more flexible compared to paidea rules. The player may change the ludus rules and get involved in a different gameplay the game designer has intended, although the player might not be able to win the game. Gameplay emerges from paidea rules; but without ludus rules, there is hardly any gameplay. Paidea rules can be simple, but ludus rules can lead to complex gameplay. If players do not set the ludus rules while playing SimCity, the gameplay does not exist in the play session, because the players' actions are not oriented toward achieving a goal. In Super Mario Bros. 3, if players are only playing around with the world without having the intention to solve the level, gameplay does not exist, although the ludus rules are explicitly defined by the game designer.

If we view from the prism of narratology, it

Figure 4. A game model of gameplay and rules

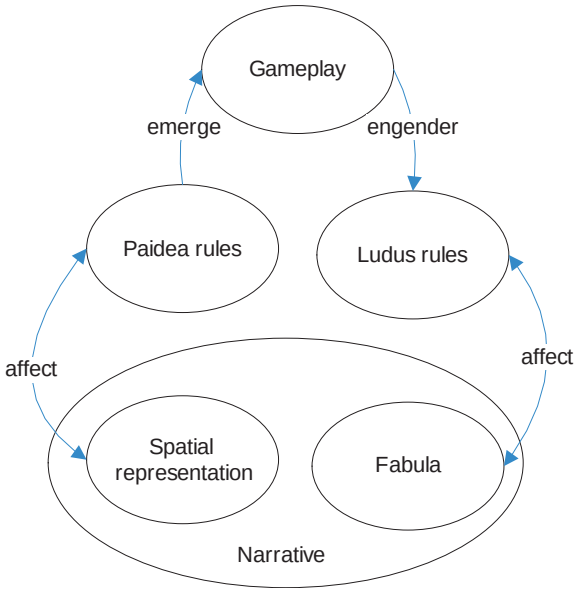


appears that games and narratives are quite similar, as computer games use narrative structures to organize their worlds. Nevertheless, games are not a mental image; they are a system defined by a set of concrete rules. Within this context, players can act freely as long as their actions conform to the rules. The chain of these actions can then be recounted in narrative discourse. In brief, the difference between narratives in computer games and linear narratives is that a linear narrative presents the facts in an immutable sequence, while a game presents a branching tree of sequences and allows the players to create their own story by making choices at each branch point (Crawford, 1982). However, there are sequences of events in games that do not become or form stories (like in Tetris). Therefore, not all games are interactive narratives; rather, some games can be interactive narratives, and these games can be used as a medium for storytelling. In fact, Frasca (1999) has attempted to relate paidea and ludus with narrative elements:

If ludus can be related to narrative plot, paidea can be related to the narrative settings. The ability to perform paidea activities is determined by the environment and the actions.

This statement is quite valuable in analyzing the relationship between rules and narratives in computer games. To perceive this subtle relationship, we would like to derive two narrative components from the narratological framework: spatiality, the space of the narrative; and fabula, the actions and events that might happen in the space. By applying these to the previous model, we have a more descriptive one (see Figure 5). The game space usually consists of compound worlds (Gingold, 2003). In most games, players travel through many locations, and enjoy the exploration of these multiple worlds and the movement between them. Compound worlds are collections

Figure 5. A game model of gameplay and narratives



of micro-worlds, which are governed by their own sets of paidea rules. These rules influence how the narrative world operates, while the movement of each world is marked by the changes in description and organization. Music, environment and, most importantly, rules change as the player moves between worlds. Ludus rules, on the other hand, are closely related to the narrative events in the narrative world. The player's actions are directly or indirectly affected by the ludus rules which, in turn, are changed according to the players' actions.

The relationship between narratives and gameplay is two-way. As narratives change, the rules become different. Evolving fabula can create new ludus rules, new sources of conflict and even new forms of gameplay. In fact, the best evolving stories can effectively change the rules of the game, something that probably would not be tolerated by a player lacking a story-driven reason. Rules in games need not be static. Narratives provide an explanation and meaning of the change of rules so that the virtual world is more believable. To exemplify this relationship, let us presume that Mario eats the mushroom and grows up in Super Mario Bros. 3. This event immediately triggers the creation of a ludus rule: to avoid being touched by monsters. It is very likely that in a certain point of the game, a monster will touch Mario and Mario will shrink. This event then again activates new ludus rules: to eat the mushroom.

THE GAME MODEL AND LEARNING IN COMPUTER GAMES

The game model that binds both gameplay and narratives reveals something about learning activities when players are engaged in game playing. In most modern computer games, the successfully playing of a game involves at least two types of learning: gameplay and the narrative. In this section, we look into these two aspects of learning in games and how we could use them in designing engaging learning software tools.

Gameplay-Oriented Learning

This relationship between rules and gameplay has a significant implication on designing pleasurable activities, such as learning. In the article *E-Learning as Computer Games: Designing Immersive and Experiential Learning*, e-learning software is interpreted as computer games, and several principles in designing interesting learning environments are outlined (Ang & Rao, 2004). If we look into the definition of paidea and ludus, it is not difficult to conclude that most conventional educational software is actually a loose type of game. It has paidea rules: Click the menu buttons and scroll the text with the mouse button, and so forth. Ludus rules are usually stated as the learning objective: to understand the concept of metamorphosis. Although the definition of game is much more intricate than just having these rules, this software could be seen as paidea or ludus games depending on the existence of an explicit goal. We are interested in investigating why it is not as engaging as commercial games with game rules.

The internal structure of a game can be characterized by its paidea rules, which can further be classified into two types: symbolic and semantic. Briefly, symbolic paidea rules explain the first layer of game interface — the input and output device interactions; while semantic paidea rules describe the narrative layer of interface. Obviously, the paidea rules of most learning software are symbolic, and do not impel learners to search for semantic meaning. The enjoyment of users should not be limited to symbolic paidea rules that define how users interact with computer devices. Learners should engage in gameplay by observing, hypothesizing, testing and updating the semantic paidea rules of the narrative environment. The pleasure of paidea should lie in the exploration of the virtual world and the discovery of paidea rules. Some learners find

some educational software interesting when they first play with it. They might have fun interacting with the mouse or keyboard. However, they soon will see through the mechanisms of the system: There is nothing more to explore. Besides, understanding the paidea rules does not let them plan for strategies to achieve the goal. Unlike Super Mario Bros. 3, where players play and observe the causality of their actions and the behavior of the spatial system, most learning software does not contain such qualities. The major "gameplay" of this software revolves around the reading of texts, since the paidea rules are oversimplified: Click and read.

The game designer not only has to design the paidea rules that define how games work, but also must define the goal of the game (ludus rules). We can further derive two kinds of ludus rules: micro and macro. Micro ludus rules contribute indirectly to winning a game, while macro ludus rules contribute directly. Computer games usually have macro ludus rules, which define the ultimate goal; while most micro ludus rules are either predefined by the game designers or created by the players during the play session. Oriented towards macro ludus rules, the player devises individual micro ludus rules in order to achieve the goal. For a game-based learning system, explicitly stated micro ludus rules can be important to scaffold learning. Micro ludus rules also function as guidance in the virtual world that steers players toward the learning objective. Learning objectives are presented as part of the narrative context. Instead of "to understand genetic configuration of animals," we can intrigue the learners "to defeat the monsters by breaking the genetic codes." Besides, this matches task-based learning, while each task is introduced as ludus rules.

Narrative-Oriented Learning

Narrative interfaces have been used in the game industry since its infancy and have successfully enticed a large portion of computer users for decades. Unfortunately, most educational software fails to take advantage of this highly effective design. Spatial design is obviously lacking, as most interfaces of conventional learning software adopt the metaphor of a book. The computer screen should not be a representation of a page of book, but a window to a new world. Learners look through the screen like through a window to a new spatial world of knowledge in which the images of real objects act coherently with virtual models (Morozov & Markov, 2000).

Like paidea rules, the interface of a game is doubled in an interesting way. First is the interface of the computer: the keyboard and the mouse. An additional interface is the narrative metaphor, which illuminates the narrative space in a new dynamic and interactive medium. The spatial design makes the first interface "disappear." Learners are not interacting with the keyboard or mouse, but with the story presented from the computer screen (Jaron & Biocca, 1992). Another issue pertaining to the spatiality of the software is that most educational software structures learning contents linearly, offers textual explanations and gives a particular spatial organization that does not reflect physical experiences. Learners should not regurgitate the context-free facts; rather, they expect to use knowledge in a contextually rich situation.

Apart from these, educational software does not offer narrativity to its users. There is hardly any action except for the clicking of menu buttons, which is hardly conceivable as stories. As Ryan (2001) has pointed out, players do not want to "gather points by hitting moving targets with a cursor controlled by a joystick"; they want to fight terrorists or save Earth from invasion by evil creatures from outer space. It is the same for learning software, which is also a type of game. Learners do not want to click the button to flip through the pages about genetics; they want to defeat monsters by analyzing and breaking their genetic codes.

LANGUAGE LEARNING AS EXTRINSIC AND INTRINSIC GAMES

We have elucidated two types of learning that might arise when playing a game. But how could these game designs be applied in language-learning software? Malone (1980) has propounded a motivational heuristic of educational games that comprises challenge, curiosity and fantasy. According to his interpretation of fantasy in computer games, two kinds of game design for learning are distinguished: intrinsic and extrinsic games. Intrinsic games rely on the understanding of the subject matter from within the game world, while extrinsic games rely on those external to the game world. Extrinsic games usually consist of a structured series of puzzles or tasks embedded in a game or narrative structure with which they have only the most slender connection. Intrinsic games build in challenges and activities that are more seamlessly integrated, more dependent on the narrative of the game. In brief, extrinsic games are used to attract users to learn a language, while in intrinsic games, the computer game itself becomes the learning activity. Therefore, the game design could be applied in language learning with two methods: the learning of the material as well as the learning of the game itself.

Extrinsic Game Learning

In extrinsic game design, the language learning is superimposed on the paidea rules of an existing game, resulting in new paidea rules. This may and may not engender new gameplay, depending on the quality of the bond between paidea rules and the learning material: Does the game world depend on the learning? In fact, this kind of game design is quite commonly used in language-learning games. Kana Warrior, for example, is a combination of first-person shooter and the learning of Japanese characters (Stubbs, 2003). In this game, the game world is dependent on the language skill. The player must improve in the language to make progress in the game. Often, extrinsic games are regarded as not as good as intrinsic games. They are somewhat the same as conventional e-learning software: Learners are not learning the paidea rules or the narrative of the system, but something external — the textual description of the subject matter. This is rather similar to reading books; we are not learning how the book operates, but the contents in the book. However, an extrinsic game could be effective in making some boring aspects of language learning interesting, such as rote learning of Japanese characters. The key factors that make game-based learning appear more interesting than typical e-learning software are its paidea rules and narratives. It is made enjoyable by binding paidea and ludus rules of computer games and language learning with narratives, creating an imaginary learning space that is engaging and immersive. It is very much like inventing a new form of book, in which every turning of the page yields a more interesting experience to the reader. The paidea is made fun, but it does not help the learning process.

Theoretically, almost every genre of games from the industry — racing games, board games, action games and so forth — could be used for this purpose. Extrinsic game design can be used to develop language-learning games for spelling, character recognition and vocabulary that require memorization and repetitive learning. Extrinsic design can be characterized as drill and practice, in which learning is context-free. Games-based learning has the potential for motivating drill and practice by offering an environment in which learners actually enjoy repetition. To fully utilize this design in learning, the following measures are suggested:

1. Investigate the type of game that target users enjoy. If learners get no delight in the particular genre of the game, it will not be successful in motivating them.
2. Allow learners to switch off the game in the middle of playing. Although the game is used totally as an extrinsic motivation to attract

learners, they might become interested in the subject matter and want to focus only on the learning content.

3. Provide strong narratives to create drama effect. Since the bond between paidea and the subject matter is rather weak, narratives are needed to reinforce the connection apart from creating drama effect.
4. The students must be familiar with the game's paidea rules. If the learner is unfamiliar with the game, the paidea rules must be simple so that the learning of paidea rules does not interfere with the learning of the subject matter.
5. Ludus rules must lead to the learning objective. Ludus rules should be stated distinctly and give guidance to the learner.

Narrative-Oriented Intrinsic Game Learning

Unlike extrinsic games, for intrinsic game design, learning contents are seamlessly integrated into either narrative or gaming mechanisms. Hence, intrinsic games provide two ways of learning as derived from the game model previously: narrative-oriented and gameplay-oriented. In narrative-oriented game design, players need to understand the virtual world, the event, the character and, most importantly, the story, in order to proceed. Like a book, the game mechanisms are trivial compared to the narrative mechanisms. The learning material is woven into the game as a story and, strictly speaking, the learning process is almost the same as for extrinsic games. Although the learning content is woven into a narrative context, the learning is explicit because the learners are shown the learning material in the form of text or graphics. Paidea rules merely define how learners should discover and read the material.

This differs from extrinsic game design in that it serves as a complete learning situation. This is most suitable characterized as a computer-based tutorial, in which information is designed to be presented in an effective and interesting way. Material is presented to the student in a narrative structure. This design is useful for learning the cultural aspects of a language by displaying pictorial or animated narratives of social settings, while reading skills could be fostered by exposing learners to textual narratives. In short, in this game design, the learning content or environment is designed as narrative, while paidea is for navigation. To fully utilize this design in language learning, the following measures are suggested:

1. Investigate what fantasy theme the target user is interested in. Narratives are crucial factors in this design. If the user does not like the fantasy theme the designer has chosen, it is likely that the design will be a failure.
2. The learning material is designed in the narrative context. The learning content is not presented as detached items of words or characters, but is connected to form a narrative.
3. The narrative should be able to stimulate the player to know what happens next. Curiosity is incited through the twist of narrative plots.
4. Goals are divided into several sub-goals to scaffold learning. Generally, the sub-goals are gradually presented to lead learners to the learning objective.
5. The control over the program is not as crucial as the control of the flow of the learning content.

Gameplay-Oriented Intrinsic Game Learning

In gameplay-oriented intrinsic games, players learn in a virtual world by interacting with the characters and events with languages. In this game design, paidea rules do not act as a simple interactivity that allows the player to discover predefined material. The learning material is embedded tactfully into the paidea rules of the

game. This is used as a game design that prevails, as it demands active experimentation rather than observation, of its subject material. It is also a way to explore, to test models and hypotheses, and to construct and acquire new knowledge in a way traditional media never can. At the extreme end of an intrinsic game for language learning, it would be a computerized conversation game. In fact, a project has been undertaken to develop a language training game in a fully 3D virtual world. The character in the game would be able to respond to what the learner speaks via Natural Language Processing Parser (Johnson, Marsella, Mote, Viljhálmsson, Narayanan, & Choi, 2004).

While in extrinsic games, the learning material is read and understood, in intrinsic games, the learning is experienced. In this design, the game designers — rather than implementing the material for the player to experience — implement a system of parts that come together to form the material in the hands of the player. This design can be used for subjects that require logical thinking, where information is not fact-based but rule- or process-based, such as the grammar of a language. It can be attributed to simulation. This simulation, however, is different from scientific simulations. This design is narrative and context-based, rather than simulating a scenario such as the lab experiment that is not relevant to real-life experiences. It provides an enticing problem-solving environment where students play an authentic role, exploring at will, creating their own ideas of its underlying structure and synthesizing strategies, which reflect their understanding of this structure. To fully use this design, the following measures are suggested:

1. The game should have explicit goals. Unlike scientific simulations, which have no goals, this design should provide clear and unambiguous goals.
2. It should have a narrative theme. This is true for three types of design; as, without narratives, the learner will be just manipulating words and alphabets.
3. The games should be able to stimulate the player to know more about the mechanism of the system by giving clear feedback.
4. The control over the program is crucial. The interaction will determine how the learner observes and infers the rules of the system,

Table 2. The summary of extrinsic and intrinsic game design

	Extrinsic	Intrinsic	
		Narrative-oriented	Gameplay-oriented
Paidea rules	Paidea rules are loosely linked with learning contents	Paidea rules define the navigation of learning contents	Paidea rules define the construction of learning contents
Ludus rules	Fictional learning objectives	Fictional learning objectives	Fictional learning objectives
Narrative	Narrative has little or no connection with learning	Narrative provides a context and contents for learning	Narrative provides a context for learning
Learning	Explicit, Context-free	Explicit, Context-based	Tacit, Context-based

which are also the subject matter.

Table 2 is the summary of game-based learning design.

AN ANALYSIS OF GAME-BASED LANGUAGE LEARNING WITH THE GAME MODEL

In this section, a theoretical view of games in computer-based learning is elucidated based on the game model and method of game-learning integration illustrated in the previous section. Two case studies are presented to demonstrate the implementation of "Slime Forest" in learning Japanese language, as well as how "Alien Language" is used for learning English, Spanish, French and German.

Case Study: Slime Forest

Slime Forest is a game similar to a role playing game (RPG) created to teach three sets of kana (Japanese characters). This is a summary of

Figure 6. Screenshots of Slime Forest (courtesy http://lrnj.com/sfa)

(a) the world map

(b) dealing with one Japanese character

(c) dealing with a katakana word

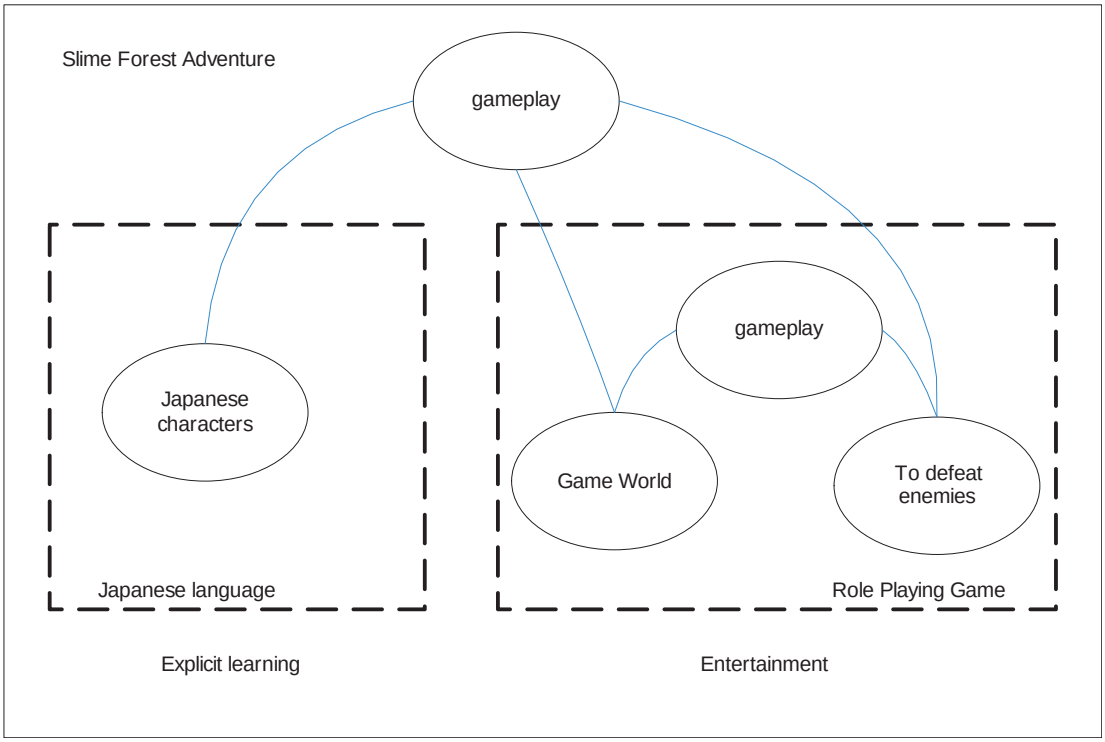
Slime Forest:

The game starts in a cave where the player is required to venture into the world outside to sell potatoes. The player will then be assigned, one by one, several sub-tasks and fight slime monsters in the forest to accomplish the tasks.

The game features two complete sets of hiragana and katakana, and 200 kanji, the Chinese characters. The aim of this game is to create a learning environment that provides a fun way of memorizing these characters, which are usually learned in a classroom via rote learning. In addition, players are also expected to learn some words borrowed from non-Chinese foreign languages, written in katakana. The instruction of the game is in English; therefore, this game aims at English speakers who wish to learn the Japanese language.

Basically, there are two types of activities: the world, and the battle in which the learning takes place. In the world map, the player plays a role of the game protagonist and explores different locations, such as the cave and the castle. The player needs to gather information about various missions by talking to people in the game. In the forest, the player will get involved in battles to fight slime monsters by typing the Romanized pronunciation of a particular kana that appears on top of the monster. If it is answered correctly, the player gains a chance to attack. Actually, the ultimate goal of this project is to design a game for everyone, even for those who are not interested

Figure 7. Theoretical view of Slime Forest



in learning Japanese.

We would like to examine the game and learning using the theoretical model of gameplay. Based on the definition of paidea and ludus, one can make learning Japanese a form of game by simply adding ludus rules; say, "Those who manage to write down 10 kanas the fastest win the game." In Slime Forest, users are more eager to learn the language because they have an interesting goal in mind: to kill the monsters, so they can progress in the game. However, adding ludus rules to a boring activity merely reduces the boredom. This form of game is not fun enough to engage learners for hours. What would have happened if the rules of "Who Wants to be a Millionaire" were reduced to answering the questions and wining $1 million without the "safe havens," "lifelines" and so on? Apparently, extra paidea rules are added to make the game really interesting.

It is noted that without the learning part, the game is a complete RPG game system (see Figure 7). The learning of Japanese characters is integrated, though in a loose manner, with the paidea rules of the battle system, resulting in a new form of gameplay oriented toward the ludus rules of the RPG. Besides, the integration of language learning and entertainment would not have been successful if not for narratives. The narrative in Slime Forest has at least two functions: to introduce fantasy learning goals (to fight monsters and save the princess) and to project the game as a complete virtual space with characters and events that retain the learner's interests. The narrative gives an explanation why players have to perform certain tasks. It is unlike certain educational software, in which the user is rewarded with a game playing session after completing learning tasks. By applying a well-integrated narrative, the learning and the game are bound together. Narratives also increase the urgency that pushes learners to complete the learning tasks.

It is concluded that the RPG game used in Slime Forest serves only as an external motivation. The integration of game and learning is extrinsic. It by no means aids the users in understanding the language. Some educators argue that games should not be used merely as means to motivate students to learn, and that play and learning must be mutually constitutive (Jenson, 2002). Indeed, learning should be made self-motivating, so learners are willing to learn it voluntarily. However, we should realize that not all aspects of language learning are internally motivating. Some are boring and difficult, and we need to help learners learn. In fact, if the learning was motivating, people may have learned them on their own and we probably need not put them in the curriculum (Prensky, 2001).

Case Study: Alien Language

Alien Language is a game distributed over the Internet used to support the teaching of "parts of the body" in four modern languages: English, French, Spanish and German. This is a brief summary of Alien Language:

The aliens are on a mission to collect creatures from around the galaxy for the alien zoo. You need to help them transport the creatures, label the specimens and cure the sick aliens.

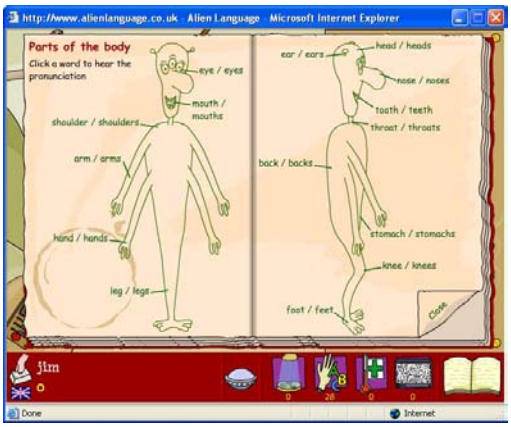
The motivation of this project is to create supplementary material for foreign language education based on a particular topic. The key aspects of language learning — spelling, grammar and sentence construction — are the focus of this project. The game can be used in many different combinations; for example, learning English for German speakers, learning Spanish for French speakers and so forth. It assumes that learners have known the basic vocabulary and sentences on the target language. However, if they need help, they can get a translation of the game instruction in their native language by pressing the control button.

Basically, the learners have three major tasks.

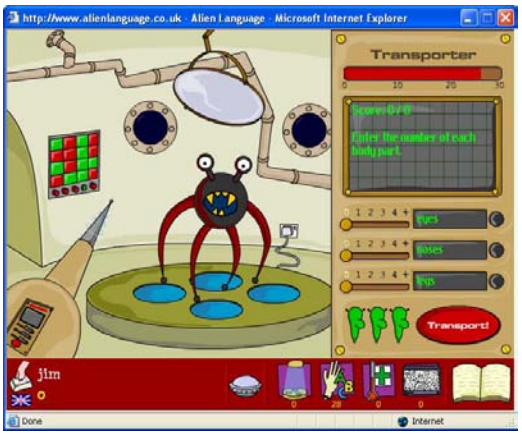
First, they need to record the number of each body part the alien has and transport it to the alien zoo. Second, they will label the specimen in the museum by typing the name of a body part. Third, they need to construct sentences from a set of given words to identify the medical condition of the alien. The game includes a trivia game — resembling the popular "Who Wants to be a Millionaire" game — that tests the understanding of the learners. It also contains a simple dictionary of body parts.

One of the best ways of learning a language is to use it in daily life. When interacting with people, we are actively receiving what people say, reflecting the meaning in our mind and constructing sentences. Different ludus rules could be introduced to create different scenarios to the language learning environment, hence, creating a game-like learning activity. If we examine Alien Language with the game model, we could see that the learning is designed to be part of the paidea rules. Take the hospital activity, for example: It is actually a micro version of a real-life conversation environment, where the player constructs

Figure 8. Screenshots of Alien Language (courtesy www.alienlanguage.co.uk)



(a) the dictionary

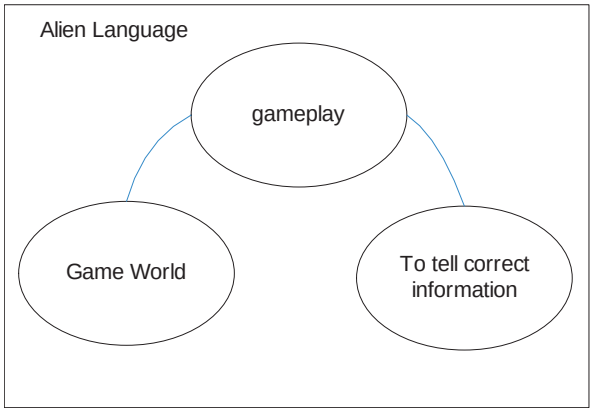


(b) the transporter

(c) the body museum

(d) the hospital

Figure 9. Theoretical view of Alien Language



sentences from words, although the sentence construction activity is far less complex than the real-life conversation.

Figure 9 shows a similar gameplay model as shown in the previous section. No external learning material is imposed to this internal structure of game. The paidea rules of the game are, in fact, the grammar rules that the player needs to learn in order to proceed in the game. Predefined ludus rules exist for each activity: to construct a meaningful sentence based on a given context, to construct the word and to count the number of each body part correctly. The use of narrative is also obvious in this case. The game projects a fantasy space with imaginative characters. This creates a more experiential learning environment, as the learners are put in a role within the narrative space. The design is coupled with narrative so the learners are not just manipulating the words and grammar rules, but are solving context-based problems and overcoming challenges.

Unlike Slime Forest, the learning happens in Alien Language itself. The players learn to play the game, and at the same time, they learn the language. The learning happens internally in the game, as the learning content is an integral part of the game structure. It is a gameplay-oriented intrinsic game design. Instead of learning something external to the game, the learner of Alien Language plays an authentic role and carries out interesting tasks.

Cross-Case Analysis

The two cases are examined together to find out what elements differentiate them and what they have in common. The observation is done from two perspectives: ludology and narratology, to derive something about computer game designs for learning. First, it is noted that even if the language learning part is removed from Slime Forest, the game is still a complete RPG on its own. The language learning is superimposed on the paidea rules and can be easily changed to something else, such as answering mathematics questions. Therefore, paidea rules of the RPG are used to attract learners into the learning environment. For Alien Language, however, this is quite different, since

the paidea rules are actually designed specifically for language learning. The learning contents cannot be swapped without significant modification of the fundamental structure of the game.

Second, both have clear and explicit ludus rules that bring about gameplay. The learning objectives are not stated explicitly as the learning of a language. Although both games use fantasy goals, such as defeating monsters and transporting aliens, the goals will eventually lead to language learning, since without leaning the specific language, the goals can never be achieved. Third, both games make full use of the narrative metaphor in designing the user interface. The games are designed as a fantasy world with narrative events. They also feature explicit characters that interact with learners to provide challenges: a fantasy narrative explanation of ludus rules. Some characters in Alien Language also give guidance to the learners regarding language contents. Actions taken by the learner in the game are conceivable and can be recounted as narratives. The following table summarizes the case studies.

DISCUSSION AND FUTURE DIRECTION

By studying ludology and narratology, we are able to derive something on how these theories are useful in designing language-learning software applications. It is maintained that computer game theories provide a better framework for designing language-learning software tools, making the experience of learning more immersive and engaging. Computer game-based language learning is expected to be better than its traditional counterpart from two perspectives: learning effectiveness and motivation. It is more effective in the sense that knowledge is constructed instead of being transmitted, especially for intrinsic game designs. It is also motivating, where it challenges the learners, intrigues their curiosity and brings about fantasy. However, an empirical study needs to be conducted to verify the advantages, and these results should help guide designers of educational games to consider how to effectively balance the demands of motivation and learning. Moreover, implementing game-based learning in light of language education needs detailed studies on the

Table 3. Summary of cross-case analysis

	Slime Forest	**Alien Language**
Paidea rules	The learning material is loosely incorporated as part of the paidea rules of the RPG	The paidea rules are carefully designed based on the target languages
Ludus rules	Ludus rules are fantasy objectives, which are not related to the learning (killing monster has nothing to do with learning Japanese)	Ludus rules are fantasy objective, which are related to the learning (reporting illnesses is related to languages)
Narrative	Like paidea rules, the narrative serves no purpose in the learning. It merely makes the game appear more interesting.	The narrative presents a context in which learning takes place.

nature of language learning, which could be approached from linguistics, psycholinguistics and sociolinguistics. Each of these fields seems to pose an insightful view of language acquisition from different stands. We believe that the understanding of how a language is acquired and learned, either by an infant or an adult, might open up a new door for a more novel use of computer games in language learning.

SUMMARY

We have analyzed two educational games from the perspective of ludology and narratology that explain two important types of game design in language-learning applications. In Slime Forest, computer games are used to attract the learner to learn a language aspect, while in Alien Language, the computer game itself becomes the learning activity. By analyzing both cases with computer game theories, we are able to understand them more closely, and thus derive a better principle of designing learning software based on computer games. Both extrinsic and intrinsic games are suitable for language-learning designs, although the latter is more desirable. It not only creates an engaging learning environment, but also an experiential one in which learners experience the knowledge first hand instead of being told.

REFERENCES

Aarseth, E. (1998). *Allegories of space: The question of spatiality in computer games.* Retrieved March 2005, from www.hf.uib.no/hi/espen/papers/space/

Ang, C. S., & Rao, R. K. (2004). *E-learning as computer games: Designing immersive and experiential learning.* Pacific Rim Conference on Multimedia, Tokyo, Japan.

Barrett, M. (1997). *Irreconcilable differences: Game vs. story, Gamasutra*. Retrieved March 2005, from www.gamasutra.com

Crawford, C. (1982). The art of computer game design. Retrieved March 2005, from www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html

Frasca, G. (1999). Ludology meets narratology: Similitude and differences between (video) games and narrative. Retrieved March 2005, from http://www.ludology.org

Frasca, G. (2001). *Video games of the oppressed: Video games as a means for critical thinking and debate*. Unpublished master's thesis, Georgia Institute of Technology.

Gingold, C. (2003). *Miniature gardens & magic crayons: Games, spaces, & worlds.* Georgia Institute of Technology

Jaron, L., & Biocca, F. (1992). An insider's view of the future of virtual reality. *Journal of Communications, 42*(4), 150-172.

Jenson, J. (2002, April). Serious play: Challenges of educational game design. *Proceedings of the AERA Annual Meeting,* New Orleans, LA.

Johnson, W. L., Marsella, S., Mote, N., Viljhálmsson, H., Narayanan, S., & Choi, S. (2004, June). Tactical language training system: Supporting the rapid acquisition of foreign language and cultural skills. *Proceedings of the STIL/ICALL 2004 Symposium on Computer Assisted Learning.*

Juul, J. (2002). The open and the closed: Game of emergence and games of progression. In F. Mäyrä (Ed.), *Computer game and digital cultures conference proceedings.* Tampere. Tampere University Press.

Lindley, C. A. (2002, June). The gameplay gestalt, narrative, and interactive storytelling. *Proceedings of the Computer Games and Digital Cultures Conference,* Tampere, Finland.

Malone, T. W. (1980, September). What makes

things fun to learn? Heuristics for designing instructional computer games. *Proceedings of the 3rd ACM SIGSMALL Symposium and the First SIGPC Symposium on Small Systems.*

Morozov, M. N., & Markov, A. I. (2000, December). *How to make courseware for schools interesting: New metaphors in educational multimedia*. International Workshop on Advanced Learning Technologies: Design and Development Issues. IEEE Computer Society.

Murray, J. J. R. (1978). *A history of board-games other than chess.* New York: Hacker Art Books.

Prensky, M. (2001). *Digital game-based learning.* New York: McGraw Hill.

Ryan, M.-L. (1994). Immersion vs. interactivity: Virtual reality and literary theory, postmodern culture. *Postmodern Culture, 5*(1). Retrieved March 2005, from http://muse.jhu.edu/journals/postmodern_culture/v005/5.1ryan.html

Ryan, M.-L. (2001). Beyond myth and metaphor — The case of narrative in digital media. *The International Journal of Computer Game Research.* Retrieved March 2005, from www.gamestudies.org

Stubbs, K. (2003). E-learning: Kana no senshi (kana warrior): A new interface for learning Japanese character. *The International Conference of Computer-Human Interaction (CHI) 2003 Extended Abstracts* (pp. 894-895).

This work was previously published in User-Centered Computer Aided Language Learning, edited by P. Zaphiris & G. Zacharia, pp. 1-21, copyright 2006 by Information Science Publishing (an imprint of IGI Global).

Chapter 3.16
VIPER:
Evaluation of an Integrated Group VoiceIP Software Application for Teaching and Learning in Higher Education

John Beaumont-Kerridge
University of Luton Business School, UK

ABSTRACT

Recent developments producing new Internet conferencing (IC) and multipoint desktop conferencing (MDC) systems have emerged, which may supersede text-based and audio/video conferencing (AVC) software. The newer IC or MDC systems also integrate interactive tools and have the advantage of operating at a fraction of the cost when compared to AVC systems. Communication by face to face methods are important within the learning process, but can online methods that incorporate sound, video, and integrated online tools be as effective? AVC systems within higher education (HE) have been available for some time although the quality of such approaches, however, has been open to question. This chapter evaluates an exploratory study of one MDC application, "Voice Café," in a higher education, business school setting. For commercial distinctiveness, the academic application of this software was called "VIPER" (voice Internet protocol extended reach). Consideration is given to the software itself in terms of its features, pedagogic aspects, and how students and faculty viewed its use.

INTRODUCTION

Recent developments producing new Internet conferencing (IC) and multipoint desktop conferencing (MDC) systems have emerged over and above the text-based discussion and audio/video conferencing (AVC) software. The major differences of the newer IC or MDC systems are the integrated use of interactive tools and their advantage of operating at a fraction of the cost when compared to the AVC systems. Communication

by face-to-face methods is important within the learning process, but can online software that incorporates sound and video and interactive tools be as effective? AVC systems within higher education (HE) have been available for some time. The quality of such approaches, however, has been open to question when compared to face-to-face methods (Knipe & Lee, 2002). Compatibility issues exist between competing AVC commercial providers, which have limited deployment, they are relatively expensive, and disagreement has also been voiced about how AVC should be used (Laurillard, 1993; Mason, 1998).

Easy access to education is not available for everyone. As a result, pressure is upon teaching providers to develop different methods of communication to extend the "reach" from an institution to students. Computer conferencing using text-based systems is widely used within online courses although learners and tutors have noted problems (Cartwright, 2000; Harasim, 1997; Salmon, 2000).

This chapter evaluates an exploratory study of an MDC application, "voice café," in a higher education, business school setting. For commercial distinctiveness, the academic application of this software was called "VIPER" (voice Internet protocol extended reach). Consideration is given to the software itself in terms of its features, pedagogic aspects, and how students and faculty viewed its use.

VOICE CAFÉ AND VIPER

The business school became aware of the voice café software at the beginning of 2004. It was initially evaluated as a useful means of improving communication for staff, students, and between partner institutions. It was initially applied to the overseas MBA program. Because of the "newness" of this software and the alacrity of HE providers to search out new methods of communication, it was decided to call the academic application of this software "VIPER," short for voice Internet protocol extended reach. Figure 1 shows a screen shot of the VIPER, which has the following features:

1. **Voice to voice capability over analogue telephone modem:** For a participant to speak, similar to a "walkie talkie" participants depress the F9 or ctrl button continuously for others to be heard. Either speakers or headphones can be used with a microphone needed for speech. Important, however, was the feature that communications took place over an ordinary telephone line and a broadband connection was not needed, helpful for connections in many parts of the world do not have the advantages of broadband. The minimum requirement for the software was 33kb, which is well below the normal analogue telephone connection speed of about 40- 50kb, although in some countries speeds as low as 26k were recorded.
2. **Up to 25 synchronous connections:** The version used provided up to 25 simultaneous connections, although up to 500+ are possible. One connection could be one person, a group in front of one computer, or a computer in a lecture theatre. Where there was more than one person, a suitable screen or projector was needed, as well as speakers and a "moderator" to control the computer to allow for a full dialogue to take place.
3. **Browser capability with "follow me" functions:** The ability to combine voice and browser capability proved to be a most powerful feature. Not only could Web sites be accessed, but also materials for presentation within discussion could be viewed. The PowerPoint™ "publish and save as Web page" function proved to be most useful because of its ease converting PPT files to HTML files, which also provided an index

within the conversion process.

4. **"Hands Up" functions:** Due to the absence of a video of the student, the tutor is not able to "see" if a student wishes to ask a question. The hands up facility is activated by the student using the right mouse button over the participants icon and the five options of (a) ask a question, (b) make a statement, (c) thumbs up image, (d) thumbs down image, or (e) never mind. The last function enables the user to reset this feature. Activating this provides a small icon for each of the items (a) to (d), which is viewable by all. There is also an overview screen that lists who is waiting to ask a question or make a statement, and a sum total of "thumbs up" or "thumbs down."
5. **Text chat, group and private:** Two text chat facilities are available, group and individual. The former allows all participants to see the posted messages, which can be saved in either a text or HTML format. The individual messaging allows only the participant and tutor to see the message.
6. **Interactive whiteboard:** This tool was interactive for all participants with a standard palette for the variety of drawing tools available. One interesting feature was the ability to "capture" a browser image for interactive use with students. In addition, as would be expected with software such as this, browser compatible images could be loaded from the desktop.
7. **Tutor video out:** Albeit a frame rate of some 24 frames per second, only one image is available of the tutor and this all participants can see. The default size of the image is 150x150 pixels, although this can be enlarged. The minimal video capability was, however, one of the reasons for the very low bandwidth requirement. This was essential for operation in some parts of the world where it was known that only analogue telephone over poor standard lines would be available. In addition, it was noted early on that once the video had been set up, and a tutor's image was available, it could then be paused further decreasing the bandwidth requirement.
8. **Installation and network requirements:** The voice café software operates on the client PC via a "plug in" on Internet Explorer™ v6.0 or later browser software. It requires a PC with at least a Pentium III equivalent, appropriate memory, hard drive, mic, and headphones or speakers. Functionally it worked via servers in the U.S. and Australia. Installation took about three minutes on a broadband connection, but 30 minutes on a dialup. This was, however, a once only operation unless the software was updated, which was automatically sensed by the installed program. Setup in the home environment did not generally meet with problems unless the PC specification was below specification. Setup in an office or college environment, however, was more problematic due to the need for firewall ports required opening (9500 to 9509 outgoing traffic only). The major difficulty under these circumstances was being able to contact the appropriate member of staff. In almost all instances, once the network member of staff was contacted any communications' problems were resolved in a matter of minutes. What often caused the delays, often days or weeks, was the difficulty of contacting the member of the network staff.
9. **256 bit security encryption:** Previous experience by voice café designers were commercial considerations of confidentiality and outsiders "listening in" to conversations and documents under consideration. Although not tested, the software boasts this high level of security. The only drawback with this, however, was the inability to enter and show in the browser Web sites that required

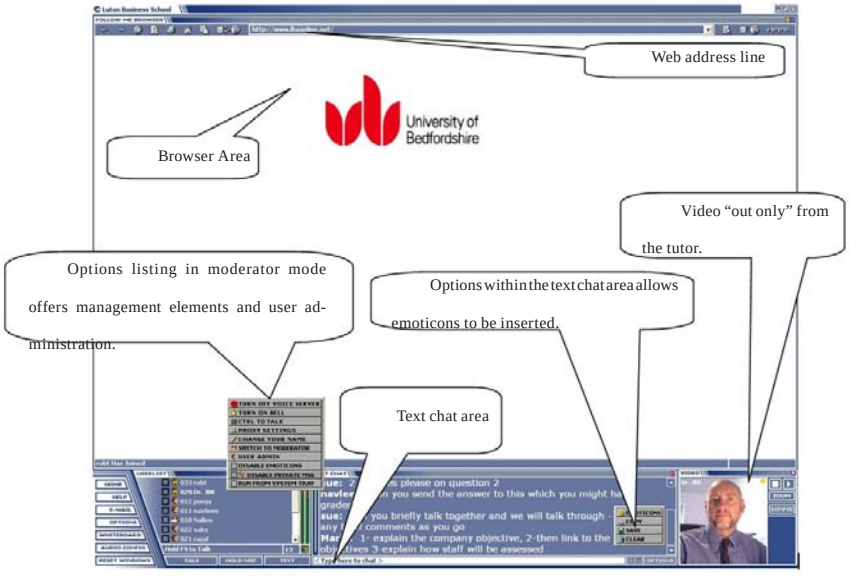
a username and password for access.

10. **Management functions:** The moderator has mute, boot, and banish functions if needed. Userful also was the ability to use multiple usernames and passwords, that is, one cohort could be assigned one username and password. This enabled a much easier management of student and staff access.

Pedagogic Considerations

Early models of learning suggested the "extended classroom" as a good design for online teaching and learning. VIPER would certainly enable this, either in the form of "one to many" or "many to many" with respect to tutors and students. Criticism exists, however, that online tutors and instructors come to the new medium from traditional backgrounds with skills that do not translate well (Schieman, Taere, & McLaren, 1992), an issue, which is still reported (Salmon, 2003). This is mostly based, however, on text-based online learning models and this MDC application. Because VIPER incorporates voice and basic IT skills, may need a shorter learning curve and as a result such difficulties may not arise. Even so, the online environment is dynamic with developments in design continually being added to the knowledge base of this domain. Laurillard's important work in 1993 has been re-evaluated, and this original taxonomy cannot easily be applied to group work (Britain & Liber, 1999). Indeed, Laurillard (2000) herself has added more in the light of recent developments. It is perhaps as a result of this and the variety of online tools available within the "blended suite" that a single framework for teaching and learning to which a majority can subscribe as best practice has yet to emerge. This exploratory evaluation of the VIPER is therefore set against participant experience, observation, and interactivity.

Figure 1. Screen shot of Viper software



Evaluation of VIPER

These were conducted with three student groups and faculty:

1. Six selected undergraduate business studies students.
2. A dissertation support group.
3. An MBA student group.
4. Faculty staff.

Selected Marketing and Business Undergraduate Students

Of six students, four were connected via computers that were "on campus," whilst the remaining two were connected via external connections to the University. Of particular interest were set up issues, learning curves of participants, and techniques of verbal discussion online. The two sessions began with the tutor outlining a topic and then inviting discussion. The set up and installation phase went well without event for all of the students. Anticipated firewall problems did not arise, and all equipment and software functioned appropriately. As a consequence, the use of the software was reliable.

Initially, students and the tutor engaged only in verbal discussion, and in the first session text chat used only very occasionally while a participant was talking. In the second session, however, particularly the students who considered themselves IT literate used the text function much more extensively. At one stage the tutor did ask the students to stop using the text function since they were using it to organize a forthcoming social function. This did appear to be similar to a face-to-face session where the tutor was asking the students in the "back row" to be quiet!

There are a number of tools in the VIPER suite available to both participants and tutor. It was not unexpected that only the voice, text and browsing functions were used but these were, toward the end of the second session all were used extensively. When asked, after the sessions had been completed, participants reported that they found the software acceptable, but two of the six stated that they would prefer face-to-face lectures. One commented:

I would not like to study in this way, being on my own in front of a computer. I would miss my friends.

Participants in the demonstration classes reported the software was "interesting." The quality of sound was reported as being acceptable. There was a request, however, by some "to slow down," since speaking quickly prevented comprehension. This appeared to be overcome in the first instance by asking participants to speak relatively carefully, but was not noted as an issue in later sessions. The sessions were deemed by the students and staff, however, as being successful, with the lectures having been conducted successfully.

Dissertation Student Group

This postgraduate group was completing their MSc in business related studies, and two were abroad for the data gathering exercise (Germany and India). Initial meetings were held face-to-face to begin the study process and resolve issues of subject matter and research directions. The set up of the software was straight forward and uneventful for all of the students. This was unexpected since all participants reported a mistrust of IT in general, reporting critical failures at some stage in their academic lives. The dissertation was a major academic stage, and seemed to heighten their lack of confidence in IT.

Regular meetings were arranged, at which it was intended that all participants would login at a set time. Students produced Word™ documents for discussion, for example literature reviews, questionnaires, and the group and tutor would discuss issues that arose. The tutor printed the

Word™ document in an Adobe Acrobat™ format, and then placed them on a Web site so they could be accessed by the software and viewed by all in the group. In one instance, while the group was actively engaged, the student who was in India e-mailed a document (draft questionnaire), which the tutor converted "on the fly" and uploaded to a Web site for group discussion. It was noted that adequate IT skills on the part of the tutor are needed for this, not least the ability to "multitask" while the discussions are ongoing. After the sessions were completed, participants reported that they found the software relatively straightforward to use, and had the convenience of saving travel time. One participant stated:

I found VIPER to be really good for preparation, but for the really difficult issues that I did not understand, I really needed to meet my tutor to talk things through. It was frustrating, and I really needed to be in front of my tutor. (Student 9)

While the students were in the data collection phase, face-to-face contact was not possible. Although this might be preferable, learning objectives were achieved using VIPER. This student achieved good grades without the face-to-face communication, but none the less felt the need for this type of support and viewed VIPER methods for deep learning, secondary. A similar result found by Katz and Yablon (2003), where no significant differences in grade performance between Internet and traditional lecture based courses were reported, but attitudinal differences, however, was identified where students preferred off-line methods.

MBA Student Group

Twenty-two participants were involved in learning and teaching situation, and 12 were selected for the evaluation on a random basis. Although this study concentrated upon the use of VIPER, other in course tools and techniques were also being used. These included face-to-face teaching, Blackboard™ virtual learning environment, and bespoke CD-ROM materials. The students on the MBA course met for an induction session at the beginning of the course. The intention was to create a better environment for socialization processes to take place rather than using online communication methods.

Those who considered themselves IT competent found no problems with the set up or use of the software. The following statement typified this for one such participant:

Good. Simple to use and a good way to hold tutorials. I have also found it excellent for our study group to use weekly to keep in touch.

One student, due to work commitment was posted to Saudi Arabia during the course. He was able to set up his laptop from his hotel room and participate as effectively as others within sessions. Although positive statements were received about the software and its use, comments of problems in this category are interesting and provide a different perspective:

Examples of problems reported were:

1. **A problematic Internet service provider (ISP), (blocked port access):** This was resolved by IT support by informing the student to request the ISP provider of the appropriate instructions.
2. **A PC well below the required and notified specification:** Overcome by the student purchasing a computer that was a reasonably new specification and not seven years old.
3. **Smooth functioning not occurring at peak periods due to a "free" ISP provider:** Resolved by the student "upgrading" to a paid service.
4. **Some individuals reported difficulties**

with audio settings: This last aspect was resolved by providing a "set up routine," which students undertook when first logging into the system. This appeared in later sessions to resolve the problems.

Both moderated and un-moderated meetings were observed, and this provided an important collaborative mechanism for students. From these groups VIPER was reported as being very helpful for students to "stay in touch," and the level of academic use for the collaborative work varied from complex assignment oriented to social conversations. Better learning independence is noted as one of the higher order skills of learning sometimes characterized by the preference by students for un-moderated meetings to challenge the given "norms" (Salmon, 2000). Students appeared to require a need to develop "online" meeting skills, however, when they met through VIPER, which improved over time. Conversations were more relaxed in later meetings, with almost no problems being reported.

Where moderated group meetings were organized with tutors with no set agenda, requests were made for more information relating to assessment and other issues, for example, housekeeping matters such as the "font size" and "layout" of documents to be submitted. Both faculty and students felt that questions such as these improved the face-to-face meetings because much of the "trivia, but necessary trivia" had been resolved leaving the physical meetings to more important matters on the course. Students also considered this to be valuable, and saved considerably on travel time.

THE FACULTY PERSPECTIVE

Salmon (2000) emphasizes that the skills of the moderator are paramount in the success of an online course. Theoretically, the basic use of the system should not impede the communication of the group and tutor. This appeared to be the case within the MBA group. One tutor commented:

…It has improved my listening and understanding skills of what a student was doing. Sometimes there would be a silence, and at first I was trying to work out what was going on. Then I began to realize the difference between when the student was thinking and when they were writing down notes. It enabled me to be more effective about who to draw into discussion and when there appeared to be some delay. It is a different kind of skill. (Tutor 1)

In addition to audio from the perspective of interaction and the available online tools, VIPER appeared to provide subtle additional aspects of communication.

…The communication was surprisingly active. It was not just all one way, it engaged many senses, and I had all my things around me, which I could quickly refer to. I was able to set myself up because they could not see. (Tutor 3)

This tutor went on to say:

…One soon learnt to bring the students in by asking questions, searching questions. Some students cannot bear to have a silence; others can so noting those who had not answered and bringing them in later.

More extensive use of the system was made following requests from some of the students. Observations of the seven tutors involved provided the following main usages of the system, and can be grouped under three headings: (a) voice only with hands up indicator, (b) voice only with hands up indicator plus Web browsing, (c) voice only with hands up indicator, Web browsing plus the use of the whiteboard.

Level I: Voice Only and Hands Up Indicator

Tutors and students engaged in conversations about topics, using the hands up indicator for dichotomous questions of the group. Often this was used to bring contributions together to redirect the flow of a discussion, a technique described by Salmon (2000) as "weaving." There was a perception by the tutors that Level I use was straightforward and easy enough after the set up procedures were complete. The text chat facility appeared to be used very rapidly within all sessions, contained questions, brief replies, clarity, or confirmation of issues between participants while the session was progressing. It was a main supporting communications channel supporting the voice discussion. It also served as a tangible record post meeting via a "copy of text" function.

Level II: Voice, Hands Up Indicator and Web-Based Materials

The browser capability enabled Web-based materials to be presented in conjunction with the other tools. The materials were either Web sites relevant to the topic or learning materials prepared for the Web specifically for the session, normally PowerPoint™ files saved in an HTML Web format or pdf files. One tutor demonstrated the dynamic use of the system:

We were discussing an assignment case study that was due, and one of the students mentioned that because they were at work did not have the material with them. The assignment was converted to a pdf document on the fly, posted to a Web site, and then used in the discussions. (Tutor 6)

The VIPER system operates by sending the Web address to participants, and thereby providing the same image to all those connected within that session. One obvious advantage of this method is the saving of bandwidth. Some equivalent systems send an "image" to connected participants. This can require a high bandwidth due to the file size of the images being sent.

Level III: Voice, Hands Up Indicator, and Web-Based Materials Plus the Whiteboard

The whiteboard "background Web browser capture" capability means documents and diagrams can be presented for participants to interact as a part of the discussion using the integrated tools. Students were requested to "mark" or "write text" on the images to elicit participant viewpoints pre, post, and during discussion. The same group and tutor also used the "capture browser" function to create an image of the assignment upon which all participants could put questions and comments for discussion.

…It was interesting; I could go through the issues in a much more interesting way. It was much more interactive than in class. (Tutor 4)

On an operational level, significant variations existed. In terms of the skills ability of tutors, some required very little guidance, while others needed considerable support both prior to, during, and post sessions. Tutors as individuals it would appear will need to go through a learning curve process if this type of system is to be used within teaching and learning. As a result, depending upon the ICT of the individual tutors varying levels of support will be required, an issue reported by other studies (Garrison, 1998; Salmon, 2000, 2003).

An interesting aspect has arisen in the use of VIPER between tutors in distant geographic regions. International courses require much administration, management, and communication "staying behind" after sessions to discuss issues of management, quality, and pragmatic housekeeping issues. This does represent an important ad-

vantage over previous communications methods. These were prohibitive due to cost, or limiting since they were based on text-based systems, telephone, or face-to-face meetings.

SENIOR MANAGEMENT PERSPECTIVE

The view of senior management focused upon the strategic advantages of the software:

VIPER currently provides a USP, a unique advantage, which is very difficult in today's HE market place. It enables personal teaching at a distance. It also gives us further reach in terms of target markets. It has enabled us to win two overseas contracts against other HE institutions, and has dragged us to the attention in front of others…I am surprised, however, that other Universities have not used this type of software more extensively. (Dean, University of Luton Business School)

One contract, for example, to provide the MBA with the British Council in India, was won in competition against a number of other Universities. Although the distance learning packages were all similar, the inclusion of VIPER was unique and provided the significant advantage. Other issues raised by faculty concerned how the software was going to develop, over time with suggestions of video to improve the "comfort" of communication although the mechanics of large student numbers may be problematic.

SUMMARY

VIPER has shown itself to be a useful communications tool, for both students and tutors. It has currently given a unique advantage to the University of Luton Business School product portfolio enabling it to win contracts in competition with other Universities.

The technical issues surrounding its use were mostly straight forward, with the only problems appearing where some wanted to use the system from within a commercial network that had stringent firewall policies. These were overcome once the appropriate network staff were contacted and appropriate ports opened. The low bandwidth requirement also enabled VIPER to function where other, more demanding MDC software applications might not have been possible.

Overall, the general view of the students that have used the system was positive. This was particularly true of those who were distant geographically and needed to make contact on a regular or emergency basis. Students and faculty did need repetitive use to gain confidence and expertise in the use of VIPER. The system was used for a wide range of applications from important learning issues, very basic house keeping matters, to a social aspect of students keeping in regular contact with each other. Students as recipients of the teaching management approaches appear also to demand the highest level of interaction available, irrespective of the tutor capability or indeed perhaps the need of the subject matter. As a result, the student learning curve appeared to progress ahead of the faculty learning curve, which can be a source of stress for members of staff, with the consequence of training resource implications.

Staff also found VIPER useful, from the perspectives of teaching and learning as well as keeping good communications with staff in partnership colleges that were based abroad. Issues of pedagogic design were important in conjunction with the three levels of use of VIPER from voice only to full use of all tools and the interactive whiteboard. Good planning and preparation to provide the maximum benefit for the student experience also emerged as important from a tutor's perspective. As a result, VIPER would appear to offer another helpful communications

platform to add to the blended mix of online tools that are available for teaching and learning in an HE, business school context.

REFERENCES

Britain, S., & Liber, O. (1999). *A framework for pedagogical evaluation of virtual learning environments*. Report to JISC Technology Applications Program.

Cartwright, J. (2000). Lessons learned: Using asynchronous computer-mediated conferencing to facilitate group discussion. *Journal of Nursing Education, 39*(2), 87-90.

Garrison, D., (1998). Distance education for traditional universities: Part-time professional learning. *Journal of Distance Education, 13*(2), 74-78.

Harasim, L. M. (1997). *Interacting in hyperspace: Developing collaborative learning environments on the WWW.* Retrieved October 1, 2005, from http://www.worldbank.org/html/fpd/technet/mdf/edi-trng/har1.htm

Katz, J., & Yablon, B. (2003). Online university learning: Cognitive and affective perspectives. *Campus Wide Information Systems, 20*(2), 48-54.

Knipe, D., & Lee, M. (2002). The quality of teaching and learning via videoconferencing. *British Journal of Educational Technology, 33*(3), 301-311.

Laurillard, D. (1993). *Rethinking university teaching: A framework for the effective use of educational technology.* London: Routledge.

Laurillard, D. (2000). *Keynote presentation, Alt-C 2000 Conference, Umist.* Retrieved December 20, 2003, from http://www2.umist.ac.uk/isd/lwt/altc/programme/keynote.htm#laurillard

Mason, R. (1998). *Globalising education: Trends and applications.* London: Routledge.

Salmon, G. (2000). *E-moderating: The key to teaching and learning online*. London: Kogan Page.

Salmon, G. (2003). *E-moderating: The key to teaching and learning online* (2nd ed). London: Kogan Page.

Schieman, E., Taere, S., & McLaren, J. (1992). Towards a course development model for graduate level distance education. *Journal of Distance Education, 7*(2), 51-65.

This work was previously published in Cases on Global E-Learning Practices: Successes and Pitfalls, edited by R. Sharma & S. Mishra, pp. 123-134, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 3.17
A Pliant-Based Software Tool for Courseware Development

Marcus Vinicius Santos Kucharski
Pontifical Catholic University of Paraná, Brazil

Isaac Woungang
Ryerson University, Canada

Moses Nyongwa
University of Manitoba CUSB, Canada

ABSTRACT

The increasing importance of e-learning has been a boosting element for the emergence of Internet-based educational tools. As we move into the information age, tremendous efforts are made in the development of new information and communication technologies for educational purposes. The ultimate goal is to facilitate e-learning methodologies and acquisition. The chapter's contribution is in the area of open source software for technology-enhanced learning. First, we report on the capabilities of Pliant, a novel software framework for Web-based courseware development. Pliant' design features upon which e-learning capabilities are built are presented, showing that Pliant has some advantages over existing software, including flexibility, efficiency, and universal usability. A case study of the use of Pliant in the project "Multilanguage Database for Localization" developed at the CUSB School of Translation is presented. Second, we present Academia,[3] a Pliant-based courseware development Web portal, and its use in translation studies at CUSB.

INTRODUCTION

The widespread availability of Web-based educational systems and standard-based courseware systems, and their deployment in educational institutions, including educational community as a whole, has raised a clear concern regarding their "universal usability" scope (Hochheiser & Shneiderman, 2001). A thorough analysis of

the situation and informal discussions with "online teachers" and teacher educators show that Web-based educational tools are quite far from achieving their main goal—that is, being used by a wide distance audience in a cost-effective and educationally sound manner, and in particular, endowing Web literacy to both young, old, novice, expert, and end users with less computing background. This chapter reports on the capabilities of Pliant, a high-level and flexible programming language and Web development framework. It shows how Pliant can be used both for high-level programming and e-learning purposes, while meeting educational and software-oriented expectations. Academia, an example of an open-source, lightweight, Web-based courseware tool fully implemented in Pliant, is presented. This portal has been designed to help instructors quickly create, post, manage, and deliver Web-based courses and other e-learning resources. A case study of the usability of Academia at a Canadian institution is presented. This Pliant-driven application is meant to show the efficiency of the Pliant's framework as a supporting tool for e-learning methodologies and acquisition.

The chapter is organized as follows. First, we briefly introduce the main streams driving the development of Web-based educational tools, and situate Pliant in that context. We then present an overview of the Pliant approach in terms of language constructs—here, we present our view of the Pliant architecture, and its underlying design features upon which e-learning capabilities can be supported. Next, we discuss various e-learning capabilities of Pliant, while highlighting their relationships to some of the main topics of this book. These include:

a. A description of Pliant as a tool for consolidating e-learning methodologies/acquisition—here, elements for exploration, data management, teaching, communications, and users' management are presented;
b. A description of Pliant as a tool for learning programming languages and Web programming—a case study of the use of Pliant in a project entitled "Multilanguage Database for Localization," developed at the CUSB School of Translation, is also presented; and
c. A description of Academia—here, our focus is on showing how this portal has been used as a tool for supporting translation studies at the CUSB School of Translation.

We also introduce Co-op Web,[4] a Pliant-based Web portal developed at Ryerson University, Canada, used to administer the Cooperative Education and Internship Program.

Some shortcomings of our framework and how these can be addressed as future research themes are then offered, in the perspective of enhancing e-learning methodologies and acquisition. Future developments of our framework are also highlighted, and finally, our conclusion synthesizes our discussion and presents our final remarks on Pliant's e-learning features.

BACKGROUND

Web-Based Educational Tools

The exponentially increasing number of educational courses being offered over the Web has spurred a growing industry of software tools to assist in the creation of Web-based curriculum and in performing class management tasks. For this reason, Web-based educational tools and standard courseware systems are two main research and development streams in the field of e-learning. The development of these systems can be categorized in two complementary streams. The first stream is based on the traditional approach of "hardwiring" high-quality educational material items in the course content—that is, the learning content used by the student resides in the system. Well-

known examples of course management systems built upon this approach are Blackboard (2002) and WebCT (2002). The second stream is based on an adaptive approach, where a model of goals, preferences, and knowledge of each individual student is built and then used throughout the interaction with the student in order to adapt to the needs of that particular student. In this case, the learning content does not reside in the system, but in other distributed servers. Independently of the approach used, the majority of Web-based educational systems are based on technologies developed in the areas of hypermedia and intelligent tutoring systems (Brusilovsky, Stock, & Strapparava, 2000; Brusilovsky, 2001). The Pliant-based educational tool introduced in this chapter falls within the first stream. Pliant is a standalone and Web-based language, which encapsulates both the "human" and "computer" levels of thinking and coding programs. This unique privilege makes it an exceptional language, compared to any other existing one. It demonstrates that Pliant has a higher level of flexibility, adaptability, and integration, providing for higher software development capabilities and enhancements. Thanks to Pliant's HTTP (hypertext transfer protocol) server power, including server-side rendering, any mainstream Web browser should always be enough to access Pliant's documents, and by extension, Pliant e-learning materials. These features can be used to develop standalone Web portals for teaching/educational purposes, or to improve the current state-of-the-art e-learning development tools, while maintaining educational expectations, and economical, time constraints, and human resources limitations.

MAIN FOCUS OF THE CHAPTER

An Overview of the Pliant Approach

The Pliant project[5] was initiated in 1984 by Hubert Tonneau (Tonneau, De Mendez, & Santos, 1984). In his analysis of numerous software developments, Tonneau realized

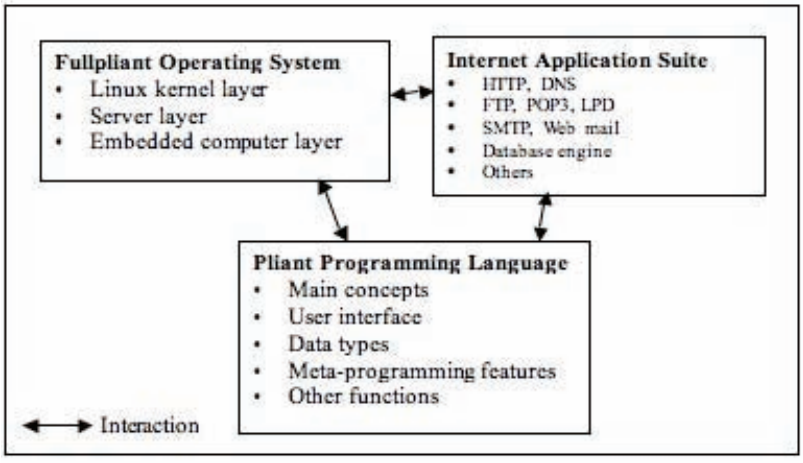
1. The lack of coherence between applications, libraries, and so forth often required a large amount of glue between relevant pieces of code.
2. It seemed impossible to conciliate high-level constructions allowing improved expressiveness and conciseness in specific contexts, with low-level adaptability allowing efficiency and optimized handling of exceptional cases.

From these considerations, the introduction of a new, efficient, and multi-level language with a flexible syntax and structure, which could be adapted to particular application contexts, seemed appropriate. The Pliant language is thus oriented towards efficiency, understood in terms of computational resources, as well as programming adaptability (De Mendez, De Mendez, Santos, & Tonneau, 2000). The main design structures of the language can be described as modularity, dynamic compilation, and full reflexivity, allowing for the redefinition of the syntactical, compilation, and code optimization rules. New application services have then been integrated at the language level (good examples are scheduling primitives and database management), hence suppressing usual gaps and interfaces between applications. From this point of view, an application is seen as a set of libraries, or even as a language extension possibly introducing its own syntactical changes. These applications may also be gathered into a coherent execution context, leading to an actual operating system, called *Fullpliant* (whose source code is a size of 4.2 Mb only). This framework can be executed in two different ways: as a program executing various servers (on Linux or Windows platforms), or as an operating system lying on top of a Linux kernel. Pliant comes with many pre-built servers including DNS (Domain Name System), FTP (File Transfer Protocol), POP3 (Post Office

Protocol version 3), HTTP, SMTP (Simple Mail Transfer Protocol), LPD (Line Printer Daemon protocol), remote execution, secured channel, and a database engine. The HTTP multi-site Web server provides the standard application interface. A powerful server-side dynamic page mechanism has been introduced, on which existing applications (Forum, photography correction and high-fidelity printing, Web-mail, etc.) rely, as well as additional HTTP-related servers (such as Web-based Distributed Authoring and Versioning–WebDAV). The limitation of the HTML (Hyper Text Markup Language)/Javascript scheme has led to the introduction of an enhanced extended Pliant browser, valuable as a state-of-the-art user interface for possibly distributed application software. Being used in an industrial context since 2000, *Fullpliant* is also concerned with security issues. The transparent integration in the dynamic page extension of signature and right verification mechanisms obviously demonstrates that security may be achieved without unnecessary additional programming complexity. Pliant (De Mendez et al., 2000) may thus be seen as a triad (see Figure 1): (1) the *Pliant programming language* and low-level libraries, (2) an *Internet applications suite*, and (3) the *Fullpliant operating system.*

The *Pliant programming language* is human oriented, that is, its syntax is trivial and strongly typed. Its expressiveness allows the user to program in a high abstraction level. The language is also reflexive, allowing the user to change the way Pliant parses and compiles expressions. In other words, users have a high degree of freedom to redefine the Pliant language itself, should they dislike a particular feature of the language or want to extend it (De Mendez, 1998). In addition to these meta-programming[6] capabilities of Pliant, the framework includes other modern programming principles, such as static typing, dynamic objects, lazy evaluation, reflective compiling, reference counting garbage collection, built-in debugger utility, scalability from low-level systems programming to high-level control languages, and an easy-following syntax. The Pliant compiler is dynamic and efficient, producing code, on-the-fly, as efficient as the best C compilers. The *Pliant Internet applications suite* consists of a set of servers, a database engine, data encryption tools, and a handful of Web application tools, such as the

Figure 1. Pliant architecture



HTTP server configuration tool, an online forum, Web mail, and printer configuration tool, to name a few. In the core of the suite is the Pliant HTTP server. It is in charge of hosting and dynamically translating Pliant Web pages, written in a subset of the Pliant language called the page format into HTML. The *Fullpliant operating system* (to be used by advanced users) has two main goals: facilitate the system administration of a set of computers, and make it easy to automate repetitive tasks. It appears from the above description that the two key design features of Pliant, upon which e-learning capabilities can be developed and supported, are the Pliant' intrinsic meta-programming constructs and the Pliant' ability to accommodate several multi-site Web servers.

Pliant' E-Learning Capabilities

E-learning can be defined as the use of network technology to design, deliver, select, administer, and adapt learning activities. Here, two basic types of technological solutions can be used: synchronous model (such as audio-video streaming and videoconference), and asynchronous model (such as hypertext publication). Existing e-learning management systems such as WebCT and Blackboard incorporate both models and corresponding services in different ways. However, due to certain limitations inherent to system stability, troubleshooting, cost-reduction, file format accommodations, Web browsers, customization, and others, none of these platforms includes a support to help manage the dynamics of e-learning activities. In an attempt to overcome these disadvantages, studies of management issues in e-learning environments has become critical for the success of Internet-based educational services. A relatively recent work on the management of e-learning environments has shown the effectiveness of using the workflow concepts (E-Workflow, 2003), techniques, and tools to help manage e-learning. Under this concept, the e-learning process is considered as a two-level interconnected process: the e-learning environment and the e-learning activities. The e-learning environment is further broken down into five different phases, each with its own set of tasks. These are:

1. **The conceptual phase:** Course subject, target audience, contents, budget organization, and so forth;
2. **The planning phase:** Details for the establishment and preparation of a specific instance of an e-learning action;
3. **The execution phase:** Period of time during which the students are active in the learning process; and
4. **The procedural evaluation phase:** An analysis of how the e-learning fulfilled its aims; additionally, the workflow e-learning environment model incorporates features such as improved efficiency, better process control (i.e., standardization of working methods), improved users service (i.e., greater predictability in levels of response to users), flexibility (i.e., ease of redesigning in line with changing needs); and
5. **The business process improvement:** Streamlining and simplification of processes.

The e-learning activities are concerned with the monitoring of actions and interactions among the above described phases. They are controlled by means of *learning objects*. The efficiency of an e-learning management system using workflow is measured by its capability in reusing learning objects. In this respect, workflow software and XML (Extensible Markup Language) are viable tools for describing learning objects. Any attempt to provide implementation techniques that could result in promoting the deployment of reusable contents or learning objects for e-learning purpose, while enhancing the integration capability of its underlying software platform, is therefore highly desirable for both the providers of educa-

tional services and the e-learning research community. Pliant seems to be a suitable candidate able to fulfill these requirements because of the versatility of the language constructs.

Pliant as a Tool for Consolidating E-Learning Management Systems Platforms

Any e-learning management system is driven by its underlying software platform (i.e., the set of programs that provide functionality to the application). At a human level, programs appear as algorithms, that is, a list of tasks, each being expressed by a single word or by a full sentence with subordinate clauses referring to subtasks. At a high (symbolic) level, a program is stated as a list of expressions. Each expression is a piece of raw data characterized by its semantics. At a low (i.e., code) level, a program is a set of instructions, where each instruction is a function call with a set of parameters. At this level, no ambiguity should remain.

A programming language is a bridge between the human way of thinking of an algorithm and the computer way of coding a program. Prior languages, including those on which the existing e-learning management systems are built, focus either on the semantics, but fail to be efficient at code level, or on efficiency, hence failing to be intuitive and easy to use by a human. To our knowledge, Pliant is the *first language* that makes the "single language" option a possible one by acting as a bridge between the aforementioned two levels. We assert that Pliant is the best one available, because it addresses this bridging goal at the highest level of flexibility and the best level of efficiency. This ability to write all code using a "single language" means better internal communication, time-saving improvements, load sharing, and shorter code. It also means reasonable scalability, adaptive user interface, easy switching from a closed software-like model to an open software-like model, more flexibility, customization, development power, strong design and high-quality program, low cost, dynamic highly reflexive compiler, less hardware limitations, adaptive hardware, and so forth. The list of goods is long. In order words, Pliant tries to bring as much expression power as possible, without impacting low-level code performance. These capabilities allow Pliant to be seen as a kit that greatly simplifies the distribution of software. In addition, Pliant can also offer e-learning potential, such as Internet-based learning and didactical requirements.

As a developed Web technology, Pliant is an Internet suite containing material needed to start an Internet site, including a database engine, a forum, a graphical toolkit, dynamic pages, and mail support. Therefore, it provides a suitable multimedia support to teachers and students, just as other proven e-learning systems, but with the added flexibility and adaptability of the underlying software and hardware platforms as pointed out previously. Pliant provides the choice to select elements according to the needs of teachers and students, and independently of their program of study. These elements can be divided into three interrelated groups, classified as: (1) elements for exploration and data management, (2) elements for teaching and communications, (3) and elements for users' management.

Elements for Exploration and Data Management

Pliant allows for a selection of various types of advanced Web browsers—Netscape, Mozilla, Internet Explorer, Konqueror—and is open to other better ones available. The choice of a browser depends on the security, portability, and computer power requirements. It also provides various types of servers suitable for Internetworking, such as HTTP, FTP, DNS, SMTP, POP3, Web mail, backup system, files browser, database engine, and so forth.

Elements for Teaching and Communications

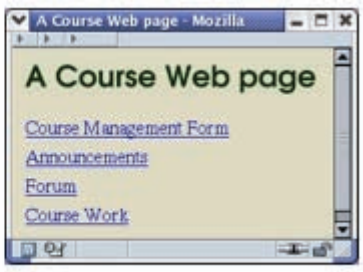
Pliant can be used as a tool for enhancing course management systems. It provides a flexible software support for a variety of learning processes such as the distribution of documents and communications through the Internet. Pliant's online forum application provides asynchronous communication between instructors and students. The simple structure of the language allows one to easily create course Web sites on which one can post course notes for anytime access by students. With some training in Pliant programming, novices can quickly move on to the development of their own handy, tailor-made teaching tools, such as course assessments, online tests, and an online grade book. To illustrate the above points, we consider a simple case of a typical course Web page written in Pliant, and its results as shown in Figure 2.

Several interface options are provided:

1. *Access to course documents*, in this case exemplified by a link to the *course management form* and the course work, but can be a list of documents, such as readings, lecture notes, schedules, syllabi, course management forms, organization of course projects, priorities, and details, and so forth. It also supports the import and export capabilities by means of inserted Web site links, allowing the instructors to gain access to a complete set of teaching tools provided by academic publishers, or to create a package of the course content that can later be imported into another course.
2. *Course announcements*, a key place to put daily, weekly, or monthly time-sensitive course information such as deadline changes, clarifications, remainder of upcoming class chats, schedules, important events and dates, and so forth.
3. *Forum* is a Web portal that behaves as a virtual classroom and lightweight chat. It enables users to participate in an online collaboration with students and instructors. As a discussion board, messages are posted to the board, and every permitted user is able to read the messages and reply to them. Like a bulletin board, one copy of the message exists, and only the course designer has the right to delete the messages. A forum is a tool that fosters communication and collaboration as a way to enhance course material. Several forums can be created simultaneously, providing for a frame for team working. Each forum is assigned a thread

Figure 2. Typical course Web page in Pliant

```
title "A Course Web page"
table columns 1 border 0
 cell
  link "Course Management Form" "cmf.pdf"
 cell
  link "Announcements" "announ.html"
 cell
  link "Forum" "forum.html"
 cell
  note "Course Work"
    title "Course work"
```



(i.e., a discussion session) so that all replies to a given message are contained within the same thread. Within a forum, a messaging program is implemented that allows one to send e-mail messages to the users who are members of the forum, and to keep track of those messages. As a collaboration tool, a forum allows its users to enter into a real-time discussion with instructors, students, and colleagues; to access the Web; and to engage in question-and-answer sessions. The option of considering grouping student lists into several small groups can also be applied to keep the conversation manageable due to the synchronous nature of the discussion forum panel. It is important to point out that collaboration sessions throughout forums are recorded by means of subjects and messages. The leader of the session (course designer) must start the recorder to create an archive.

Elements for Users Management

In an e-learning prospective, the user management capabilities of Pliant is mostly reflected on the ways that Pliant enables the instructor to manage the users in their course sites. This involves the following types of setting privileges: enrolling users in the course, which means that the user must already have an account; removing users from the course; and creating groups of users within a course with the right to modify groups. The instructor has the option of giving the group access to its own private discussion board, virtual classroom, group file exchange, or group e-mail.

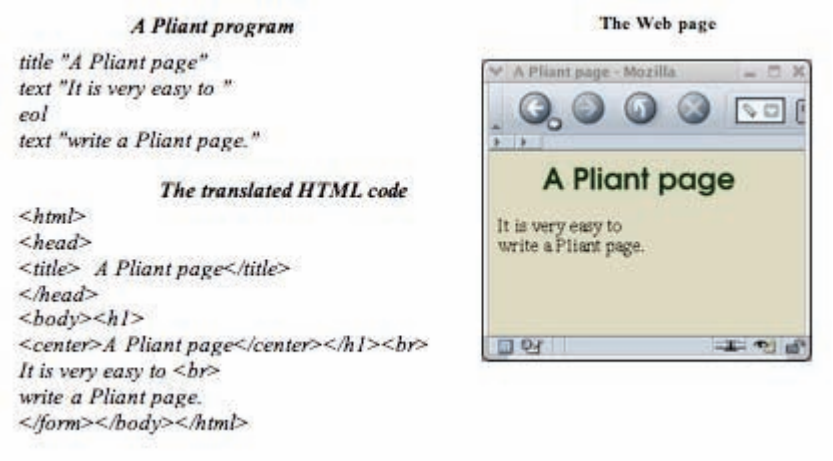
Pliant as a Tool for Learning Programming Languages and Web Programming

Due to the Pliant's meta-programming features, Pliant can be customized for learning and teaching purposes. Standard Pliant applications are browser based—that is, the programmer can host his or her Pliant program in the Pliant HTTP server and then interact with it using a Web browser. Browser-based Pliant programs have file extension *.page* and are written using Pliant's Web page programming instructions, called the *.page format*, an alternative to HTML/XML. Throughout the aforementioned interaction, the HTTP server works as follows: it keeps listening to requests from clients (i.e., other browsers on the Web); once it gets a request for a page, it translates the respective *.page* Pliant program into HTML and JavaScript code and sends it to the client (browser). If the client requests a plain HTML page hosted in the server, then the server simply sends it as it is. Hence, for the client, there is no difference; it is plain HTML/JavaScript coming from the server side. If the requested page does not point to a *.page* or static HTML file, then the Pliant HTTP server recursively searches for a file called *virtual_tree.page* in the path of the requested URL. This Pliant concept, called *virtual tree mechanism*, provides an easy mapping of data sets to URLs. Without it, HTTP options would be used—a not as clever and convenient solution. We relied heavily on such contrivance to implement one of Academia's[7] subsystems—the course Web content renderer.

To illustrate this simplicity and power of the Pliant language combined with the HTTP server, we now present short examples of Web applications written in Pliant. The Pliant program presented in Figure 3 illustrates how easy it is to write a simple static Web page with a title and some text. It shows a simplified HTML code generated by the Pliant HTTP server. The client will see this application as shown in Figure 3 (right side). As illustrated, Pliant has the capability of generating and caching online graphics when server-side font rendering has been requested.

The command title "A Pliant page" produces a page title; the command text, whose argument is a string, outputs text. Pliant provides a

Figure 3. A Pliant program and its resulting Web page



plethora of commands for writing Web pages. The interested reader should check the Pliant Documentation Initiative site (De Mendez et al., 2000). The next example illustrates how Pliant can be used to generate dynamic Web pages, such as a Web page for converting currency from Euros to Canadian Dollars. The programmer writes a page as follows:

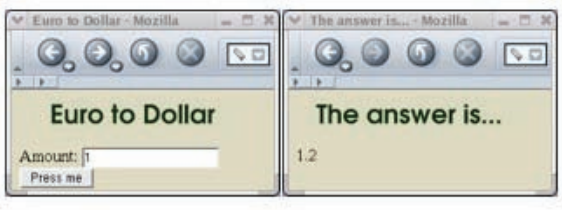
```
title “Euro to Dollar”
  var Float euro := 1
input “Amount: “ euro
  button “Press me”
title “The answer is…
text (string euro*1.2)
```

The user types in the amount in Euros he or she wants to convert (left of Figure 4). When the user clicks on the button, a new page, whose code is defined in the shadow of the button (i.e., indented with respect to the button instruction), will present the result of converting the input value from Euros to dollars (right of Figure 4). Notice that the function call (*string euro**1.2) transforms the numerical result of the expression *euro**1.2 into a string.

For programming languages development purposes, Pliant’s default syntax is lighter than others because of the following main features:

1. Many parentheses are implied by indentation,
2. The ‘:’ operator replaces some extra parentheses, and
3. There is no ‘,’ operator to separate the parameters of a function.

Figure 4. Dynamic Web page in Pliant



These features, combined with the meta-programming ones, make Pliant a language of choice for teaching the programming concepts. Because the purpose of this chapter is not on "experiencing programming languages," we will not elaborate much on Pliant's language specifications referred to the main concepts, user interface, data types, meta-programming optimizers, and other programming features. Interested readers can find detail information at http://old.fullpliant.org/pliant/language/.

In short, beginners can use Pliant as an interpreted language by writing small pieces of code and running them directly. Experienced programmers can run Pliant as a compiled language—that is, by writing efficient programs while using most high-level programming features of object-oriented languages and the expression power of logical programming languages. Pliant is also an ideal linker, in the sense that one can write different pieces of a project in different programming styles with all parts interacting.

Case Study on the Use of Pliant in the "Multilanguage Database for Localization" Project

In today's world of global operations and international technical communication, the Internet is fast becoming the primary port of call for information, education, training, and services. Consequently, more and more development initiatives go beyond local borders. Experienced professionals understand that to be effective, training must be done in the right language and with culturally appropriate resources and methodologies. In response, Canadian businesses, corporations, and universities have quickly become aware of the benefits of *localization* as one of the boosted agents shaping the new economy.

Localization can be defined as the process of taking a product and making it linguistically and culturally appropriate to a target locale (country, region, and language) where it is used and sold. This multi-layered process requires programming, linguistic skills, translation skills, cultural knowledge, and most importantly, an *e-learning development platform support.* It has been recognized as an important part of the educational program in Translation and Computer Science schools at Canadian universities and abroad, where students obtain basic education and training (both onsite and Web based) in both computational methods and localization techniques.

The increasing importance of terminology banks and translation memories in the translation process, a sub-component of the localization process, has created a need for developing language-based repositories for the purpose of using them in the localization training and practices. As a response to this deficiency, the "Multilanguage Database for Localization" project (Nyongwa & Aubin, 2004) was launched at the CUSB School of Translation, in collaboration with Ryerson University. The objective of this research is to develop a comprehensive database framework of languages which can efficiently support the localization training and practices. One subcomponent of this project entitled "Enhancement of the Pliant Language Database Engine" has been to investigate how the Pliant system, although not originally meant for language acquisition, can be used as a benchmark tool to support the user interface design of the aforementioned database framework, at least for the online localization training portion. In this context, thanks to its meta-programming feature, the Pliant language design has been altered and successfully tested to allow for customized content-building instructions. These later features were then used through *Academia—a Pliant-based courseware tool*,[8] to support the localization training in various capacities. Two scenarios are described later in this chapter to illustrate some of these capacities. The first scenario, described in the subsection entitled "Case Study of a Course Delivery," dis-

cusses a particular instance in the teaching of the "translation process" portion of the course in localization. The second scenario, described in the subsection entitled "Pliant System, Translation and CALL," illustrates how the "terminology" component of the localization course is taught using Academia.

Usability of Pliant in Teaching and Project Management Contexts

To further assess the usability of Pliant in teaching and project management contexts, we have implemented two Pliant-based applications. The first one is *Academia*, a courseware development Web portal, and the second one is *Co-op Web*, a management Web portal. In the sequel, we describe the first application in-depth, followed by a brief description of the second one. The reason is that the second application does not directly serve the purpose of this book.

Why Academia?

For instructors to carry on the mundane activity of setting course Web pages, two options are possible: design new course Web sites from scratch or resort to a courseware tool. However, such a tool is usually expensive and complex. The source code is usually proprietary, that is, it does not allow end users to further customize the code to suit their needs. Academia can address these deficiencies. The reader may then ask why one would use Academia and not another well-known courseware tool that provides every single feature an instructor can possibly fancy. Our answer: because as a Pliant-based application, end users can build on Academia's design and source code to develop more advanced and customized courseware materials, since there is no need to resort to different languages (as traditional approaches do) to develop applications that involve dynamic pages and databases.

Academia Features

Courseware products such as Blackboard (2002) and WebCT (2002) will take HTML documents, along with other media and resources, and quickly organize them into a framework specifically designed for delivery of Web-based courses and other learning resources. Frequently, they are used to complement traditional lecture-based programs. Courseware products are helpful to educators who are unfamiliar with programming, allowing easy integration of password protection, interactive activities, tracking of student progress, and so forth. Overall, the interface is fairly simple for the designer, as many use templates and wizards extensively to assist in course content creation. Step-by-step guides support creation of a range of components, from the course homepage, to bulletin boards, to quizzes and marking systems. When compared to the aforementioned courseware tools, Academia is a much 'lighter' application in the sense that it provides only the essential features required to set up a simple Web page for a course. Such a Web page may include links to a locally hosted copy of the course syllabus, assignment descriptions, instructor's lecture notes, and other materials.

Academia's User Interface Design

The design concept behind Academia's user interface was: *it should be as simple as possible, but not simpler.* In the first stage of the design process, a list of the functionality required of the system to accomplish the goals of the project and the potential needs of the users was prepared. From the instructor, manager, and students' viewpoints, this list included the major functionalities illustrated in the user-case diagrams presented in Figure 5. In the second stage of the design process, an analysis of the potential users of the system was carried out through discussion with instructors who already had previous experience

with other computer-based teaching tools, with instructors who had no previous experience with such systems, and with university students attending courses taught by the authors of this chapter. Typical questions presented to these individuals were of the sort:

- What would the user want the system to do?
- How would the system fit in with the user's normal workflow or daily activities?
- How technically savvy is the user and what similar systems does the user already use?
- What interface look-and-feel styles appeal to the user?

The answers were then compiled and cross-checked with the functionality list obtained in the respective analysis, resulting in a new minimal functionality set and overall look-and-feel style.

In the next stage, a site flow of the system that showed the hierarchy of the pages was developed. Next, prototyping and usability tests (Nielsen, 1993) were performed. In this stage, a proof-of-concept Web application was developed that showed the basic functionality set and content. Fast prototyping of the system was facilitated by the intrinsic features of the Pliant programming language. In regards to usability tests, the so-called *talk aloud protocol* (Wang, 2000), where you ask the user to talk about their thoughts during the experience, was performed via personal interviews with a small set of users—only instructors who volunteered to serve as guinea pigs during this tests—or 'looking over their shoulder' while they attempted to perform specific tasks with the system.

The ultimate goal of the Pliant software is "universal usability." As such, the Pliant user interface, or equivalently, the communication channel between the user and the functional elements of the system, has been intentionally made *text mode* based. This allows the user to focus on the task at hand and reduce the amount of overhead knowledge required to communicate effectively with Pliant and its underlying e-learning system (Academia). The function of the Pliant interface subsystem resumes in assigning user input to internal representations of Pliant's application and internal representations of the application to output that is comprehensible to the user. Thanks to the meta-programming features of the Pliant language and its reflective architecture, we believe that Pliant is compliant with the CALL aspects (i.e., the teaching and learning processes) with respect to online learning.

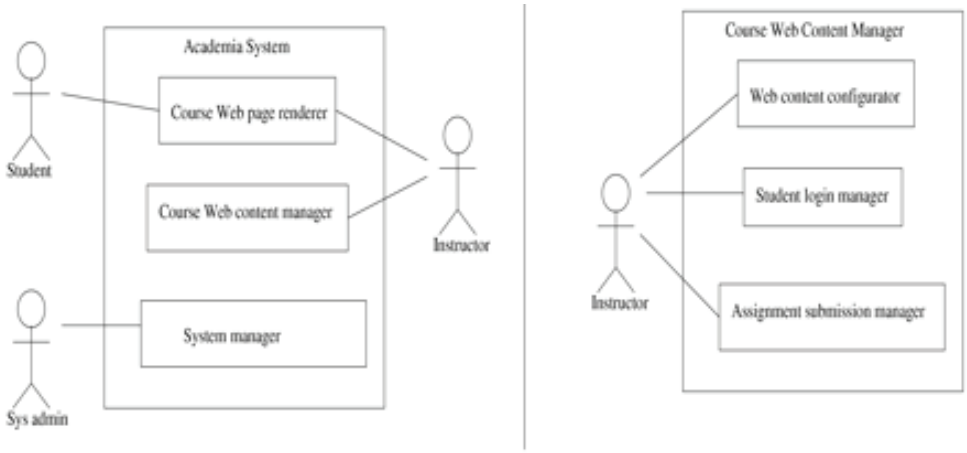
Academia's System-Level Capabilities

These are presented here by means of use cases and sequence diagrams. Such notations are commonly used in the area software engineering for describing a system without revealing or implying any particular implementation of the system (Booch, Rumnaugh, & Jacobson, 2005). The use case (left of Figure 5) presents Academia's subsystems and its actors.[9] The system administrator actor uses the *system manager subsystem* to keep track of the instructors registered in the system and the courses they teach. Instructors use the *course Web content subsystem* to set up the Web page of their courses. Once a course Web page has been configured, students can then visit the respective Web page, which is dynamically created by the *course Web page renderer.*

Course Web Content Manager Subsystem

Figure 5 (right side) depicts the course Web content manager subsystems. The *student login manager* subsystem allows the instructor to upload and manage the student's login access to the system. The *assignment submission manager* subsystem allows the instructor to view currently online submitted assignments and to download a compressed file containing all the assignment

Figure 5. Academia system-level architecture



submissions. The *course Web content configurator* subsystem allows the instructor to interactively configure a template design for the course Web page. The instructor can define his or her contact information, such as office number, office hours, lecture times, and locations; upload course syllabi, assignment descriptions, and lecture notes to the server; add an announcement board to the course Web page; and set up an online discussion board for the course.

Course Web Page Renderer

The course Web page configurator subsystem stores the configuration settings of a course Web page template design in a database. This database indexes courses by their term, year, code, and section number. The course Web page renderer uses this information and Pliant's virtual tree mechanism (explained earlier) to retrieve and display the Web content stored for a particular course. More specifically, suppose that an instructor whose username is *john.doe* has already configured the Web content for the following course:

***Term**: Winter*
***Year**: 2005*
***Code**: Phil 660*
***Section**: 001*
***Name**: Philosophy of Love and Sex*

If a client (browser) requests the URL[10] *http://academia.org/browse/mycourses/john.doe/,* then the Academia course Web page renderer would dynamically create a Web page that lists links to all currently hosted courses of instructor *john.doe.* It does that by using the virtual tree mechanism because there is no *index.html* file in the above location on the server. The Pliant program that implements the course Web page renderer resides in the *virtual_tree.page* file, located at the root of the path /browse/. Therefore, when the client requests the above URL and the server finds the *virtual_tree.page* file in /browse/, it stores in the internal variable *virtual_path* the remainder subpath */mycourses/john.doe/* and runs the *virtual_tree.page* program. For this particular case in which the above path starts with *mycourses,* the renderer parses the path and extracts the relevant informa-

tion needed for determining which courses to list on the dynamically created Web page.

A similar process takes place when a client (browser) requests the following URL: *http://academia.org/browse/courses/Winter/2005/Phil660/001/index.html*. However, no *index.html* (or *index.page*) file actually exists in the above location. Hence, analogous to what we explained above, the Pliant virtual path mechanism will again run the renderer. This time, the internal variable *virtual_path* will hold the sub-path */courses/Winter/2005/Phil660/001/*. Notice that this path provides all the information to locate the Web content for a course in Academia's database. The renderer parses this path to extract the course information (code, term, etc.). It then retrieves the Web content for the course from the database and dynamically creates a Web page for the course.

Towards Integrating Academia with Other Learning Management Systems

The initial idea underlying Academia's design was to provide an "*as simple as possible*," but not simpler courseware development and e-learning tool. As such, integration with more robust learning management systems (LMSs) was not a priority during the development. Notwithstanding, all database files used or produced by Academia are in ASCII text, and information is stored in XML-like data structures. As an example, below we list an excerpt of a user database:

```
<pdata path="/user/marcus/contact">7062</pdata>
<pdata path="/user/marcus/email">marcus.santos@mac.com</pdata>
<pdata path="/user/marcus/first_name">Marcus</pdata>
<pdata path="/user/marcus/homepage">www.cs.ryerson.ca/m3santos</pdata>
<pdata path="/user/marcus/language">English</pdata>
```

Notice that each entry in the database is encoded as simple XML code. Therefore, interfacing such data with an LMS would be straightforward. Pliant also includes many modern programming principles such as meta-programming, static typing, objects, reflective compiling, reference counting garbage collection, built-in debugger, clean syntax, and many pre-built components such as HTTP, FTP, SMTP, POP3 servers, a tree-based distributed data model, and a database engine. The intrinsic features of these core components can be used to support open Internet standards such as ECMAScript, thus, a-fortiori, to facilitate the compliance of Pliant with SCORM, but there still a long way to go to achieve this goal, and we have left it for future work.

A Case Study of Academia

This section reports on the findings from a study to experimentally compare some available Web-based learning tools used for the CUSB Certificate Program in Translation. These findings are discussed in relation to basic usability issues of Web-based tools, in terms of online course pedagogy, technological infrastructure, and students' perceptions. Our investigation attempts to justify the use of Academia as an e-learning methodology and acquisition tool.

The Context

Translation is a professional activity that requires both mental and physical settings, where profits and ethics must be met, just like in any other profession. Becoming a professional translator requires hard work, curiosity, open-mindedness, and experience. Based on these basic facts, the School of Translation at CUSB foresaw the importance of Web-based education in the mid-

1990s, then introduced a few online courses in its Translation program. By the year 2000, the entire Certificate Program, composed of 10 three-credit courses, was successfully launched. This program (referred to as the *TOP* program) was designed to accommodate the draft curricula for various areas of expertise within translation and the actual needs of the translation industry. It has been set up primarily for people who are interested in studying translation while maintaining their current employment. It was particularly aimed at those who work in remote areas and would find it impossible to attend classes at a local university. At each session, students may decide to take only one course at a time, or more, depending on their own personal timetable. A number of Internet-based courses, including *Localization,* were gradually introduced into the TOP program.

Technological Infrastructure

The CUSB infrastructure is made up of three servers, a dozen PCs for instructors, and a 10-PC laboratory for students. There are also three other high-tech laboratories and a multimedia center where on-campus students can work. Several programming tools and Blackboard (2002) are used by students and professors to develop and manage Web-related documents. Although this courseware tool was greatly appreciated by the students and staff, complaints were quickly raised, mostly on usability issues—such as long login sequence and difficult navigation. To address these drawbacks, Academia was used to supplement Blackboard.

Online Course Methodology

Unlike traditional courses, online courses are space and time independent. Meanwhile, in the case of the TOP program courses, a framework has been developed that meets the traditional division of an academic year—into sessions of 15 weeks each. Courses are delivered according to this division. The content of each course is organized as followed: knowledge review, lectures, practices, debates and discussions in the forum, exchanges of e-mails with instructors, and evaluation. Courses content is built into modules to facilitate individual learning. The evaluation consists of four formal tests, one in the fourth week, the second in the eighth week, the third in the twelfth week, and the last one in the fifteenth week.

Case Study of a Course Delivery

To demonstrate the capability of Academia as a support for e-learning methodology and acquisition, we have implemented an instance of a particular course entitled "Localization" on a specific problem: the problem of encoding/decoding the characters in a text file during the lifetime of the localization process. Localization is a process by which a product (in this case a text file) should be adequately adapted to the characteristics of each country and equipped with a presentation that is acceptable at least at the level already reached for its original locale. This process requires various types of operations involving the targeted language and written communication (localization of translations), as well as the supported technical infrastructure for data processing (software localization). Both types of localization sub-processes are technology dependent and are mandatory in a training program in Translation such as the TOP program. Here, we focus only on the software localization process. One of the major problems when running this process is the lack of a clear mechanism that ensures the identification of the encoding used to save/open any type of document, or to escape extended characters that are not supported by the targeted encoding technique. This issue is particularly important in this context since it determines translatable and non-translatable data. Among proposed solutions, XML has been proposed as a viable one (Savourel, 2000)

for the implementation of a multilingual solution. However, handling XML-like data structure files within the localization process is still a difficult task. To circumvent this difficulty, the Pliant-based capabilities of Academia are embodied in new standards or software processes such as XPath (Extensible Path), which provide several new features of dealing with data in general and localizable text in particular, based on the file format. Many state-of-the-art translator tools, available at *http://www.w3.org/*, are then used to apply the above mechanism in a concrete example. Depending on the power of the translator tool, the *Fullpliant*, and the type of XML file to be translated, the required steps are:

1. *Mark up the information in the text to be localized,* that is, find answers to the following questions: What text is translatable? Which language is chosen? Which local is referred to? What are the acronyms explanations? What constitutes the verbals and nominals in short sentences of the targeted text?
2. Using the Translator tool, *create an XSL template (document) with appropriate parameters.* This is achieved by writing a skeleton of HTML elements and by using the XSL style sheet commands to provide the text.
3. *Process the file to be localized by using the above template and by choosing the locale to work with.* Steps 1 and 2 are not always straightforward and need some good understanding of XML and XPath.

The following methodology was used. Prior to the localization course, students are given some basics in HTML and XML programming. They are introduced to the structural power of both languages and to Academia as a programming platform. Their attention is then focused on the simplicity of XML, and the way its features can be used to extract the translatable and non-translatable text in the file format to be localized. Students are taught, by means of examples, on how XML code can be written based on the file format and content, and then structured in a way that the translator can use it to generate the desired output. Students are then asked to compare the XML and corresponding HTML files. Then, the localization course is taught to students through different modules (process, Web site, project management, etc.), providing them with necessary material to tackle other steps of the translation process. At the end, students are asked to extract HTML and XML files in English and to localize them in Canadian French.

Findings

The separation of the content from the format allows the translation task to move faster while reducing the time allocated to pre-translation and post-translation processing. Due to the expressive power capabilities of the Pliant-based framework upon which Academia is built, students have gained a great level of confidence when integrating the localization information as a component of XML, rendering the translation process more achievable than ever. This integration would have been more difficult without the use of built-in and enticing capabilities for data processing provided by the Academia platform, which greatly simplifies the entire localization process.

Pliant System, Translation, and CALL

Even though the Pliant system was not originally meant for language acquisition, the flexibility it provides in terms of integration features makes it possible. In technical translation and specialty languages teaching, terminology is an essential topic that should be studied. The teaching of terminology in the context of the localization course using Academia has been addressed in a CALL-like methodology manner. For a *given text* that needs to be translated from one language

to another, the *steps* followed by the students, as well as the corresponding *Pliant-based methods* implemented to achieve these steps, are shown below.

1. Identify and establish a list of "appropriate" terms from the text to be translated. This task is achieved through the design of the Data Discovery Module (DDM). Here, some predefined language-based metrics are used to identify the aforementioned terms, and a Pliant-based user-interface is developed to extract these terms and stored them in DDM.
2. Ask the students to search for two or three contexts of the usage of each term from an established list obtained from various terminology repositories. This task is achieved through the design of the Data Context Module (DCM), where dictionaries and textual databases are stored. It mandates the implementation of a Pliant-based Web interface for data search and retrieval (using the Pliant mainstream servers) in these repositories.
3. Search for equivalent terms in the target language through available dictionaries and databases. This task is achieved by means of the Pliant-based Web interface for data search and retrieval described in Step 2.
4. Divide the text to be translated into individual sentences to be translated, and then allocate one sentence per student. This task is achieved through the design of the Division Module (DM). A Pliant-based script is developed to build this module.
5. Each sentence is translated by a student and sent to the discussion forum for revision by other students within the group. This task is achieved through the design of the Translation Module (TM). A Pliant-based forum is designed and implemented to handle the communication and transfer of information between all participants.
6. The revised sentences are put together again to create the translated text. This task is achieved through the design of the Assembly Module (AM). A Pliant-based script is developed to build this module.
7. The translated text is sent to the instructor for evaluation, verification, and validation. This task is achieved through the design of the Quality Assessment Module (QAM)—a set of Pliant-based evaluation and validation methods (or functions). This module also provides the potential for integrating the Pliant system in a distributed environment.

Co-Op Web

The Department of Computer Science at Ryerson University offers a five-year bachelor's program in Computer Science with a Co-op option (CSCC). The program requires the management of students' applications, admissions, career planning resources, course selection requirements, graduation requirements, financial requirements, and job placements, to name a few. To efficiently address these challenges, Co-op Web was developed using Pliant; it is currently use to administer the CSCC program and has shown great satisfaction from its usability' point of view.

Actual Management Aspects of Pliant after Development

Our current challenge remains to understand the experiences of instructors (and students) as they adopt Academia as a course management system and integrate it into their teaching (respectively learning), either in isolation or as a complement to existing learning management systems. Our plan is to study several patterns explaining how instructors/students experiment with individual features of Academia, facing both technical and integration challenges, and attempting to adapt Blackboard or any other sophisticated learning system to match their goals and practices. Aca-

demia has become "mission critical" in fulfilling some of the teaching and learning central goals: enriching the student/instructor learning experience and advancing access to resources for teaching, learning, and research. Academia's training programs are currently being handled throughout the university, in parallel with Blackboard training programs, where faculty and students can receive assistance with all aspects of Academia and Blackboard operations. Several upgrades have been done and are continuously done to the Pliant framework in order to solidify its underlying e-learning capabilities and integration features, both transparent through Academia. For example, at the departmental site, faculty members can now upload their final grades from a Blackboard grade book into a Web grade roster, instead of having to enter them manually. A tracker has also been set to report general bugs, feature requests, fixes, and other issues such as sensitive security problems.

In its current form, the dynamic compilation structure of Pliant offers many opportunities of experimentation for academic and industrial research as well. Thanks to Pliant's control over generated codes, academic and industrial fields used Pliant for physical modeling purposes. For example, in thermic equilibrium when injecting thermoplastic material in a mould, Pliant was used to update some models without having to recompile the system. In Mathematics, Pliant was used to re-implement some graph manipulation tools.

The integration of Pliant with SCORM, NLN, and IMS format learning objects is possible due to the dynamic compilation architecture of Pliant and its reflectivity. This option is currently under investigation. Another current application scenario is the use of Pliant to manage distributed systems. This has been achieved so far in some cases thanks to the meta-programming feature of Pliant and its tree-based distributed data model, which allows one to maintain a database of hardware available on each machine and of software to be deployed there.

To circumvent some of the drawbacks of Pliant's architecture, it is suggested that we look at some high-level optimization algorithms in academic literature and implement them in the Pliant language, the goal being to rewrite the entire framework in Pliant. This will facilitate the pliant deployment, its integration with other systems including learning management systems, and its ability to embrace and extend existing services, as well as enhancing Pliant's robustness in terms of code generation and reusability.

FUTURE TRENDS

The Pliant *.page* mechanism, which we have used to implement Academia, has proven very convenient for quickly prototyping and writing simple user interfaces. However, the current (and future) trend is to provide programming languages and Web application development platforms rich in interactive features, such as graphical user interface elements, to create dynamic, nice-looking, and functional user interfaces. To this aim, the Pliant team recently started the development of the Pliant browser. The Pliant browser consists of a new language (an extension to the *.page* language) for developing Web applications and an HTTP "bridge" for translating the Pliant browser interface to HTTP/Javascript.

For the future, once the additional features provided by the Pliant browser are in place, we plan to give Academia a new makeover on its user interface and features, such as a quiz/survey tool, and added flexibility to course Web page design and "look-and-feel." The style sheet mechanism used by the Pliant browser should greatly facilitate the implementation of a more flexible styling mechanism for Academia. Ultimately, Academia will stand as an example of how end users (teachers) of Pliant tend to become developers of Web-

based software solutions. An instructor with little or no programming background can smoothly migrate from a user of a simple system to a programmer of also simple pragmatic systems. The contribution of this framework to students is also noticeable. Student engagement can be improved by online instructional multimedia material, and course online content can be easily tailored to the students' needs.

Since new technologies always afford new roles for teachers as learners and researchers, we also intend to pursue our current research program endeavors, aiming at developing e-learning strategies for the purpose of promoting reflection on teaching and collaborative learning using the Pliant framework. For example, how might teachers/teacher educators use Pliant tools for reflection and research into their classrooms? What are the most critical Pliant design patterns that would optimize their knowledge-building efforts? How will they use that information in their instructional decisions? These challenges are currently under investigation, and our ultimate goal is to produce a solution in the form of an analysis toolkit.

Finally, we would like to initiate a comprehensive evaluation (empirical study) of the use of Academia at both CUSB and Ryerson University. Our targeted audiences are students and staff. We are currently preparing an online questionnaire with the aim of gaining substantial and quantitative-based reactions to the use of Academia as a complement to the already sophisticated Blackboard platform for the purpose of e-learning. Our intention is to measure how far our framework can be useful in delivering and managing online courses.

CONCLUSION

We have described Pliant as a "standalone" and "Web-based" language that encapsulates both the "human" and "computer" levels of thinking and coding programs. This unique privilege makes it an exceptional language, compared to any other existing one. It also demonstrates a higher level of flexibility, reasonable adaptability, and integration, providing for higher software development capabilities and enhancements. These qualities can be exploited to improve the current state-of-the-art e-learning development tools, while meeting educational expectations, economical and time constraints, and human resources limitations. The main advantages of Pliant over other integrated software solutions are high transferability, flexibility, and maintainability. We have also presented Academia, a lightweight courseware application fully implemented in Pliant. When compared to mainstream courseware applications, Academia is surely over-simplified. Finally, a simple, yet concrete example of how Academia was used in a Translation program at CUSB was proposed, along with some insights on the e-learning methodology and pedagogy that were used.

REFERENCES

Blackboard. (2002). *Blackboard Course Management System 5.1.* Retrieved from http://www.blackboard.com/

Booch, G., Rumnaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language user guide* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.

Brusilovsky, P. (2001, October 23-27). WebEx: Learning from examples in a programming course. In W. Fowler & J. Hasebrook (Eds.), *Proceedings of WebNet'2001, the World Conference of the WWW and Internet* (pp. 124-129), Orlando, FL.

Brusilovsky, P., Stock, O., & Strapparava, C. (Eds.). (2000, August). Adaptive hypermedia and adaptive Web-based systems. *Proceedings of the*

AH 2000 International Conference, Trento, Italy. Berlin: Springer-Verlag (LNCS 1892).

De Mendez, P.O. (1988). Pliant: Expressive power plus efficiency. *Proceedings of the SALCOM-IT Workshop and Review Meeting,* Barcelona, Spain.

De Mendez, M., De Mendez, P.O., Santos, M.V., & Tonneau, H. (2000). *Pliant documentation.* Retrieved from http://old.fullpliant.org/

E-Workflow. (2003). *The workflow portal.* Retrieved from http://www.e-workflow.org

Hochheiser, H., & Shneiderman, B. (2001). Universal usability statements: Marking the trail for all users. *ACM Interactions, 8*(March-April), 16-18. Retrieved from http://www.acm.org/pubs/citations/journals/interactions/2001-8-2/p16-hochheiser/

Nielsen, J. (1993). *Usability engineering.* Boston: Academic Press.

Nyongwa, M., & Aubin, M.C. (2004). *Plan de développement stratégique: Traduction et langues.* Retrieved from http://www.ustboniface.mb.ca/

Savourel, Y. (2000). XML technologies and the localization process. *Multi-Lingual Computing & Technology, 11*(7).

Tonneau, H., De Mendez, P.O., & Santos, M.V. (1984). *Pliant homepage.* Retrieved from http://pliant.cx

Wang, M. (2000). *Evaluating the usability of Web-based learning tools.* Master's Thesis, Department of Computer Science, University of Victoria, Canada.

WebCT. (2002). *WebCT Course Management System 3.8.* Retrieved from http://www.webct.com

KEY TERMS

Academia: A courseware development Web portal, fully implemented in the Pliant language.

Courseware: Computer software and associated materials designed for educational or training purposes.

E-Learning: The use of network technology to design, deliver, select, administer, and adapt learning activities.

Fullpliant: Name given to the Pliant' operating system.

Pliant: The first efficient, truly extendable, customizable programming language. It is suited both for small scripts and for very large applications, and could be described as a combination of reflexive C, C++, typed Lisp, and clean syntax in a single language.

Software Design: Process of problem solving and planning for a software solution.

Usability: A measure, in our context, of how easy it is to use software to perform prescribed tasks.

ENDNOTES

1 This chapter is an extended version of a preliminary work entitled "Pliant: More Than a Programming Language, a Flexible E-Learning Tool," published in the *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2004* (pp. 505-510). Chesapeake, VA: AACE.

2 Collège Universitaire de Saint-Boniface, University of Manitoba, Winnipeg, Manitoba, Canada.

[3] Academia was developed at the Department of Computer Science at Ryerson University, Toronto, Canada.

[4] The Co-Op Education and Internship program at Ryerson University is managed by means of a Co-op Web portal available at *http://www.scs.ryerson.ca/~co-op*.

[5] This should not be confused with other project of the same name available online at *http://www.pliant.org/*.

[6] Meta-programming refers to the ability of Pliant to eliminate the barrier between low-level languages like C and high-level languages like Lisp or Python.

[7] Academia is a Pliant-based Web portal. Its architecture is described later in the section entitled "Academia's System-Level Capabilities."

[8] Please refer to the section entitled "Why Academia?" for an introduction to this tool.

[9] Actors are objects outside of the scope of the system, but that have significant interactions with it.

[10] All URLs mentioned in this chapter are fictitious.

This work was previously published in Handbook of Research on E-Learning Methodologies for Language Acquisition, edited by R. de Cássia Veiga Marriott & P. Lupion Torres, pp. 166-185, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Section IV
Utilization and Application

This section introduces and discusses the ways in which information technology has been used to shape the realm of software applications and proposes new ways in which IT-related innovations can be implemented within organizations and in society as a whole. These particular selections highlight, among other topics, intelligent software agents in e-commerce and utilizing open source software in organizations. Contributions included in this section provide excellent coverage of today's changing environment and insight into how evolutions in software applications impact the fabric of our present-day global village.

Chapter 4.1
Intelligent Software Agents with Applications in Focus

Mario Janković-Romano
University of Belgrade, Serbia

Milan Stanković
University of Belgrade, Serbia

Uroš Krčadinac
University of Belgrade, Serbia

INTRODUCTION

Most people are familiar with the concept of agents in real life. There are stock-market agents, sports agents, real-estate agents, etc. Agents are used to filter and present information to consumers. Likewise, during the last couple of decades, people have developed software agents, that have the similar role. They behave intelligently, run on computers, and are autonomous, but are not human beings.

Basically, an agent is a computer program that is capable of performing a flexible and independent action in typically dynamic and unpredictable domains (Luck, McBurney, Shehory, & Willmott, 2005). Agents are capable of performing actions and making decisions without the guidance of a human. Software agents emerged in the IT because of the ever-growing need for information processing, and the problems concerning dealing and working with large quantities of data.

Especially important is how agents act with other agents in the same environment, and the connections they form to find, refine and present the information in a best way. Agents certainly can do tasks better if they perform together, and that is why the multi-agent systems were developed.

The concept of an agent has become important in a diverse range of sub-disciplines of IT, including software engineering, networking, mobile systems, control systems, decision support, information recovery and management, e-commerce, and many others. Agents are now used in an increasingly wide number of applications — ranging from comparatively small systems such as web or e-mail filters to large, complex systems such as

air-traffic control, that have a large dependency on fast and precise decision making.

Undoubtedly, the main contribution to the field of intelligent software agents came from the field of artificial intelligence (AI). The main focus of AI is to build intelligent entities and if these entities sense and act in some environment, then they can be considered agents (Russell & Norvig, 1995). Also, object-oriented programming (Booch, 2004), concurrent object-based systems (Agha, Wegner, and Yonezawa, 1993), and human-computer interaction (Maes, 1994) are fields that constantly drive forward the development of agents.

BACKGROUND

Although the term 'agent' is widely used, by many people working in closely related areas, it defies attempts to produce a single universally accepted definition. One of the most broadly used definitions states that *"an agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives"* (Wooldridge and Jennings, 1995).

There are three main concepts in this definition: *situatedness*, *autonomy*, and *flexibility:*

- *Situatedness* means that an agent is situated in some environment and that it receives sensory input and performs actions which change that environment in some way.
- *Autonomy* is the ability of an agent to act without the direct intervention of humans. It has control over its own actions and over its internal state. Also, the autonomy implies the capability of learning from experience.
- *Flexibility* means that the agent is able to perceive its environment and respond to changes in a timely fashion; it should be able to exhibit opportunistic, goal-directed behaviour and take the initiative whenever appropriate. In addition, an agent should be able to interact with other agents and humans, thus to be 'social'.

For some researchers - particularly those interested in AI - the term `agent' has a stronger and more specific meaning than that sketched out above. These researchers generally mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualized or implemented using concepts that are more usually applied to humans. For example, it is quite common in AI to characterize an agent using mentalistic notions, such as knowledge, belief, intention, and obligation (Wooldridge & Jennings, 1995).

INTELLIGENT SOFTWARE AGENTS

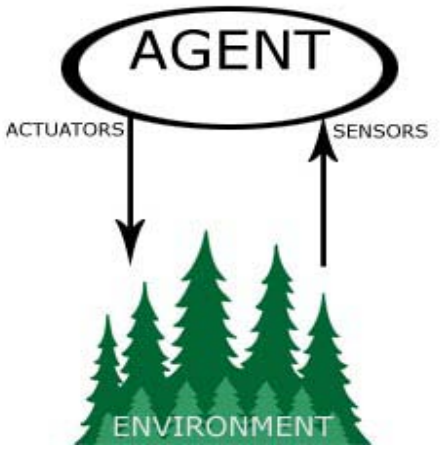
Agents and Environments

An agent collects its percepts through its sensors, and acts upon the environment through its actuators. Thus, the agent is proactive. Its actions in any moment depend on the whole sequence of these inputs up to that moment. A decision tree for every possible percept sequence of an agent would completely define the agent's behavior. This would define the function that maps any sequence of percepts to the concrete action – *the agent function*. The program that defines the agent function is called the *agent program*. So, the agent function is a formal description of the agent's behavior, and the agent program is a concrete implementation of that formalism. (Krcadinac, Stankovic, Kovanovic & Jovanovic, 2007)

To implement all this, we need to have a computing device with appropriate sensors and actuators on which the agent program will run. This is called *agent architecture*. So, an agent is essentially made of two components: the agent architecture and the agent program.

Also, as Russell and Norvig (1995) specify,

Figure 1. Agent and environment



one of the most sought after characteristics of an agent is its *rationality.* An agent is rational if it always does the action that will lead to the most successful outcome. The rationality of an agent depends on (a) the performance measure that defines what is a good action and what is a bad action, (b) the agent's knowledge about the environment, (c) the agent's available actions, and (d) the agent's percept history.

The Types of Agents

There are several basic types of agents with respect to their structure (Russell & Norvig, 1995):

1. The simplest kind of agents are the *simple reflex agents*. Such an agent only reacts to its current percept, completely ignoring its percept history. When a new percept is received, a rule that maps that percept to an action is activated. Such rules are known as *condition-action rules.*
2. *Model-based reflex agents* are more powerful agents, because they maintain some sort of internal state of the environment that depends on the percept history. For maintaining this sort of information, an agent must know how the environment evolves, and how its actions affect the environment.
3. *Goal-based agents* have some sort of goal information that describes desirable states of the world. Such an agent's decision making process is fundamentally different, because when a goal-based agent is considering performing an action it is asking itself "would this action make me happy?" along with the standard "what this action will have as a result?".
4. *Utility-based agents* use a utility function that maps each state to a number that represents the degree of happiness. They are able to perform rationally even in the situations when there are conflicting goals, as well as when there are several goals that can be achieved, but none with certainty.
5. *Learning agents* do not have a priori knowledge of the environment, but *learn* about it. This is beneficial because these agents can operate in unknown environments and to a certain degree facilitates the job of developers because they do not need to specify their whole knowledge base.

Multi-Agent Systems

Multi-Agent Systems (MAS) are systems composed of multiple autonomous components (agents). They historically belong to *Distributed Artificial Intelligence* (Bond & Gasser, 1998). MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of a single problem solver (Durfee and Lesser, 1989). In a MAS, each agent has incomplete information or capabilities for solving the problem and thus has a limited viewpoint. There is no global system control, the data is decentralized and the computation is asynchronous.

In addition to MAS, there is also the concept of a *multi-agent environment,* which can be seen as an environment that includes more than one agent.

Thus, it can be cooperative, or competitive, or a combined one, and creates a setting where agents need to interact (socialize) between each other, either to achieve their individual objectives, or to manage the dependencies that follow from being situated in a common environment. These interactions range from simple semantic interoperation (exchanging comprehensible communications), client-server interactions (the ability to request that a particular action is performed), to rich social interactions (the ability to cooperate, coordinate, and negotiate about a course of action).

Because of the issues due to heterogeneous nature of agents involved in communication (e.g., finding one another), there is also a need for *middle-agents*, which cover cooperation among agents and connect service providers with service requesters in the agent world. These agents are useful in various roles, such as *matchmakers* or *yellow page agents* that collect and process service offers ("advertisements"), *blackboard* agents that collect requests, and *brokers* that process both (Sycara, Decker, & Williamson, 1997). There are several alternatives to middle agents, such as Electronic Institutions – a framework for Agents' Negotiation which seeks to incorporate organizational concepts into multi-agent systems. (Rocha and Oliveira, 2001)

Communication among agents is achieved by exchanging messages represented by mutually understandable language (syntax) and containing mutually understandable semantics. In order to find a common ground for communication, an *agent communication language* (*ACL*) should be used to provide mechanisms for agents to negotiate, query, and inform each other. The most important such languages today are *KQML* (Knowledge Query and Manipulation Language) (ARPA Knowledge Sharing Initiative, 1993) and *FIPA ACL* (FIPA, 1997).

AGENT APPLICABILITY

There are great possibilities for applying multi-agent systems to solving different kinds of practical problems.

- *Auction negotiation model*, as a form of communication, enables a group of agents to find good solutions by achieving agreement and making mutual compromises in case of conflicting goals. Such an approach is applicable to trading systems, where agents act on behalf of buyers and sellers. Financial markets, as well as scheduling, travel arrangement, and fault diagnosing also represent applicable fields for agents.
- Another very important field is *information gathering*, where agents are used to search through diverse and vastly different information sources (e.g., World Wide Web) and acquire relevant information for their users. One of the most common domains is Web browsing and search, where agents are used to adapt the content (e.g., search results) to the users' preferences and offer relevant help in browsing.
- *Process control software systems* require various kinds of automatic (autonomous) control and reaction for its processes (e.g. production process). Reactive and responsive, agents perfectly fit the needs of such a task. Example domains in this field include: production process control, climate monitoring, spacecraft control, and monitoring nuclear power plants.
- *Artificial life* studies the evolution of agents, or populations of computer simulated life forms in artificial environments. The goal is to study phenomena found in real life evolution in a controlled manner, hopefully to eliminate some of the inherent limitations and cruelty of evolutionary studies using live animals.

- Finally, *intelligent tutoring systems* often include pedagogical agents, which represent software entities constructed to present the learning content in a user-friendly fashion and monitor the user's progress through the learning process. These agents are responsible for guiding the user and suggesting additional learning topics related to the user's needs (Devedzic, 2006).

Some of the more specific examples of intelligent agent applications include Talaria System, military training, and Mobility Agents. Talaria System (The Autonomous Lookup And Report Internet Agent System) is a multi-agent system, developed for academic purposes at the University of Belgrade, Serbia. It was built as a solution to the common problem of gathering information from diverse Web sites that do not provide RSS feeds for news tracking. The system was implemented using the JADE modeling framework in Java. (Stankovic, Krcadinac, Kovanovic & Jovanovic, 2007) Talaria System is using the advantages of human-agent communication model to improve usability of web sites and to relieve users from annoying and repetitive work. The system provides each user with a personal agent, which periodically monitors the Web sites that the user expressed interest in. The agent informs its user about relevant changes, filtered by assumed user preferences and default relevance factors. Human-agent communication is implemented via email, so that a user can converse with her/his agent in natural language, whereas the agent heuristically interprets concrete instructions from the mail text (e.g., "monitor this site" or "kill yourself").

Simulation and modelling are extensively used in a wide range of military applications, from development, testing and acquisition of new systems and technologies, to operation, analysis and provision of training, and mission rehearsal for combat situations. The Human Variability in Computer Generated Forces (HV-CGF) project, undertaken on behalf of the UK's Ministry of Defence, developed a framework for simulating behavioral changes of individuals and groups of military personnel when subjected to moderating influences such as caffeine and fatigue. The project was built with the JACK Intelligent Agents toolkit, a commercial Java-based environment for developing and running multiagent applications. Each team member is a rational agent able to execute actions such as doctrinal and non-doctrinal behaviour tactics, which are encoded as JACK agent graphical plans. (Belecheanu et al., 2005)

Mobility Agents is an agent-based architecture that helps a person with cognitive disabilities to travel using public transportation. Agents are used to represent transportation participants (buses and travelers) and to enable notification of bus approaching and arrival. Information is passed to the traveler using a multimedia interface, via a handheld device. Customizable user profiles determine the most appropriate modality of interaction (voice, text, and pictures) based on the user's abilities (Repenning & Sullivan, 2003). This imposes a personal agent to take care that abstract goals, as "go home", are translated into concrete directions. To achieve this, an agent needs to collect information about user-specific locations and must be able to suggest the right bus for the particular user's current location and destination.

FUTURE TRENDS

Future looks bright for this technology as development is taking place within a context of broader visions and trends in IT. The whole growing field of IT is about to drive forward the R&D of intelligent agents. We especially emphasize *the Semantic Web, ambient intelligence, service oriented computing, Peer-to-peer computing* and *Grid Computing.*

The Semantic Web is the vision of the future Web based on the idea that the data on the Web can be defined and linked in such a way that it can

be used by machines for the automatic processing and integration (Berners-Lee, Hendler, & Lassila, 2001). The key to achieving this is by augmenting Web pages with descriptions of their content in such a way that it is possible for machines to reason automatically about that content. The common opinion is that the Semantic Web itself will be a form of intelligent infrastructure for agents, allowing them to "understand" the meaning of the data on the Web (Luck et al., 2005).

The concept of *ambient intelligence* describes a shift away from PCs to a variety of devices which are embedded in our environment and which are accessed via intelligent interfaces. It requires agent-like technologies in order to achieve autonomy, distribution, adaptation, and responsiveness.

Service oriented computing is where MAS could become very useful. In particular, this might involve web services, where the Quality Of Service demands are important. Each web service could be modeled as an agent, with dependencies, and then simulated for observed failure rates.

Peer-to-peer (P2P) computing, presenting networked applications in which every node is in some sense equivalent to all others, tends to become more complex in the future. Auction mechanism design, agent negotiation techniques, increasingly advanced approaches to trust and reputation, and the application of social norms, rules and structures - presents some of the agent technologies that are about to become relevant in the context of P2P computing.

Grid Computing is the high-performance agent-based computing infrastructure for supporting large-scale distributed scientific endeavour. The Grid provides a means of developing eScience applications, yet it also provides a computing infrastructure for supporting more general applications that involve large-scale information handling, knowledge management and service provision. The key benefit of Grid computing is flexibility – the distributed system and network can be reconfigured on demand in different ways as business needs change.

Some considerable challenges have still remained in the agent-based world, such as the lack of sophisticated software tools, techniques and methodologies that would support the specification, development, integration and management of agent systems.

CONCLUSION

Today, research and development in the field of intelligent agents is rapidly expanding. At its core is the concept of autonomous agents interacting with one another for their individual and/or collective benefit. A number of significant advances have been made over the past two decades in design and implementation of individual autonomous agents, and in the way in which they interact with one another. These concepts and technologies are now finding their way into commercial products and real-world software solutions. Future IT visions share the common need for agent technologies and prove that agent technologies will continue to be of vital importance. It is foreseeable that agents will become the integral part of informational technologies and artificial intelligence in the near future, and that is why they should be kept an eye on.

REFERENCES

Agha, G., Wegner, P., & Yonezawa, A. (Eds.). (1993). *Research directions in concurrent object-oriented programming.* Cambridge, MA: The MIT Press.

ARPA Knowledge Sharing Initiative. (1993). *Specification of the KQML agent-communication language – plus example agent policies and architectures.* Retrieved January 30, 2007, from

http://www.csee.umbc.edu/kqml/papers/kqml-spec.pdf.

Barber, K. S., and Martin, C. E. (1999). *Agent Autonomy: Specification, Measurement, and Dynamic Adjustment*, Autonomy Control Software Workshop, Seattle, Washington.

Belecheanu, A. R., Luck, M., McBurney P., Miller T., Munroe, S., Payne T., & Pechoucek M. (2005). *Commercial Applications of Agents: Lessons, Experiences and Challenges.* (p. 2) Southampton, UK.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). The Semantic Web, *Scientific American*, pp. 35-43.

Bond, A. H., & Gasser, L. (Eds.). (1998). *Readings in distributed artificial intelligence.* San Mateo, CA: Morgan Kaufmann Publishers.

Booch, G. (2004). *Object-oriented analysis and design (2nd ed.).* MA: Addison-Wesley.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004, February)., *Web services architecture. W3C working group note 11.* Retrieved January 30, 2007, from http://www.w3.org/TR/ws-arch/.

Devedzic, V. (2006). *Semantic web and education.* Berlin, Heidelberg, New York: Springer.

Durfee, E. H., & Lesser, V. (1989). Negotiating task decomposition and allocation using partial global planning. In L. Gasser, & M. Huhns (Eds.), *Distributed artificial intelligence: Volume II* (pp. 229–244) London: Pitman Publishing and San Mateo, CA: Morgan Kaufmann.

FIPA (1997). *Part 2 of the FIPA 97 specifications: Agent communication language.* Retrieved January 30, 2007, from http://www.fipa.org/specs/fipa00003/OC00003A.html.

Krcadinac, U., Stankovic, M., Kovanovic, V., & Jovanovic, J. (2007). Intelligent Multi-Agent Systems in: Carteli, A., & Palma, M. (Eds.). *Encyclopedia of Information Communication Technology*, Idea Group International Publishing, (forthcoming)

Luck, M., McBurney, P., Shehory, O., & Willmott, S. (2005). *Agent technology: Computing as interaction.* Retrieved January 30, 2007, from http://www.agentlink.org/roadmap/al3rm.pdf.

Maes, P.(1994) Agents that reduce work and information overload. *Communications of the ACM*, 37(7), 31–40.

Repenning, A., & Sullivan, J. (2003). The Pragmatic Web: Agent-Based Multimodal Web Interaction with no Browser in Sight, In G.W.M. Rauterberg, M. Menozzi, & J. Wesson, (Eds.), *Proceedings of the Ninth International Conference on Human-Computer Interaction* (pp. 212-219). Amsterdam, The Netherlands: IOS Press.

Rocha, A. P. & Oliveira, E. (2001) *Electronic Institutions as a framework for Agents' Negotiation and mutual Commitment.* In P. Brazdil, A. Jorge (Eds.), Progress in Artificial Intelligence (Proceedings of 10th EPIA), LNAI 2258, pp. 232-245, Springer.

Russell, S. J., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. New Jersey: Prentice-Hall.

Stankovic, M., Krcadinac, U., Kovanovic, V., & Jovanovic, J. (2007). An Overview of Intelligent Software Agents in: Khosrow-Pour, M. (Ed.). *Encyclopedia of Information Science and Technology, 2nd Edition*, Idea Group International Publishing, (forthcoming)

Sycara, K., Decker, K., & Williamson, M. (1997). Middle-Agents for the Internet, In M. E. Pollack, (Ed.), *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 578-584). Morgan Kaufmann Publishers.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2), pp. 115–152.

KEY TERMS

Actuators: Software component and part of the agent used as a mean of performing actions in the agent environment.

Agent Autonomy: Agent's active use of its capabilities to pursue some goal, without intervention by any other agent in the decision-making process used to determine how that goal should be pursued (Barber & Martin, 1999).

Agent Percepts: Every information that an agent receives trough it's sensors, about the state of the environment or any part of the environment.

Intelligent Software Agent: An encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives (Wooldridge & Jennings, 1995).

Middle-Agents: Agents that facilitate cooperation among other agents and typically connect service providers with service requesters.

Multi-Agent System (MAS): A software system composed of several agents that interact in order to find solutions of complex problems.

Sensors: Software component and part of the agent used as a mean of acquiring information about current state of the agent environment (i.e., agent percepts).

This work was previously published in Encyclopedia of Artificial Intelligence, edited by J. Dopico; J. de la Calle; A. Sierra, pp. 950-955, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.2
Simulation Modelling within Collaborative Spatial Decision Support Systems Using "Cause–Effect" Models and Software Agents

Raja Sengupta
McGill University, Canada

ABSTRACT

Solutions to spatial environmental problems often require the integration of dynamic simulation models within GIS to create spatial decision support systems (SDSS) that can generate responses to theoretical "What if?" scenarios. Extending this paradigm to a collaborative spatial decision support system, however, faces significant challenges. This includes the inability of computationally intensive models to provide real-time results, and the inability of novice end users to effectively parameterize the models. Effective solutions to these problems proposed here include the use of "cause-effect" models to link inputs to outputs for a limited number of scenarios, as well as utilizing software agents that assist novice users in determining the correct input parameters for the models. Examples from the St-Esprit watershed SDSS serve to elucidate the proposed solutions.

INTRODUCTION

Sprague (1980) defined decision support systems (DSS) as computer software that are: (a) designed to solve the kinds of semi- and unstructured problems that upper level managers often face; (b) able to combine analytical models with traditional data storage and retrieval functions; (c) user-friendly and accessible by decision makers with minimal computer experience; and (d) flexible and adaptable to different decision-making approaches. Extending this definition, Armstrong, Densham, & Rushton (1986) used the term spatial decision support systems to refer to computer programs

that help decision-makers solve semistructured spatial problems through the integration of analytical models, spatial data, and traditional geoprocessing software (such as GIS). Therefore, an SDSS supports the spatial decision-making process by providing access to models that help users assess the impact of alternative solutions on stakeholders in collaborative environments (Armstrong, 1994).

Several spatial decision support systems (SDSS) implementations have been proposed or demonstrated in literature that incorporates analytical modeling with traditional GIS software packages for decision support. For example, the NELUP DSS was developed to study the impact of policy changes (at a global, European, national, regional, county, or local level) on the rural landscape, agriculture, and environment using economic, hydrologic, and habitat models (Watson & Wadsworth, 1996). Another SDSS, which relocated supply stores for school districts, was developed to find the optimal number of regions and service locations within each region to serve a dispersed geographical pattern of demand (Armstrong, Rushton, Honey, Dalziel, Lolonis, De, & Densham, 1991). The "WaterWare" DSS was developed as a comprehensive SDSS to support the development of an integrated river basin management plan to resolve conflicting uses such as recreation, agriculture, water supply, and the environment (Jamieson & Fedra, 1996). And finally, faunal habitat and landscape ecology models are being increasingly integrated with GIS to support decision making at the landscape level (McGarigal & Marks, 1993; Larson & Sengupta, 2004; Zhu, Healey, & Aspinall, 1998)

However, current SDSS often act as a basic interface to complicated models, leaving the end user to figure out model input parameters on their own. Therefore, the execution of such models in the SDSS in real time creates three hurdles for end users of the SDSS:

1. The models are computationally demanding, and can take up system resources that essentially render them impossible to execute in real time;
2. Most SDSS expect the end user to be well versed with the input data requirements and critical parameters of the model, which requires scientific knowledge of the process being simulated; and
3. Most models integrated within SDSS aren't well suited to assisting a collaborative decision-making process, with the above two points creating significant complexity while running models in group settings.

In effect, the lack of real-time model runs forces the collaborative decision-making process to rely on preset examples and/or static map outputs of sophisticated numerical simulations. This limitation not only severely restricts the range of options that can be evaluated by the group, it also forces individuals to rely on third-party technical support for generation of key scenarios.

This chapter describes the development of two strategies that can allow dynamic simulation scenarios to be incorporated within an SDSS to make it capable of supporting collaborative research, namely (i) "cause-effect" models linking outputs to spatial inputs, and (ii) software agents to assist users in selecting and understanding model parameters and input variables. Taken together, these two approaches can be combined with traditional multicriteria analysis (e.g., analytical hierarchy process or AHP) and GIS-based overlay procedures, thereby providing multiple decision-makers access to real-time generation of "What if?" scenarios incorporating dynamic spatial process simulations and allowing for collaborative problem-solving (Jankowski, 1995).

The effectiveness of this procedure in creating the prototype of a collaborative planning tool is tested using the example of the St-Esprit watershed in Quebec. In the first part, "cause-effect" models of stream discharge linked to varied land use scenarios are generated using the Soil and Water Assessment Tool (SWAT) program. In the

second part, conceptual models of software agents are discussed that could assist users in mastering the nuances of SWAT.

CAUSE-EFFECT MODELS

Modelling Runoff and Sediment Loading of the Saint-Esprit Watershed using SWAT

The watershed modeled in SWAT drains a portion of the St. Esprit County in Québec, 50 km north of Montreal (Figure 1). The St. Esprit River is a tributary of the L'Assomption River, which flows into the St. Lawrence. The St. Esprit watershed is 26.1 Km^2, of which 28% is forest, 26% corn, 12% hay, 8.5% wheat, and 7.3% residential (Figure 2). Generally, the relief of this region is rolling with a 64 m difference from the highest to lowest point, and slopes generally well below 5%. The soils are more gravelly and sandy in the headwaters, while clays predominate at the lower reaches of the basin. The soil texture classes are mainly loamy sands, sandy loams, and clays (see Figure 3).

SWAT, developed by the Blackland Research Center of the Texas A&M University System, the Grassland, Soil, and Water Research Laboratory of the USDA, and the US EPA Office of Science and Technology, is a public domain, physically based, distributed parameter hydrologic model specifically designed for agricultural watersheds (Srinivasan & Arnold, 1994).

SWAT relates a set of spatially distributed input parameters representing land use, soils, and climate to the output parameters of streamflow, sediment concentration, and nitrate and phosphate loading. For example, Figure 2 shows the land-use maps for the St. Esprit watershed, obtained in 1995 by a land survey carried out by the Brace Water Resources Center at the Macdonald Campus of McGill University (Enright, Papineau, & Madramootoo, 1998). This data, combined with nonspatial climate data obtained from Environment Canada (consisting of solar radiation, temperature, rainfall, evapotranspiration, and wind speeds from 1994 to 1998) and spatially explicit soil (Figure 3) and topographic maps obtained from the "Institut de Recherche et Development en Agroenvironnement" (IRDA), can be used to estimate stream discharge and sediment output from the mouth of the river (Figure 4). Obviously, there exists a complex mathematical relationship within the model where the output of the model

Figure 1. Location map of Saint-Esprit watershed

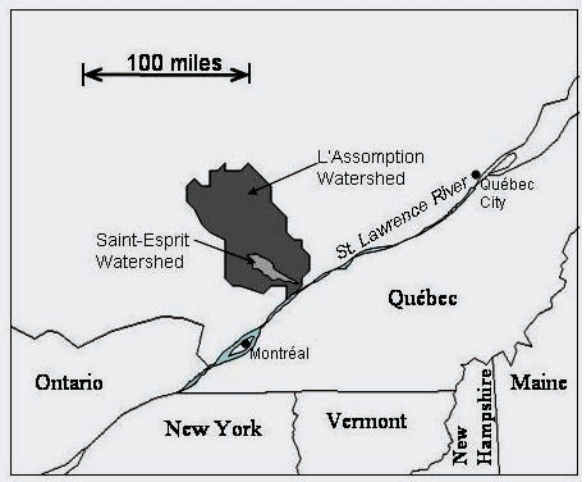


Figure 2. Landuse Map of Saint-Esprit Watershed

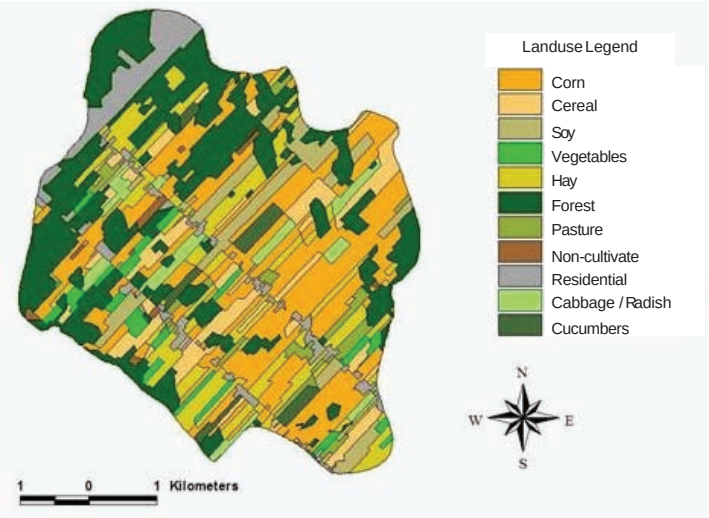


Figure 3. Soils Map (with streams overlay) of Saint-Esprit Watershed

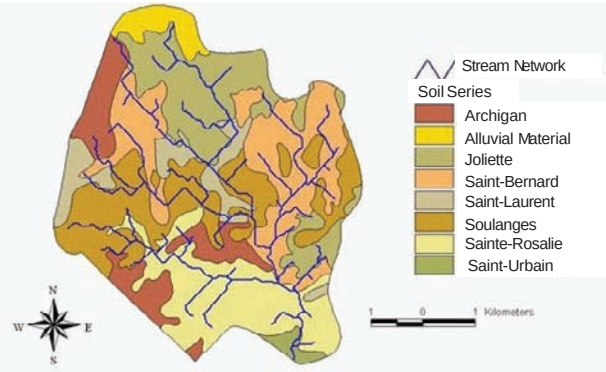
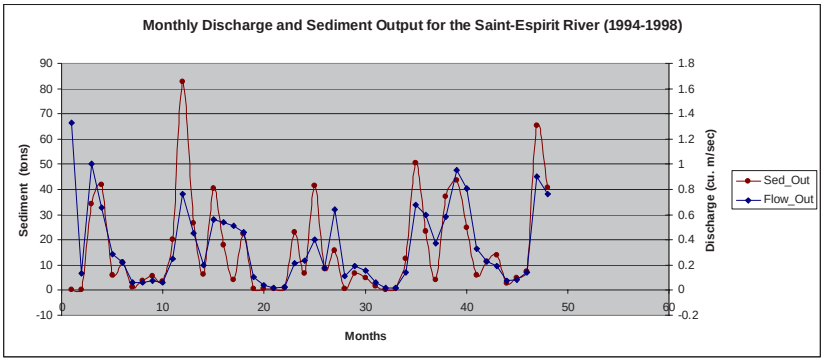


Figure 4. SWAT output showing simulated monthly discharge and sediment loading at the mouth of the St-Esprit River (1994-1998).



(e.g., discharge) is dependent on the input parameters (e.g., land use, soils, and climate), as shown in a simplified manner in Figure 5.

Modelling the Model: "Cause-Effect" Models that Link Nonspatial Inputs to Outputs

The interesting feature of the relationship shown in Figure 5 is that for an unchanged set of spatial data, the relationship between inputs and outputs is nonspatial in nature, and can be derived using either statistical techniques or with heuristic methods such as neural networks (Bekele & Nicklow, 2005). In other words, if land use, topography, or soils are not altered, then the relationship of the independent nonspatial variable, that is, climate, to the dependent nonspatial output, for example, discharge, varies with time but not across space. This important characteristic of SWAT and other similar spatially distributed hydrological models can therefore be exploited to build simplified models of relationships between the two nonspatial components, which is termed a "cause-effect" model here.

As demonstration, two cause-effect models were built for the Saint-Esprit SWAT simulation using a neural network software and linear regression respectively. In both cases, an identical dataset consisting of select climate variables (i.e., rainfall, snowmelt, and evapotranspiration) and discharge for 241 monthly simulations (1994-1998) were used to derive the relationship between these variables. For the first model, the neural network software Neurosolutions (http://www.neurosolutions.com) consisting of two layers of neurons connected in a feed-forward fashion was used. Eleven sample points were set aside as test cases, and the remaining 230 points used to train the neural network. Within the software, a genetic algorithm is used to train the neurons multiple times, and the default value of 1,000 iterations was selected to converge the neurons on a solution. The neural network was, however, able to converge within the first hundred iterations, indicating that a robust solution had been found. This solution was then used to predict discharge values. Figure 6 shows the comparison of the actual values for discharge for the eleven test points compared with the values predicted by the Neural Network.

The second cause-effect model was built statistically using linear regression. Its results were even more striking in that a single climate variable, snowmelt, explained over 78% of the variability in discharge. The strong correlation

Figure 5. Relationship between Input and Output variables

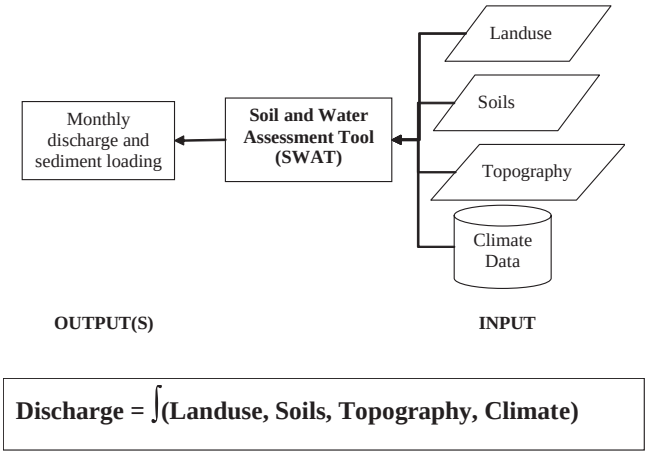
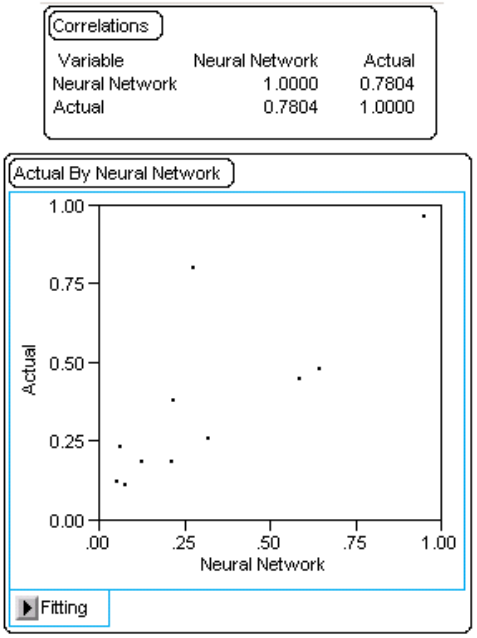


Figure 6. Comparison of discharge values predicted by the Neural Network with actual values; statistical results and scatter plot indicate a correlation coefficient of 0.78

Predicted Values (cu. m/sec)		Actual Values (cu. m/sec)
0.2158	0	.3854
0.5858	0	.4558
0.9506	0	.97
0.0506	0	.129
0.2118	0	.1915
0.3167	0	.2661
0.6421	0	.4862
0.2723	0	.8089
0.0772	0	.1198
0.0623	0	.2373
0.1251	0	.1919



amongst these two variables can be observed in a time-series plot and explained with the help of simple linear regression, as shown in Figure 7.

The cause-effect models discussed above indicate that if the spatially-explicit input datasets are not altered between model runs, it may be possible to devise simplified relationships that mimic the model's output without having to re-run the model itself. This would allow real-time simulations in a collaborative –decision-making environment, whereby users could get to work with instantaneous model outputs. In the case of SWAT shown previously, it was possible to relate only the climate variables to discharge (given that land use, topography, and soils remained constant). But what about land-use change scenarios, which entails changes to a spatially explicit input? For example, decision makers may wish to evaluate the effectiveness of placing riparian buffer strips adjacent to major streams, an accepted management practice that reduces nonpoint source pollution (Cey, Rudolph, Aravena, & Parkin, 1999; Peterjohn & Correll, 1984). In that case, it is proposed that a "cause-effect" model be developed for the finite set of land-use change scenarios that are likely to be evaluated by the decision makers. Figure 8 provides the example of four scenarios where decision makers must evaluate the establishment and effectiveness of riparian buffer strips of varying widths. In such a case, four 'cause-effect" models, one for each scenario, would need to be developed to assist in collaborative decision making.

SOFTWARE AGENTS

What are Intelligent Software Agents?

The term "intelligent agents" originated as an extension of artificial intelligence (AI) research

Figure 7. Statistical relationship between snowmelt and discharge (r2 = 0.78, p < 0.0001)

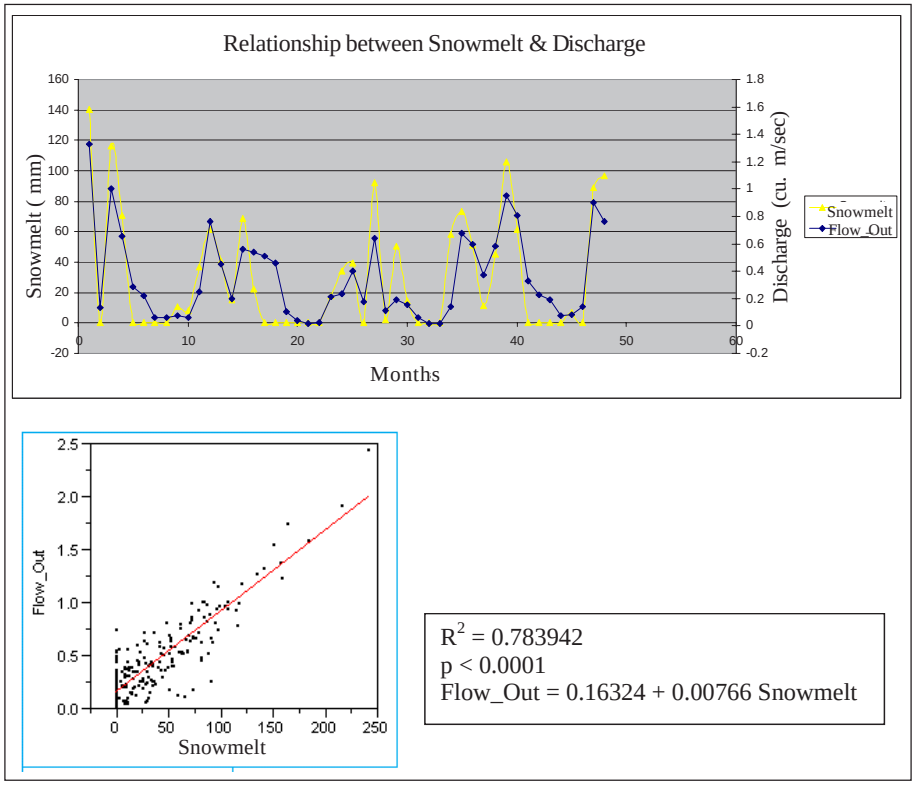
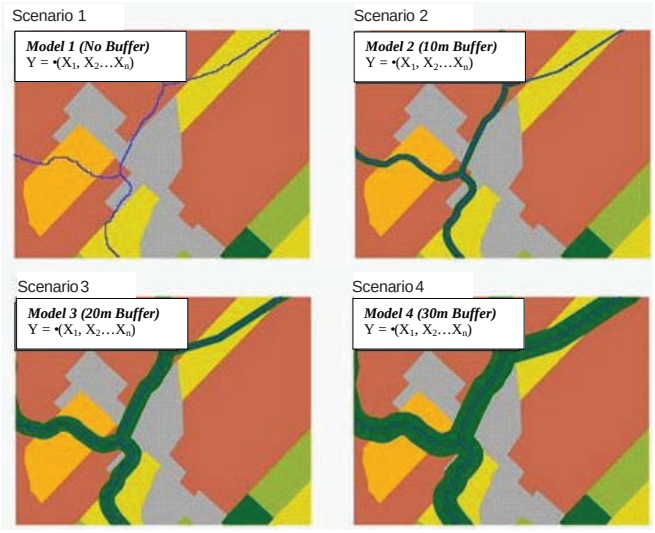


Figure 8. Varying widths of riparian band in the four scenarios, each with its own 'cause-effect' model



in late 1980s and early 1990s (Woolridge & Jennings, 1995). It refers to relatively autonomous software that manages information searching/retrieval and simulation in complex and changing operating environments such as the Internet. There is general consensus that to be considered an intelligent agent, the software must possess the following four properties: (a) autonomous behavior, (b) ability to sense its environment and other agents, (c) ability to act upon its environment alone or in collaboration with others (sometimes via an agent communication language), and (d) possession of rational behavior (Woolridge, 1999; Woolridge & Jennings, 1995). In some cases, intelligent agents should not only be able to respond to, but also learn from their environment (Maes, 1994). Humanistic characteristics such as beliefs, desires, intentions, (Shoham, 1993) and emotions and trust (Maes, 1994) also could form a part of agent behaviour. Since their inception, agents have become a popular technology for a variety of computer applications, ranging from managing human-computer interactions to simulating social interactions.

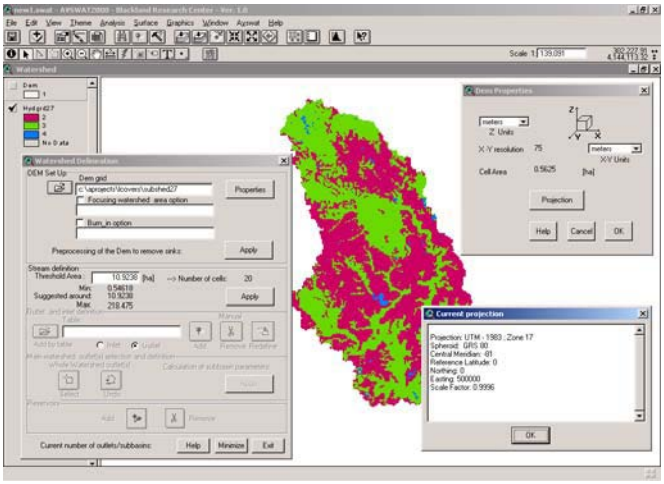
Software agents, a subclass of intelligent agents, are designed to act autonomously to manage complex information. Generally in GIScience, software agents can be oriented by four tasks (Nwana, 1996). First, *interface agents* assist users in interacting with a specific software or hardware environment by automating routine or difficult computing tasks. For example, Campos, Naumov, & Shapiro (1996) describe a user interface agent that provides assistance to users of the GIS software ARC/INFO. Interfacing is the primary task of software agents. Second, *collaborative agents* enable interagent, interhuman, or agent-human collaboration for tasks that require cooperation and consensus building. A subset of interface agents learns from observing repeated user-software interactions, thereby acting on behalf of users. Sengupta and Bennett (2003) detail an agent-based environment that uses multiple interacting agents to retrieve and manipulate data and spatial models to support decision-making environments. Third, *mobile agents* specialize in navigating networks by moving between computers, sensing security, and Internet protocols. Tsou and Buttenfield (2002) describe a mobile, distributed, geospatial analysis environment using mobile agents. Last, *information agents* specialize in data mining and information retrieval. An example of an information agent in GIScience/geography is one that surfs the Web for digital geospatial data, given a specific map extent and cartographic scale (Luo, Wang, & Xu, 2003).

Here I propose that *interface agents* be used to help users of collaborative SDSS parameterize the SWAT model, freeing them to focus on metatasks such as evaluating alternative land-use scenarios. The complexity of the input variables and the possible assistance afforded by agents is described in the following sections.

A Look at the Complexity of SWAT's Input Parameters and its Implications for Collaborative Decision-Making

Although there is an Arcview extension called AVSWAT available for SWAT (De Luzio, Srinivasan, & Arnold, 2004), it still requires the end user to have adequate knowledge of GIS data formats, structure of attribute tables, and projections. For example, the screen shot in Figure 9 shows how novice users may have to navigate pop-up windows with terminology like "DEM Set Up" and statements like "Project UTM – 1983; Zone 17." More complications arise when setting up soil parameters, where the users must deal with STATSGO/SURGO soil scheme classifications in order to complete one simulation cycle. This often leads to a bottleneck where most end users of SDSS rely on a GIS technician to operate the system (Armstrong & Densham, 1995). As a result, an interactive process dialog between the system and the decision makers is no longer possible. This severely distracts from the collaborative

Figure 9. Setting up input parameters for SWAT; Step 1 involves delineating sub-watersheds from a DEM



decision-making environment as members of the group wrestle with the nuances of the software. In addition, technologically adept members of the group begin to gain an unfair advantage in the collaborative decision-making process.

Agents to the Rescue

The *interface agents* proposed here assist users in two instances: first to parameterize models, and second to evaluate plausible scenario alternatives. The key to agent-based modeling is flexibility. Therefore, the *interface agents* themselves are simply front ends to a larger knowledge base that can take the form of other agents within Blackboard Architectures (Sengupta & Bennett, 2003) or changeable elements of knowledge stored in a database. The key function of the *interface agents* is to search the knowledge base to find possible sequence of transformations/solutions that completes the task at hand.

In the case of parameterizing SWAT, end users should not need to deal with complexities and specificities of the model, such as the land use code used by SWAT for representing vegetation. This information can be embedded in a database. An *interface agent*, as shown in Figure 10, is able to retrieve this and other details from database searches and place it in the model input files for the user, when necessary. Further, the agent is also able to assist users enter missing pieces of information, such as composition of the buffer strips, by converting user input into the numerical equivalents (e.g., land-use codes) desired by the model.

A second *interface agent* can be used to define spatially explicit scenarios. For example, in Figure 11, the agent is used to define regions within the map where land-use change is desired, which is then used as input to SWAT. In the figure, the user is prompted to demarcate areas on the map that need to be converted to buffer areas. Following the demarcation, the agent automatically creates buffer strips within the land use map (note that a second pop-up window not shown in the figure asks for the width of the buffer strips, and places the strips adjacent to land use coded as "water"). Both these agents hide the actual complexity of setting up the SWAT input files from the user.

Figure 10. Interface Agent assistance in defining parameters for the model SWAT.

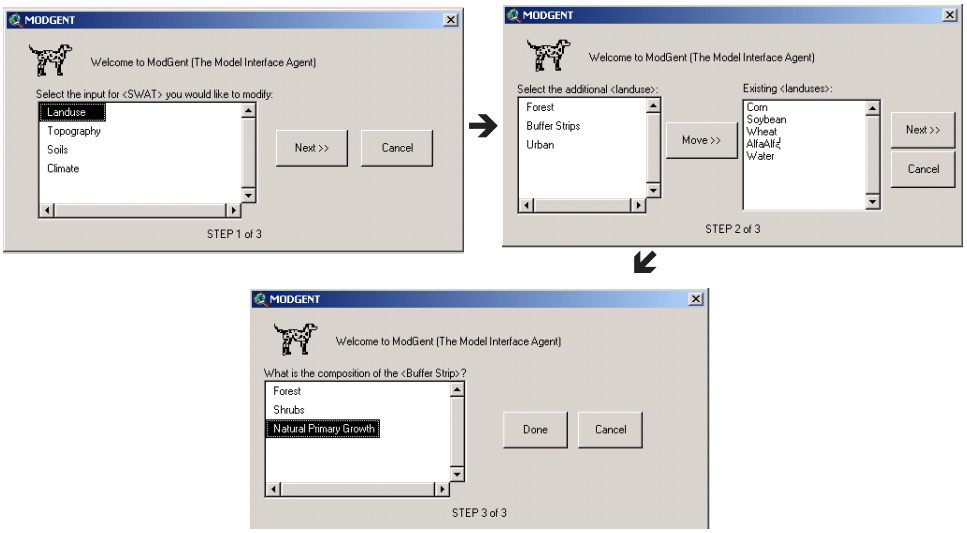
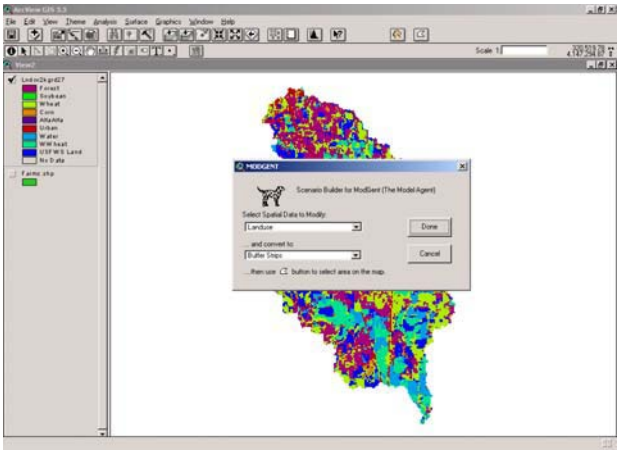


Figure 11. Interface Agents assist in development of land use scenarios



CONCLUSION

Using either the "cause-effect" models or intelligent agents or a combination of the two, it is possible to reduce both the time required to run a model multiple times, as well as the complexity of setting up its input parameters. As shown with examples from the St-Esprit SDSS, these methods can be very useful in real-world scenarios where decision makers, who are often nonGIS experts, may have to contend with deriving discharge information from simulation models like SWAT. Simplifying these tasks should enable better use of the SDSS in collaborative decision-making environments and restore the tool to its rightful position: in the hands of decision makers who have no GIS experience, but feel comfortable using it to make informed decisions.

REFERENCES

Armstrong, M. P. (1994). Requirements for the development of GIS-based group decision support systems. *Journal of the American Society for Information Science, 45*(9), 669-677.

Armstrong, M. P., & Densham, P. J. (1995). Collaborative spatial decision making: A look at the dark side. In *Proceedings of GIS/LIS: Vol. 1,* (pp. 11-19). Bethesda, MD: American Congress on Surveying and Mapping.

Armstrong, M. P., Densham, P. J., & Rushton, G. (1986). Architecture for a microcomputer-based spatial decision support system. In *Proceedings of the Second International Symposium on Spatial Data Handling* (pp. 120-131). Williamsville, NY: IGU Commission on Geographical Data Sensing and Processing.

Armstrong, M. P., Rushton, G., Honey, R., Dalziel, B. T., Lolonis, P., De, S., & Densham, P. J. (1991). Decision support for regionalization: A spatial decision support system for regionalizing service delivery systems. *Computers, Environment and Urban Systems, 15,* 37-53.

Bekele, E. G., & Nicklow, J. W. (2005, May 15-20). Hybrid evolutionary search methods for training an artificial neural network. *Proceedings of the Conference of the Environmental and Water Resources Institute,* Anchorage, AK. Reston, VA: ASCE.

Campos, D., Naumov, A., & Shapiro, S. (1996). Building an interface agent for ARC/INFO. *Proceedings of the ESRI International User Conference,* Palm Springs, CA. Redlands, CA: ESRI.

Cey, E., Rudolph, D., Aravena, R., & Parkin, G. (1999). Role of the riparian zone in controlling the distribution and fate of agricultural nitrogen near a small stream in southern Ontario. *Journal of Contaminant Hydrology, 37,* 45-67.

Di Luzio, M., Srinivasan R., & Arnold J.G. (2004). A GIS-coupled hydrological model system for the watershed assessment of agricultural nonpoint and point sources of pollution. *Transactions in GIS, 8*(1), 113-136.

Enright, P., Papineau, F., & Madramootoo, C. (1998). Gestion de l'Eau dans le Bassin Versant de la Partie Superieure du Ruisseau St-Esprit (Project 61-13008, Rapport Final). Montreal: McGill.

Jamieson, D. G., & Fedra, K. (1996). The "WaterWare" decision-support system for river basin planning: 1. Conceptual design. *Journal of Hydrology, 177,* 163-175.

Jankowski, P. (1995). Integrating GIS and mMultiple criteria decision making methods. *International Journal of Geographical Information Systems, 9*(3), 252-273.

Larson, B. D., & Sengupta, R., (2004). A spatial decision support system to identify species-specific critical habitats based on size and accessibility using US GAP data. *Environmental Modelling and Software, 19*(1), 7-18.

Luo, Y., Wang, X., & Xu, Z. (2003). Extension of spatial metadata and agent-based spatial data navigation mechanisms. *Proceedings of the Eleventh ACM international symposium on Advances in Geographic Information Systems* (pp. 102-109). New York: ACM Press.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM, 37*(7), 30-40.

McGarigal, K., & Marks, B. J. (1993). *FRAGSTATS-Spatial Pattern Analysis Program for Quantifying Landscape Structure*. Corvallis: Oregon State University, Forest Science Department.

Nwana, H. (1996). Software agents: An overview. *Knowledge Engineering Review, 11*(3), 1-40.

Peterjohn, W., & Correll, D. (1984). Nutrient dynamics in an agricultural watershed: Observations on the role of a riparian forest. *Ecology, 65*, 1466-1475.

Sengupta, R., & Bennett, D. A. (2003). Agent-based modeling environment for spatial decision support. *International Journal of Geographical Information Science, 17*(2), 157-80.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence, 60*(1), 51-92.

Srinivasan, R., & Arnold, J.G. (1994). Integration of a basin-scale water quality model with GIS. *Water Resources Bulletin, 30*(3), 453-462.

Sprague, R. H. (1980). A framework for the development of decision support systems. *MIS* Quarterly, *4*, 1-25.

Tsou, M. H., & Buttenfield, B. P. (2002). A dynamic architecture for distributed geographic information services. *Transactions in GIS, 6*(4), 355-81.

Watson, P. M., and Wadsworth, R. A. (1996). A computerized decision support system for rural policy formulation. *International Journal of Geographical Information Systems, 10*(4), 425-440.

Wooldridge, M. (1999). Intelligent agents. In G. Weiss (Ed.), *Multiagent systems: A modern approach to distributed artificial intelligence* (pp. 27-77). Cambridge: The MIT Press.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: theory and practice. *Knowledge Engineering Review, 10*(2), 115-52.

Zhu, X., Healey, R. G., & Aspinall, R. J. (1998). A knowledge-based systems approach to design of spatial decision support systems for environmental management. *Environmental Management, 22*(1), 3-48.

This work was previously published in Collaborative Geographic Information Systems, edited by S. Balram; S. Dragicevic, pp. 134-149, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 4.3
Intelligent Software Agents Analysis in E-Commerce I

Xin Luo
The University of New Mexico, USA

Somasheker Akkaladevi
Virginia State University, USA

INTRODUCTION

Equipped with sophisticated information technology infrastructures, the information world is becoming more expansive and widely interconnected. Internet usage is expanding throughout the web-linked globe, which stimulates people's need for desired information in a timely and convenient manner. Electronic commerce activities, powered by Internet growth, are increasing continuously. It is estimated that online retail will reach nearly $230 billion and account for 10% of total U.S. retail sales by 2008 (Johnson et al. 2003). In addition, e-commerce entailing business-to-business (B2B), business-to-customer (B2C) and customer-to-customer (C2C) transactions is spawning new markets such as mobile commerce.

By increasing the degree and sophistication of the automation, commerce becomes much more dynamic, personalized, and context sensitive for both buyers and sellers. Software agents were first used several years ago to filter information, match people with similar interests, and automate repetitive behavior (Maes et al. 1999). In recent years, agents have been applied to the arena of e-commerce, triggering a revolutionary change in the way we conduct online transactions in B2B, B2C, and C2C. Researchers argue that the potential of the Internet for transforming commerce is largely unrealized (Begin et al. 2002; Maes et al. 1999). Further, He and Jennings noted that a new model of software agent is needed to achieve the degree of automation and move to second generation e-commerce1 applications (He et al. 2003). This is due to the predicament that electronic purchases are still largely unautomated. Maes et al. (1999) also addressed that, even though information is more easily accessible and orders and payments are dealt with electronically, humans are still in the loop in all stages of the buying process, which inevitably increase the transaction costs. Undoubtedly, a human buyer is still responsible

for collecting and interpreting information on merchants and products, making decisions about merchants and products, and ultimately entering purchase and payment information. Additionally, Jennings et al. (1998) confirmed that commerce is almost entirely driven by human interactions and further argued that there is no reason why some commerce cannot be automated.

This unautomated loop requires a lot of time and energy and results in inefficiency and high cost for both buyers and sellers. To automate time-consuming tasks, intelligent software agent (ISA) technology can play an important role in online transaction and negotiation due to its capability of delivering unprecedented levels of autonomy, customization, and general sophistication in the way e-commerce is conducted (Sierra et al. 2003). Systems containing ISAs have been developed to automate the complex process of negotiating a deal between a buyer and a seller. An increasing number of e-commerce agent systems are being developed to support online transactions that have a number of variables to consider and to aim for a win-win result for sellers and buyers.

In today's e-commerce arena, systems equipped with ISAs may allow buyers and sellers to find the best deal taking into account the relative importance of each factor. Advanced systems of e-commerce that embody ISA technologies are able to perform a number of queries and to process phenomenal volumes of information. ISAs reduce transaction costs by collecting information about services and commodities from a lot of firms and presenting only those results with high relevance to the user. ISA technologies help businesses automate information transaction activity, largely eliminate human intervention in negotiation, lower transaction and information search cost, and further cultivate competitive advantage for companies. Therefore, ISAs can free people to concentrate on the issues requiring true human intelligence and intervention. Implementing the personalized, social, continuously running, and semi-autonomous ISA technologies in business information systems, the online business can become more user-friendly, semi-intelligent, and human-like (Pivk 2003).

LITERATURE REVIEW

A number of scholars have defined the term *intelligent software agent*. Bradshaw (1997) proposed that one person's intelligent agent is another person's smart object. Jennings and Wooldridge (1995) defined agents as a computer system situated in some environment that is capable of autonomous action in this environment to meets its design objective. Shoham (1997) further described an ISA as a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes. In general, an ISA is *a software agent that uses Artificial Intelligence (AI) in the pursuit of the goals of its clients* (Croft 2002). It can perform tasks independently on behalf of a user in a network and help users with information overload. It is different from current programs in terms of being proactive, adaptive, and personalized (Guttman et al. 1998b). Also, it can actively initiate actions for its users according to the configurations set by the users; it can read and understand user's preferences and habits to better cater to user's needs; it can provide the users with relevant information according to the pattern it adapts from the users.

ISA is a cutting-edge technology in computational sciences and holds considerable potential to develop new avenues in information and communication technology (Shih et al. 2003). It is used to perform multi-task operations in decentralized information systems, such as the Internet, to conduct complicated and wide-scale search and retrieval activities, and assist in shopping decision-making and product information search (Cowan et al. 2002). ISA's ability of performing continuously and autonomously stems from human desire in that an agent is capable of operat-

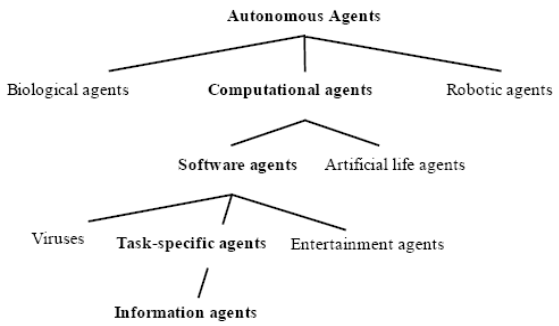
ing certain activities in a flexile and intelligent manner responsive to changes in the environment without constant human supervision. Over a long period of time, an agent is capable of adapting from its previous experience and would be able to inhabit an environment with other agents to communicate and cooperate with them to achieve tasks for human.

Intelligent Agent Taxonomy and Typology

Franklin and Grasser (1996) proposed a general taxonomy of agent (see Figure 1).

This taxonomy is based on the fact that ISA technologies are implemented in a variety of areas, including biotechnology, economic simulation and data-mining, as well as in hostile applications (malicious codes), machine learning and cryptography algorithms. In addition, Nwana (1996b) proposed the agent typology (see Figure 2) in which four types of agents can be categorized: collaborative agents, collaborative learning agents, interface agents and smart agents. These four agents have different congruence amid learning, autonomy, and cooperation and therefore tend to address different sides of this topology in terms of the functionality.

Figure 1. Franklin and Grasser's agent taxonomy (Source: Franklin & Grasser. 1996)



According to Nwana (1996b), collaborative agents emphasize more autonomy and cooperation than learning. They collaborate with other agents in multi-agent environments and may have to negotiate with other agents in order to reach mutually acceptable agreements for users. Unlike collaborative agents, interface agents emphasize more autonomy and learning. They support and provide proactive assistance. They can observe user's actions in the interface and suggest better ways for completing a task for the user. Also, interface agents' cooperation with other agents is typically limited to asking for advice (Ndumu et al. 1997). The benefits of interface agents include reducing user's efforts in repetitive work and adapting to their user's preferences and habits. Smart agents are agents that are intelligent, adaptive, and computational (Carley 1998). They are advanced intelligent agents summing up the best capabilities and properties of all presented categories.

This proposed typology highlights the key contexts in which the agent is used in AI literature. Yet Nwana (1996b) argued that agents ideally should do all three equally well, but this is the aspiration rather than the reality. Furthermore, according to Nwana (1996b) and Jennings and Wooldridge (1998), five more agent types could be derived based on the typology, from a panoramic perspective (see Figure 3).

In this proposed typology, mobile agents are autonomous and cooperative software processes capable of roaming wide area networks, interacting with foreign hosts, performing tasks on behalf of their owners (Houmb 2002). Information agents can help us manage the explosive growth of information we are experiencing. They perform the role of managing, manipulating, or collating information from many distributes sources (Nwana 1996b). Reactive agents choose actions by using the current world state as an index into a table of actions, where the indexing function's purpose is to map known situations to appropriate actions. These types of agents are sufficient for

Figure 2. A Part View of Agent Typology Source: Nwana (1996b)

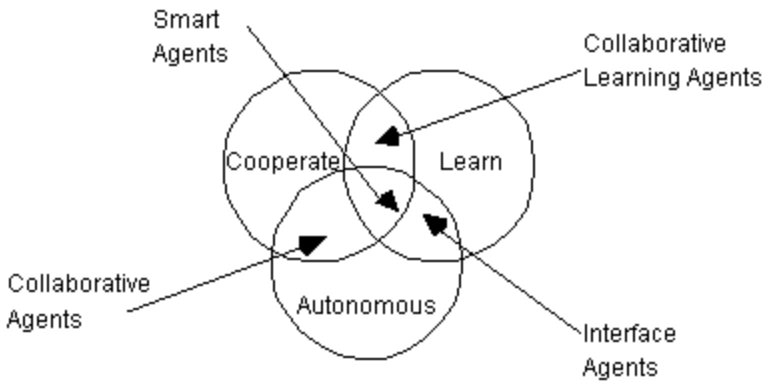
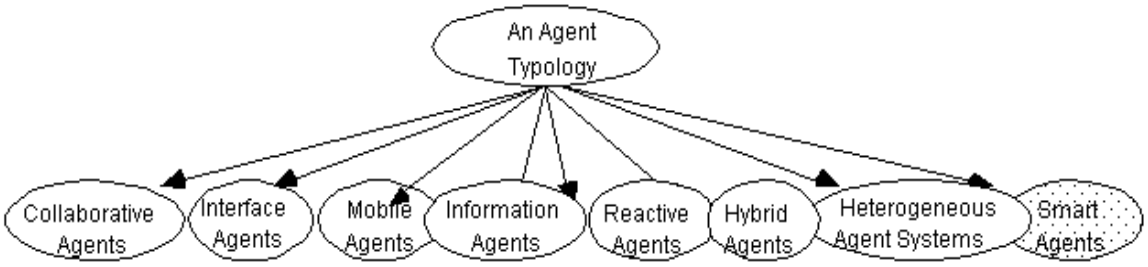


Figure 3. A panoramic overview of the different agent types (Source: Jennings & Wooldridge, 1998)



limited environments where every possible situation can be mapped to an action or set of actions (Chelberg 2003). Hybrid agents adopt strength of both the reactive and deliberative paradigms. They aim to have the quick response time of reactive agents for well known situations, yet also have the ability to generate new plans for unforeseen situations (Chelberg 2003). Heterogeneous agents systems refer to an integrated set-up of at least two or more agents, which belong to two or more different agent classes (Nwana 1996b).

CONCLUSION AND FUTURE WORK

This paper explores how ISAs can automate and add value to e-commerce transactions and negotiations. By leveraging ISA-based e-commerce systems, companies can more efficiently make decisions because they have more accurate information and identify consumers' tastes and habits. Opportunities and limitations for ISA development are also discussed. Future technologies of ISAs will be able to evaluate basic characteristics of online transactions in terms of price and product description as well as other properties, such as warranty, method of payment, and after-sales service. Also, they would better manage ambiguous content, personalized preferences, complex goals, changing environments, and disconnected parties (Guttman et al. 1998a). Additionally, for the future trend of ISA technology deployment, Nwana (1996a) describes that *"Agents are here to stay, not least because of their diversity, their wide range of applicability and the broad spectrum of companies investing in them. As we move further and further into the information age, any information-based organization which does not*

invest in agent technology may be committing commercial hara-kiri."

REFERENCES

Begin, L., and Boisvert, H. "Enhancing the value proposition Via the internet," *International Conference on Electronic Commerce Research (ICECR-5),* 2002.

Bradshaw, J.M. "Software Agents," *online: http://agents.umbc.edu/introduction/01-Bradshaw.pdf*) 1997.

Carley, K.M. "Smart Agents and Organizations of the Future," *online: http://www.hss.cmu.edu/departments/sds/faculty/carley/publications/ORGTHEO36.pdf*) 1998.

Chelberg, D. "Reactive Agents," *online: http://zen.ece.ohiou.edu/~robocup/papers/HTML/AAAI/node3.html*), 03-05 2003.

Cowan, R., and Harison, E. "Intellectual Property Rights in Intelligent-Agent Technologies: Facilitators, Impediments and Conflicts," *online: http://www.itas.fzk.de/e-society/preprints/ecommerce/CowanHarison.pdf*) 2002.

Croft, D.W. "Intelligent Software Agents: Definitions and Applications," *online: http://www.alumni.caltech.edu/~croft/research/agent/definition/*) 2002.

Franklin, S., and Graesser, A. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag,* 1996.

Guttman, R., Moukas, A., and Maes, P. "Agent-mediated Electronic Commerce: A Survey," *Knowledge Engineering Review* (13:3), June 1998a.

Guttman, R., Moukas, A., and Maes, P. "Agents as Mediators in Electronic Commerce," *International Journal of Electronic Markets* (8:1), February 1998b, pp 22-27.

He, M., Jennings, N.R., and Leung, H.-F. "On Agent-Mediated Electronic Commerce," *IEEE Transactions on Knowledge and Data Engineering* (15:4), July/August 2003.

Houmb, S.H. "Software Agent: An Overview," *online: http://www.idi.ntnu.no/emner/dif8914/ppt-2002/sw-agent_dif8914_2002.ppt*) 2002.

Jennings, N.R., and Wooldridge, M. "Applications of Intelligent Agents," *in Agent Technology: Foundations, Applications, and Markets,*1998, pp 3-28.

Johnson, C., Delhagen, K., and Yuen, E.H. "US eCommerce Overview: 2003 To 2008," *Online: http://www.forrester.com/ER/Research/Brief/Excerpt/0,1317,16875,00.html*), July 25 2003.

Maes, P., Guttnab, R.H., and Moukas, A.G. "Agents That Buy and Sell. (software agents for electronic commerce)(Technology Information)," *Communications of the ACM* (42:3) 1999, p 81.

Ndumu, D., and Nwana, H. "Research and Development Challenges for Agent-Based Systems," *IEE Proceedings on Software Engineering* (144:01), January 1997.

Nwana, H.S. "Software Agents: An Overview," *online: http://agents.umbc.edu/introduction/ao/*) 1996b.

Pivk, A. "Intelligent Agents in E-Commerce," *online: http://ai.ijs.si/Sandi/IntelligentAgent-Repository.html*) 2003.

Shih, T.K., Chiu, C.-F., and Hsu, H.-h. "An Agent-Based Multi-Issue Negotiation System in E-commerce," *Journal of Electronic Commerce in Organizations* (1:1), Jan-March 2003, pp 1-16.

Sierra, C., Wooldridge, M., Sadeh, N., Conte, R., Klusch, M., and Treur, J. "Agent Research and Development in Euroope," *online: http://www.unicom.co.uk/3in/ISSUE4/4.Asp*) 2003.

KEY TERMS

Agent: A computer system situated in some environment that is capable of autonomous action in this environment to meets its design objective.

Business-to-Business E-Commerce: Electronic transaction of goods or services between businesses as opposed to that between businesses and other groups.

Business-to-Customer E-Commerce: Electronic or online activities of commercial organizations serving the end consumer with products and/or services. It is usually applied exclusively to e-commerce.

Customer-to-Customer E-Commerce: Online transactions involving the electronically-facilitated transactions between consumers through some third party.

Electronic Commerce (E-Commerce): Consists of the buying and selling of products or services over electronic systems such as the Internet and other computer networks. A wide variety of commerce is conducted in this way, including electronic funds transfer, supply chain management, e-marketing, online transaction processing, and automated data collection systems.

Intelligent Software Agent: A software agent that uses Artificial Intelligence (AI) in the pursuit of the goals of its clients.

Ubiquitous Commerce (U-Commerce): The ultimate form of e-commerce and m-commerce in an 'anytime, anywhere' fashion. It involves the use of ubiquitous networks to support personalized and uninterrupted communications and transactions at a level of value that far exceeds traditional commerce.

This work was previously published in Encyclopedia of Artificial Intelligence, edited by J. Dopico; J. de la Calle; A. Sierra, pp. 940-944, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.4
Intelligent Software Agents Analysis in E–Commerce II

Xin Luo
The University of New Mexico, USA

Somasheker Akkaladevi
Virginia State University, USA

ISA OPPORTUNITIES AND LIMITATIONS IN E-COMMERCE

Cowan et al. (2002) argued that the human cognitive ability to search for information and to evaluate their usefulness is extremely limited in comparison to those of computers. In detail, it's cumbersome and time-consuming for a person to search for information from limited resources and to evaluate the information's usefulness. They further indicated that while people are able to perform several queries in parallel and are good at drawing parallels and analogies between pieces of information, advanced systems that embody ISA architecture are far more effective in terms of calculation power and parallel processing abilities, particularly in the quantities of material they can process (Cowan et al. 2002). According to Bradshaw (1997), information complexity will continue to increase dramatically in the coming decades. He further contended that the dynamic and distributed nature of both data and applications require that software not merely respond to requests for information but intelligently anticipate, adapt, and actively seek ways to support users.

E-commerce applications based on agent-oriented e-commerce systems have great potential. Agents can be designed using the latest web-based technologies, such as Java, XML, and HTTP, and can dynamically discover and compose E-services and mediate interactions to handle routine tasks, monitor activities, set up contracts, execute business processes, and find the best services (Shih et al., 2003). The main advantages of using these technologies are their simplicity of usage, ubiquitous nature, and their heterogeneity and platform independence (Begin and Boisvert, 2002). XML will likely become the standard language for agent-oriented E-commerce interactions to encode exchanged messages, documents, invoices, orders, service descriptions, and other information. HTTP, the dominant WWW

protocol, can be used to provide many services, such as robust and scalable web servers, firewall access, and levels of security for these E-commerce applications.

Agents can be made to work individually, as well as in a collaborative manner to perform more complex tasks (Franklin and Graesser, 1996). For example, to purchase a product on the Internet, a group of agents can exchange messages in a conversation to find the best deal, can bid in an auction for the product, can arrange financing, can select a shipper, and can also track the order. Multi-agent systems (groups of agents collaborating to achieve some purpose) are critical for large-scale e-commerce applications, especially B2B interactions such as service provisioning, supply chain, negotiation, and fulfillment, etc. The grouping of agents can be static or dynamic depending on the specific need (Guttman et al., 1998b). A perfect coordination should be established for the interactions between the agents to achieve a higher-level task, such as requesting, offering and accepting a contract for some services (Guttman et al., 1998a).

There are several agent toolkits publicly available which can be used to satisfy the customer requirements and ideally they need to adhere to standards which define multi-party agent interoperability. For example, fuzzy logic based intelligent negotiation agents can be used to interact autonomously and consequently, and save human labor in negotiations. The aim of modeling a negotiation agent is to reach mutual agreement efficiently and intelligently. The negotiation agent should be able to negotiate with other such agents over various sets of issues, and on behalf of the real-world parties they represent, i.e. they should be able to handle multi-issue negotiations at any given time.

The boom in e-commerce has now created the need for ISAs that can handle complicated online transactions and negotiations for both sellers and buyers. In general, buyers want to find sellers that have desired products and services. And they want to find product information and gain expert advice before and after the purchase from sellers, which, in turn, want to find buyers and provide expert advice about their product or service as well as customer service and support. Therefore, there is an opportunity that both buyers and sellers can automate handling this potential transaction by adopting ISA technology. The use of ISAs will be essential to handling many tasks of creating, maintaining, and delivering information on the Web. By implementing ISA technology in e-commerce, agents can shop around for their users; they can communicate with other agents for product specifications, such as price, feature, quantity, and service package, and make a comparison according to user's objective and requirement and return with recommendations of purchases, which can meet those specifications; they can also act for sellers by providing product or service sales advice, and help troubleshoot customer problems by automatically offering solutions or suggestions; they can automatically pay bills and keep track of the payment.

Looking at ISA development from an international stand point, the nature of Internet in developed countries, such as USA, Canada, West Europe, Japan, and Australia, etc. and the consequent evolution of e-commerce as the new model provide exciting opportunities and challenges for ISA-based developments. Opportunities include wider market reach in a timely manner, higher earnings, broader spectrum of target and potential customers, and collaboration among vendors. This ISA-powered e-commerce arena would be different than our traditional commerce, because the traditional form of competition can give way to collaborative efforts across industries for adding value to business processes. This means that agents of different vendors can establish a cooperative relationship to communicate with each other via XML language in order to set up and complete transactions online.

Technically, for instance, if an information agent found that the vendor is in need of more

airplane tickets, it would notify a collaborative agent to search for relevant information regarding the ticket in terms of availability, price, and quantity etc. from other sources over the Internet. In this case, the collaborative agent would work with mobile agents and negotiate with other agents working for different vendors and obtain ticket information for its user. It would be able to provide the user with the result of the search, and, if needed, purchase the tickets for the user if certain requirements can be met. In the meantime, interface agents can monitor the user's reaction and decision behavior, and would provide the user with informational assistance in terms of advice, recommendation, and suggestion for any related and similar transactions.

On the other hand, however, this kind of intelligent electronic communication and transaction is relatively inapplicable in traditional commerce where different competitive vendors are not willing to share information with each other (Maes et al., 1999). The level of willingness in ISA-based e-commerce is, however, somewhat limited due to sociological and ethical factors, which will be discussed later in this paper. In addition, designing and implementing ISA technology is a costly predicament preventing companies from adopting this emerging tool. Companies need to invest a lot of money to get the ISA engine started. Notwithstanding the exciting theoretical benefits discussed above, many companies are still not sure about how much ISA technology can benefit themselves in terms of revenue, ROI, and business influence in the market where other players are yet to adopt this technology to cooperate with each other. Particularly, medium or small size companies are reluctant to embark on this arena mainly due to the factor of cost.

Additionally, lack of consistent architectures in terms of standards and laws also obstructs the further development of ISA technology (He et al., 2003). In detail, IT industry has not yet finalized the ISA standards, as there are a number of proprietary standards set by various companies. This causes a confusion problem for ISAs to freely communicate with each other. Also, related to standards, relevant laws have not surfaced to regulate how ISAs can legally cooperate with each other and represent their human users in the cyber world.

Additionally, ISA development and deployment is not a global perspective (Jennings et al. 1998). Despite the fact that ISA technology is an ad-hoc topic in developed countries, developing countries are not fully aware of the benefits of ISA and therefore have not deployed ISA-based systems on the Web because their e-commerce development levels and skills are not as sophisticated or advanced as those of the developed countries. This intra-national limitation among developed and developing countries unfortunately hinders agents from freely communicating with each other over the globally connected Internet.

SOCIOLOGICAL AND ETHICAL CHALLENGES

In the preceding sections of this paper, the technical issues involved in agent development have been addressed. However, in addition to these issues, there are also a range of social and cyber-ethical problems, such as trust and delegation, privacy, responsibility, and legal issues, which will become increasingly important in the field of agent technology (Bradshaw 1997; Jennings et al. 1998; Nwana 1996b).

- **Trust and delegation:** For users who want to depend on ISA technology to obtain desired information, they must trust agents which autonomously delegate for users to do the job. It would take time for users to get used to their agents and gain confidence in the agents that work for them. And users have to make a balance between agents continually seeking guidance and never seeking guidance. Users might need to set proper limitations for their

agents, otherwise agents might surpass their authorities.

- **Privacy:** In the explosive information society, security is becoming more and more important. Therefore, users must make sure that their agents always maintain their privacy in the course of transactions. Electronic agent security policies may be needed to encounter this potential threat.
- **Responsibility:** Users need to seriously consider how much responsibility the agents need to carry regarding the transaction pitfall. To some extent, agents are rendered responsibility to get the desired product/service for their users. If the users are not satisfied with the transaction result, they may need to redesign or reprogram the agent rather than directly blame the fault on electronic agents.
- **Legal issues:** In addition to responsibility, users should also think about any potential legal issues triggered by their agents, which, for instance, offer inappropriate advice to other agents resulting in liabilities to other people. This would be very challenging to the ISA technology development, and the scenario would be complicated since the current law does not specify which party (the company who wrote the agent, the company who customized and used the agent, or both) should be responsible for the legal issues.
- **Cyber-ethical issues:** Eichmann (1994) and Etzioni & Weld (1994) proposed the following etiquettes for ISAs which gather information on the Web.
    - Agents must identify themselves;
    - They must moderate the pace and frequency of their requests to some server;
    - They must limit their searches to appropriate servers;
    - They must share information with others;
    - They must respect the authority placed on them by server operators;
    - Their services must be accurate and up-to-date;
    - Safety: they should not destructively alter the world;
    - Tidiness: they should leave the world as they found it;
    - Thrift: they should limit their consumption of scarce resources;
    - Vigilance: they should not allow client actions with unanticipated results.

CONCLUSION AND FUTURE WORK

ISA technology has to confront the increasing complexity of modem information environments. Research and development of ISAs on the Internet is crucial for the development of next generation in open information environments. Sociological and cyber-ethical issues need to be considered for the next generation of agents in e-commerce system, which will explore new types of transactions in the form of dynamic relationships among previously unknown parties (Guttman et al. 1998b). According to Nwana (1996a), the ultimate ISA's success will be the acceptance and mass usage by users, once issues such as privacy, trust, legal, and responsibility are addressed and considered when users design and implement ISA technologies in e-commerce and emerging commerce, such as mobile commerce (M-commerce) and Ubiquitous commerce (U-commerce). It is expected that future research can further explore how ISAs are leveraged in these two newly emerged avenues.

REFERENCES

Begin, L., and Boisvert, H. "Enhancing the value proposition Via the internet," *International Conference on Electronic Commerce Research (ICECR-5),* 2002.

Bradshaw, J.M. "Software Agents," *online: http://agents.umbc.edu/introduction/01-Bradshaw.pdf*) 1997.

Cowan, R., and Harison, E. "Intellectual Property Rights in Intelligent-Agent Technologies: Facilitators, Impediments and Conflicts," *online: http://www.itas.fzk.de/e-society/preprints/ecommerce/CowanHarison.pdf*) 2002.

Eichmann, D. "Ethical Web Agents," *Second International World-Wide Web Conference: Mosaic and the Web*, October 18-20 1994, pp 3-13.

Franklin, S., and Graesser, A. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag*, 1996.

Etzioni, O., and Weld, D. "A Softbot-Based Interface to the Internet," *Communications of the ACM*, July 1994, pp 72-76.

Guttman, R., Moukas, A., and Maes, P. "Agent-mediated Electronic Commerce: A Survey," *Knowledge Engineering Review* (13:3), June 1998a.

Guttman, R., Moukas, A., and Maes, P. "Agents as Mediators in Electronic Commerce," *International Journal of Electronic Markets* (8:1), February 1998b, pp 22-27.

He, M., Jennings, N.R., and Leung, H.-F. "On Agent-Mediated Electronic Commerce," *IEEE Transactions on Knowledge and Data Engineering* (15:4), July/August 2003.

Jennings, N.R., and Wooldridge, M. "Applications of Intelligent Agents," *in Agent Technology: Foundations, Applications, and Markets*,1998, pp 3-28.

Maes, P., Guttnab, R.H., and Moukas, A.G. "Agents That Buy and Sell. (software agents for electronic commerce)(Technology Information)," *Communications of the ACM* (42:3) 1999, p 81.

Ndumu, D., and Nwana, H. "Research and Development Challenges for Agent-Based Systems," *IEE Proceedings on Software Engineering* (144:01), January 1997.

Nwana, H.S. "Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a prediction of (near-) future developments," *online: http://www.hermans.org/agents/index.html*), July 1996a.

Nwana, H.S. "Software Agents: An Overview," *online: http://agents.umbc.edu/introduction/ao/*) 1996b.

Shih, T.K., Chiu, C.-F., and Hsu, H.-h. "An Agent-Based Multi-Issue Negotiation System in E-commerce," *Journal of Electronic Commerce in Organizations* (1:1), Jan-March 2003, pp 1-16.

KEY TERMS

Agent: A computer system situated in some environment that is capable of autonomous action in this environment to meets its design objective.

Business-to-Business E-Commerce: Electronic transaction of goods or services between businesses as opposed to that between businesses and other groups.

Business-to-Customer E-Commerce: Electronic or online activities of commercial organizations serving the end consumer with products and/or services. It is usually applied exclusively to e-commerce.

Customer-to-Customer E-Commerce: Online transactions involving the electronically-facilitated transactions between consumers through some third party.

Electronic Commerce (E-Commerce): Consists of the buying and selling of products or

services over electronic systems such as the Internet and other computer networks. A wide variety of commerce is conducted in this way, including electronic funds transfer, supply chain management, e-marketing, online transaction processing, and automated data collection systems.

Intelligent Software Agent: A software agent that uses Artificial Intelligence (AI) in the pursuit of the goals of its clients.

Ubiquitous Commerce (U-Commerce): The ultimate form of e-commerce and m-commerce in an 'anytime, anywhere' fashion. It involves the use of ubiquitous networks to support personalized and uninterrupted communications and transactions at a level of value that far exceeds traditional commerce.

This work was previously published in Encyclopedia of Artificial Intelligence, edited by J. Dopico; J. de la Calle; A. Sierra, pp. 945-949, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.5
A Semantic Web–Based Information Integration Approach for an Agent–Based Electronic Market

Maria João Viamonte
GECAD – Knowledge Engineering and Decision Support Research Group, Porto Polytechnic Institute, Portugal

Nuno Silva
GECAD – Knowledge Engineering and Decision Support Research Group, Porto Polytechnic Institute, Portugal

ABSTRACT

With the increasing importance of e-commerce across the Internet, the need for software agents to support both customers and suppliers in buying and selling goods/services is growing rapidly. It is becoming increasingly evident that in a few years the Internet will host a large number of interacting software agents. Most of them will be economically motivated, and will negotiate a variety of goods and services. It is therefore important to consider the economic incentives and behaviours of e-commerce software agents, and to use all available means to anticipate their collective interactions. Even more fundamental than these issues, however, is the very nature of the various actors that are involved in e-commerce transactions. This leads to different conceptualizations of the needs and capabilities, giving rise to semantic incompatibilities between them. Ontologies have an important role in Multi-Agent Systems communication and provide a vocabulary to be used in the communication between agents. It is hard to find two agents using precisely the same vocabulary. They usually have a heterogeneous private vocabulary defined in their own private ontology. In order to provide help in the conversation among different agents, we are proposing what we call ontology-services to facilitate agents' interoperability. More specifically, we

propose an ontology-based information integration approach, exploiting the ontology mapping paradigm, by aligning consumer needs and the market capacities, in a semi-automatic mode. We propose a new approach for the combination of the use of agent-based electronic markets based on Semantic Web technology, improved by the application and exploitation of the information and trust relationships captured by the social networks.

CURRENT SITUATION

As the result of technological developments, e-commerce, namely business-to-consumer (B2C), is emerging as the new way of doing business.

In most current (first generation) e-ecommerce applications, the buyers are generally humans who typically browse through a catalogue of well-defined commodities (e.g., flights, books, compact discs, computer components) and make (fixed price) purchases (often by means of a credit card transaction). However, this modus operandi is only scratching the surface of what is possible. By increasing the degree and the sophistication of automation, on both the buyer's and the seller's side, e-commerce becomes much more dynamic, personalized, and context sensitive.

We believe that over the course of the next decade, the global economy and the Internet will merge into a global market with a large amount of autonomous software agents that exchange goods and services with humans and other agents. Agents will represent, and be, consumers, producers, and intermediaries.

When interactions among agents become sufficiently rich, a crucial qualitative change will occur. New classes of agents will be designed specially to serve the needs of the other agents. However, in order to harness the full potential of this new mode of e-commerce, a broad range of social, legal, and technical issues need to be addressed. These issues relate to things such as security, trust, payment mechanisms, advertising, logistics, and back office management. Even more fundamental than these issues, however, is the very nature of the various actors that are involved in e-commerce transactions.

In an efficient agent-mediated electronic market, where all the partners, both sending and receiving messages have to lead to acceptable and meaningful agreements, it is necessary to have common standards, like an interaction protocol to achieve deals, a language for describing the messages' content and ontologies for describing the domain's knowledge.

The need for these standards emerges due to the nature of the goods/services traded in business transactions. The goods/services are described through multiple attributes (e.g. price, features and quality), which imply that negotiation processes and final agreements between seller and buyers must be enhanced with the capability to both understand the terms and conditions of the transaction (e.g. vocabulary semantics, currencies to denote different prices, different units to represent measures or mutual dependencies of products). A critical factor for the efficiency of the future negotiation processes and the success of the potential settlements is an agreement among the negotiating parties about how the issues of a negotiation are represented and what this representation means to each of the negotiating parties. This problem is referred to as the ontology problem of electronic negotiations (Ströbel, 2001). Distributors, manufactures, and service providers may have radically different ontologies that differ significantly in format, structure, and meaning. Given the increasingly complex requirements of applications, the need for rich, consistent and reusable semantics, the growth of semantically interoperable enterprises into knowledge-based communities; and the evolution; and adoption of Semantic Web technologies need to be addressed. Ontologies represent the best answer to the demand for intelligent systems that operate closer to the human conceptual level (Obrst, Liu,

and Wray, 2003).

To achieve this degree of automation and move to new generation e-commerce applications, we believe that a new model of software is needed.

PROBLEM STATEMENT

In order to make possible the interaction between agents in a Multi-Agent Systems, it is necessary to have a communication platform, a communication language and a common ontology.

With respect to communications, there are some implications:

- There are many different ontology languages.
- Different ontology languages are sometimes based on different underlying paradigms.
- Some ontology languages are very expressive, some are not.
- Some ontology languages have a formally defined semantics, some do not.
- Some ontology languages have inference support, some do not.

However, even if the exact same language is used, the resulting ontologies can be incompatible in various ways:

- People or agents may use different terms for the same thing.
- People or agents may use the same term for the different things.
- A given notion or concept may be modeled using different primitives in the language.
- A given notion or concept may be modeled with very different fundamental underlying primitives.

Once we take this problem as a challenge, representing these differences in a common ontology becomes essential. The ontology includes the entire domain's knowledge, which is made available to all the components active in an information system (Huhns & Singh, 1997). The use of a common ontology guarantees the consistency (an expression has the same meaning for all the agents) and the compatibility (a concept is designed, for the same expression, for any agent) of the information present in the system. However, we cannot assume that all the agents will use a common ontology. In fact, as stated in (Hepp, 2007), the adoption of ontologies in e-commerce has some drawbacks, namely concerning the comprehension and correct use of the adopted ontology which is a difficult and time consuming task, often leading to conflicting interpretations and eventually wrong use of the ontology and data. Usually, each agent has its heterogeneous private ontology and it cannot fully understand another agent's ontology, giving rise to semantic incompatibilities between them. Consequently, different actors involved in the marketplace must be able to independently describe their universe of discourse, while the market is responsible for providing a technological framework that promotes the semantic integration between parties.

However, it is necessary to consider that the solution proposed to overcome theses problems, has to take into consideration the technological support already existent, namely a well-proven e-commerce platform, where agents with strategic behaviour represent consumers and suppliers.

PROPOSED SOLUTION

We propose an agent-based electronic market with an ontology-based information integration approach, exploiting the ontology mapping paradigm, by aligning consumer needs and the market capacities throughout the negotiation conversations in a semi-automatic mode, improved by the application and exploitation of the information and trust relationships captured by the social networks.

To study our proposal we will combine the use of ISEM and MAFRA Toolkit into a novel electronic marketplace approach, together with the exploitation of social semantic network services.

ISEM – Intelligent System for Electronic MarketPlaces (Viamonte, Ramos, Rodrigues & Cardoso, 2006) is a simulation platform developed at our research group. ISEM is a multi-agent market simulator, designed for analysing agent market strategies based on a complete understanding of buyer and seller behaviours, preference models and pricing algorithms, considering user risk preferences and game theory for scenario analysis. Each market participant has its own business objectives and decision model. The results of the negotiations between agents are analyzed by data mining algorithms in order to extract knowledge that gives agents feedback to improve their strategies. The extracted knowledge will be used to set up probable scenarios, analyzed by means of simulation and game theory decision criteria.

The main objectives of ISEM are: first, the ISEM system addresses the complexities of online buyers' behaviour by providing a rich set of behaviour parameters; second, the ISEM system provides available market information that allows sellers to make assumptions about buyers' behaviour and preference models; third, the different agents customise their behaviour adaptively, by learning each user's preference models and business strategies. The learning agent capacity is achieved through data mining techniques applied on-line during the market sessions within the ISEM system.

MAFRA Toolkit is the instantiation of the MAFRA-MApping FRAmework (Maedche, Motik, Silva & Volz, 2002), addressing the fundamental phases of the ontology mapping process. In particular, it allows the identification, specification and representation of semantic relations between two different ontologies. These semantic relations are further applied in the execution phase of the interoperation, by transforming the data (messages' content) as understood by one of the actors into the data understood by the other. In this sense, ontology mapping allows actors to keep their knowledge bases unchanged while supporting the semantic alignment between their conceptualizations (ontologies).

On the other hand, social network repositories aim to capture collaboratively created information and trust relationships between individuals according to different subjects (e.g. business, family, music, sport, travel, and hobbies). Social networks together with security oriented technologies form the basic infrastructure to encompass collaboration and trust relationships in many internet based activities, including e-commerce transactions and information integration.

Objectives

The proposed approach introduces a novel agent-based marketplace architecture that will be deployed within the ISEM simulator in order to assure that different agents can establish deals in a more flexible and evolved form. In order to judge the feasibility and relevance of the approach, we will use the Consumer Buying Behaviour model (CBB) as a reference (Runyon & Stewart, 1987).

Another important goal is to support the identification stage of the CBB model using ontologies to construct the most accurate model of the consumer's needs. Moreover, at the product brokering, buyer coalition formation, merchant brokering and negotiation stages, the ontology mapping process will provide the integration of the seller and consumer's models (HarmoNet, 2007).

Complementarily, the repository of relationships provided by emergent social networks will support establishing more accurate trust relationships between businesses and customers, as well as providing a better alignment (mapping) between their models. This new information is very important to feed the agents' knowledge

bases to improve their strategic behaviour. Market participant's strategic behaviour is very significant in the context of competition.

Overview

In the following sections, we will describe the electronic market objectives and model, the ontology mapping features and the proposed ontology-based, socially-driven information integration approach.

Our ontology-based services will be provided through an additional Market service provided by a specific agent, the Market Facilitator agent. This agent will be responsible for providing structural and semantic relationships between different vocabularies, which carry appropriate conversations and make agreement possible.

The Marketplace Objectives

Agents can be used in many ways in the electronic commerce applications and for this it will be necessary to create a scenario where they can interact with each other. The Marketplace function is to permit and facilitate the interaction between the agents. We support the idea of a Web site where users go to trade items. All agents are notified by the market about existing buying agents and selling agents and about what they want to sell and what they want to buy. We expect that we have agents interested in buying and selling the same category of commodities. At any time, we can have a variable number of partners in our market and every transaction has specific parameters.

A language support for inter-agent communication has to be defined; we need to ensure that all the agents participating in the market use the same language, or that the languages in use can be translated; and an ontology-based information integration to ensure transparent semantic interoperability.

Another objective is supporting some of the stages of the CBB model:

- *Need Identification*, "the consumer can be stimulated through product information". Although all the buyer agents are notified by the market about existing selling agents and about their products, the marketplace needs some additional expertise to notify the users about products available based on their profiles and last deals.
- *Product Brokering*, "the evaluation of product alternatives based on consumer-provided criteria." The agent system must be able to assist consumers in deciding which products best fit their personal criteria, they must act as recommendation agents, make predictions based on profiles and "business intelligence", possibly derived by data mining techniques and based on social-driven information.
- *Buyer Coalition Formation*, "having determined the product to buy, customers may move directly to the merchant brokering phase (see below) or they may interact with other similar buyers to try and form a coalition before moving to the merchant brokering phase. Here, a coalition is viewed as a group of agents cooperating with each other in order to achieve a common task. In these "buyer coalitions," each buyer is represented by their own agent and together these agents try to form a grouping in order to approach the merchant with a larger order (in order to obtain leverage by buying in bulk). Normally a buyer coalition model is composed of five stages: negotiation, leader election, coalition formation, payment collection, and execution stages. It is essential to have a trustworthy and reliable agent that will collect the buyer's information, divide the agents into coalitions, and negotiate with sellers (refer to Yamamoto & Sycara, 2001 and Tsvetovat & Sycara, 2000 for a full discussion of these issues).
- *Merchant Brokering*, "who to buy from, includes the evaluation of merchant alterna-

tives based on consumer-provided criteria". Having selected the desired product, and perhaps after having formed a buyer coalition, merchant brokering involves the agent finding an appropriate seller to purchase the item from. The customer agent must be able to find several providers for each item, taking into account different user preferences and based on past events and users satisfaction with certain providers.

- *Negotiation*, "this stage is about how to determine the terms of the transaction". Having selected a merchant (or set of merchants), the next step is to negotiate the terms and conditions. We believe that this is one of the major changes that will be brought about by agent-mediated e-commerce. The agents negotiate with each other by using strategies based on rule systems and users' criteria, searching for an agreement on multiple goals and considering tradeoffs between the goals.

The Marketplace Model

Our Marketplace facilitates agent meeting and matching, besides supporting the negotiation model. In order to have results and feedback to improve the negotiation models and consequently the behaviour of user agents, we simulate a series of negotiation periods, D={1,2,…,n}, where each one is composed by a fixed interval of time T={0,1,…,m}. Furthermore, each agent has a deadline $D_{\max}^{Agt} \in D$ to achieve its business objectives. At a particular negotiation period, each agent has an objective that specifies its intention to buy or sell a particular good or service and on what conditions. The available agents can establish their own objectives and decision rules. Moreover, they can adapt their strategies as the simulation progresses on the basis of previous efforts' successes or failures. The simulator probes the conditions and the effects of market rules, by simulating the participant's strategic behaviour.

Our simulator was developed based on "A Model for Developing a MarketPlace with Software Agents (MoDeMA)" (Viamonte, 2004). MoDeMA is composed of the following steps:

- Marketplace model definition, that permits doing transactions according to the CBB model.
- Identification of the different participants, and the possible interactions between them.
- Ontology specification, that identifies and represents items being transacted.
- Agents' architecture specification, and information flows between each agents module.
- Knowledge Acquisition, defining the process that guarantees the agent the knowledge to act in pursuit of its role.
- Negotiation Model, defining the negotiation mechanisms to be used.
- Negotiation Protocol, specification of each of the negotiation mechanism rules.
- Negotiation Strategies, specification and development of several negotiation strategies.
- Knowledge Discovery, identification and gathering of market knowledge to support agents' strategic behaviour.

ISEM is flexible; the user completely defines the model he or she wants to simulate, including the number of agents, each agent's type and strategies.

The Negotiation Model and Protocol

The negotiation model used in ISEM is bilateral contracting where buyer agents are looking for sellers that can provide them with the desired products at the best conditions.

We adopt what is basically an alternating protocol (Fatima, Wooldridge & Jennings, 2004 and Osborne & Rubinstein, 1994).

Negotiation starts when a buyer agent sends a request for proposal (RFP) (Figure 1). In response, a seller agent analyses its own capabilities, current availability, and past experiences and formulates a proposal (PP).

Sellers can formulate two kinds of proposals: a proposal for the product requested or a proposal for a related product, according to the buyer preference model.

On the basis of the bilateral agreements made among market players and lessons learned from previous bid rounds, both agents revise their strategies for the next negotiation round and update their individual knowledge module.

The negotiation protocol of the ISEM simulator (Figure 2) has three main actors:

- *Buyer* (B) is the agent that represents a consumer or a buyer coalition. Multiple Buyers normally exist in the marketplace in an instant.
- *Seller* (S) is the agent that represents a supplier. Multiple Sellers normally exist in the marketplace in an instant.
- *Market Facilitator agent* (MF), usually one per marketplace, coordinates the market and ensures that it works correctly. MF identifies all the agents in the market, regulates negotiation, and assures that the market operates according to established rules. Before entering the market, agents must first register with the MF agent.

The Ontology

Ontologies, as knowledge representation artefacts, allow the "explicit formal specification of a shared conceptualization" (Studer, Benjamins

Figure 1. Sequence of bilateral contracts

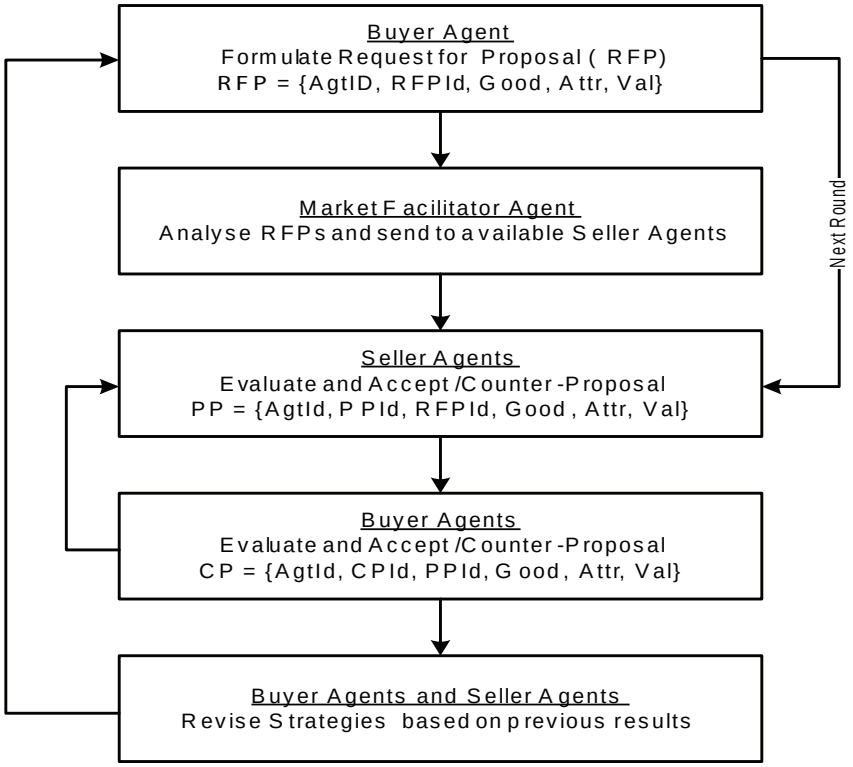
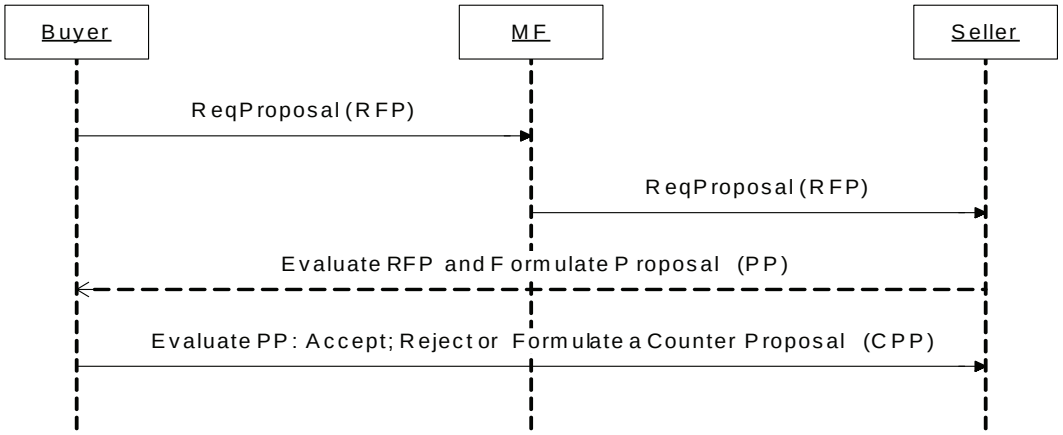


Figure 2. ISEM bilateral contract protocol



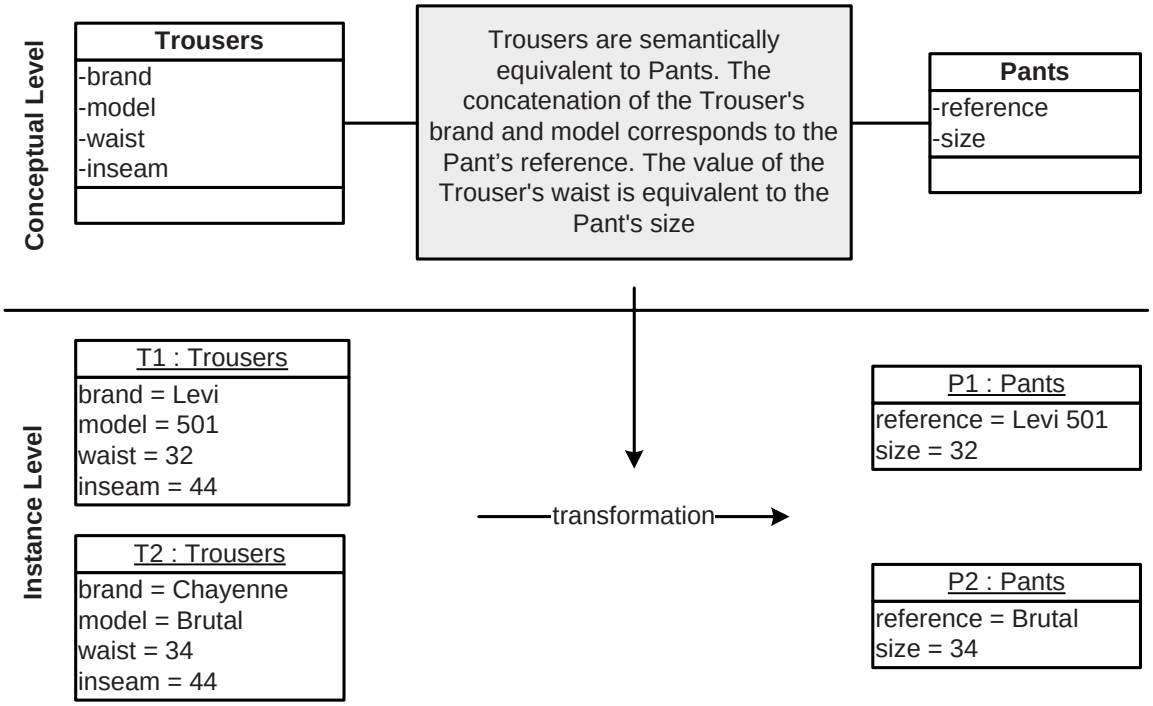
& Fensel, 1998). Ontologies raise the level of specification, incorporating semantics into the data, and promoting its exchange in an explicit understandable form. The resulting artefact is then shared between interoperating actors and is used for querying and reasoning about each others knowledge base. Ontologies are therefore fully geared as a framework for capturing, specifying, representing and sharing the domain of discourse in the scope of distinct types of e-commerce, namely B2C or business-to-business (B2B).

However, ontologies per se are not able to overcome information interoperability incompatibilities and it is hardly conceivable that a single ontology can be applied to all kinds of domains and applications. Information integration mechanisms are necessary in order to identify, represent and apply the diverse relationships between concepts and relations of two or more ontologies. Ontology Mapping is one of the most successful information integration paradigms used in the Internet and in particular in the Semantic Web. Despite a more holistic description of the Ontology Mapping Process (Maedche et al., 2002), ontology mapping is primarily the process whereby semantic relations are defined at an ontological level between source ontology entities and target ontology entities and then further applied at instance level, transforming source ontology instances into target ontology instances (Figure 3).

Therefore, unlike other information integration paradigms (e.g. Merging and Integration) (Pinto, Gómez-Pérez & Martins, 1999) that lead to definitive semantics and information loss, through ontology mapping, repositories are kept separated, independent and distinct, maintaining their complete semantics and contents.

The approach followed in MAFRA Toolkit adopts a declarative specification of semantic relations through the use of the SBO-Semantic Bridging Ontology, itself an ontology. When instantiated it becomes the description of the syntactic and semantic relations between entities of two ontologies. MAFRA Toolkit's support of the SBO manipulation hides the procedural complexity of specification and execution of relations. Additionally, its open Service-oriented architecture allows the incorporation of new types of mapping relations into the system (Services), improving the expressive power and mapping capabilities of the system. Additionally, Services play the important role of driving the semi-automatic specification of semantic relations, releasing the user from this time consuming and difficult task. Instead, the user is invited to supervise the

Figure 3. Informal representation of ontology mapping



services' process and confirm the automatically generated proposals.

MAFRA Toolkit has been successfully adopted in the EU-funded project Harmo-Ten (www.harmo-ten.org) devoted to the exchange of information in tourism. The outcome of this project is now being applied in the HarmoNet initiative (HarmoNet, 2007).

The Social Networks

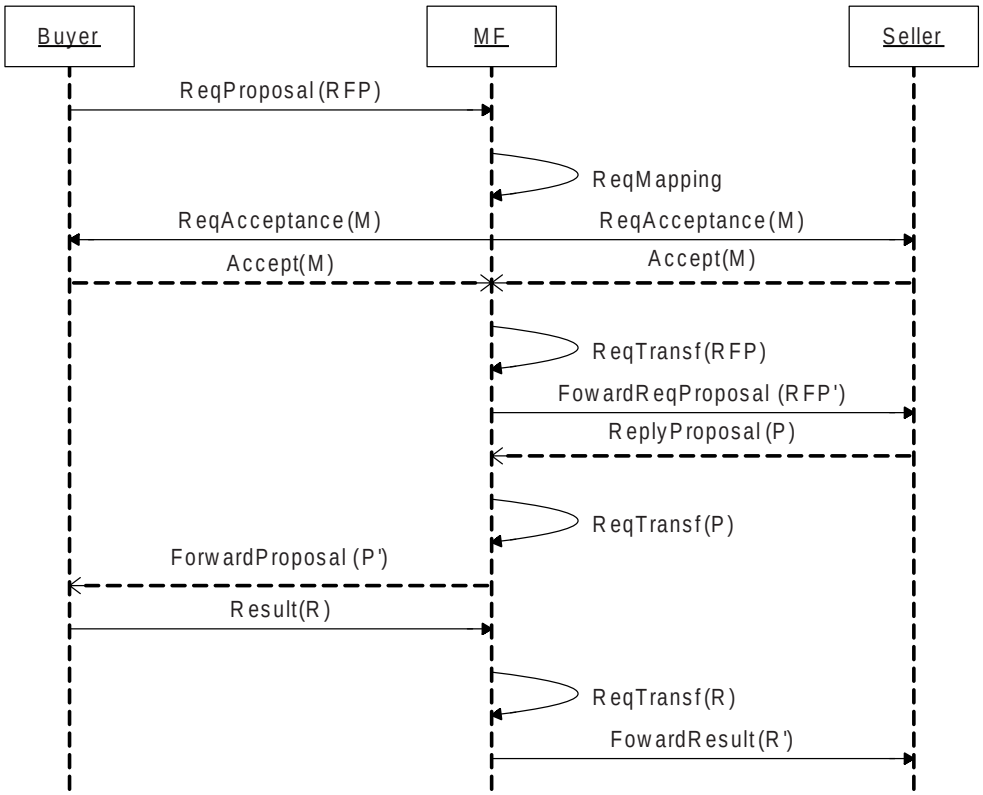
Social Networks are one of the grounding theories behind the promotion and success of Web 2.0. Social Networks connect people around issues or causes. Different interests give rise to and promote different communities (e.g. music, sports, technology, religion, fashion), which simplify many customer target initiatives such as pricing, marketing, advertising and selling. Social Network Analysis (SNA) unveils clusters of users with similar interests, even when such relations are vague or implicit. Based on users' manually created relationships, SNA provides important formal metrics such as Activity, Betweenness and Closeness. Yet, either explicit social relationship expressed through social network tools (e.g. LinkedIn, MySpace, Friendster) or through email analysis, the so called social network mapping tools are emerging that are capable of exposing relevant links between users (e.g. InFlow) and generate statistics on almost anything. As often stated in many blogs and editorials, business managers expect that Web 2.0 will bring more information about users, customers and their trends, allowing business efforts to focus on specific target communities.

Through tagging and social bookmarking, folksonomies are emerging as a valuable clas-

sification mechanism of documents and goods. Moreover, folksonomies, seen as very simple ontologies (Van Damme, Hepp & Siorpaes, 2007) are perceived as an important social interrelation mechanism allowing systems to track the relationships between individuals and groups across the social networks. This assumption is based on empirical evidence showing that users follow the tendencies originated by specific users (or groups of users). With this approach, i.e. decentralized, cooperative and intuitive development (Siorpaes & Hepp, 2007), folksonomies tend to better represent the conceptualization of a community, while at the same time the community is able to cooperatively participate in other ontology-based processes, such as information integration and evolution. This permeating process promotes discussion about future communities' tendencies, their similar desires, interests and knowledge patterns, supporting the idea of exploiting the intrinsic social knowledge for improving the e-commerce interactions.

It is expected that social network services will soon provide automatic access to SNA metrics, improved content classification and many statistical reports. For that, API, agent-based or Web Services will be used instead of the traditional Web page interaction. The Web-Of-Trust and PKI (Public Key Infrastructure) (Aberer, Datta & Hauswirth, 2005) infrastructures will be combined with such services in order to assure identity, trust and no repudiation between users and systems. Kondifi and FOAFRealm are two examples of such systems. The FOAFRealm aims to integrate several types of social semantic information sources through the same infrastructure,

Figure 4. The information integration basic protocol



and its access control rules are based in FOAF data (http://www.foaf-project.org/). Kondifi deploys and exploits relations between the formal identification mechanisms available through PKI infrastructure, and RDF-based (Resource Description Framework) information provided by accessed documents.

SOLUTION DETAILS

While the use of ontologies allows e-commerce actors to describe their needs and capabilities into proprietary repositories, the use of the ontology-mapping paradigm allows transparent semantic interoperability between them. This is the technological basis for the alignment between needs and capabilities of consumer and supplier, even when they use different ontologies. Based on this approach we can obtain the minimal requirements to support our proposed solution.

Minimalist Approach

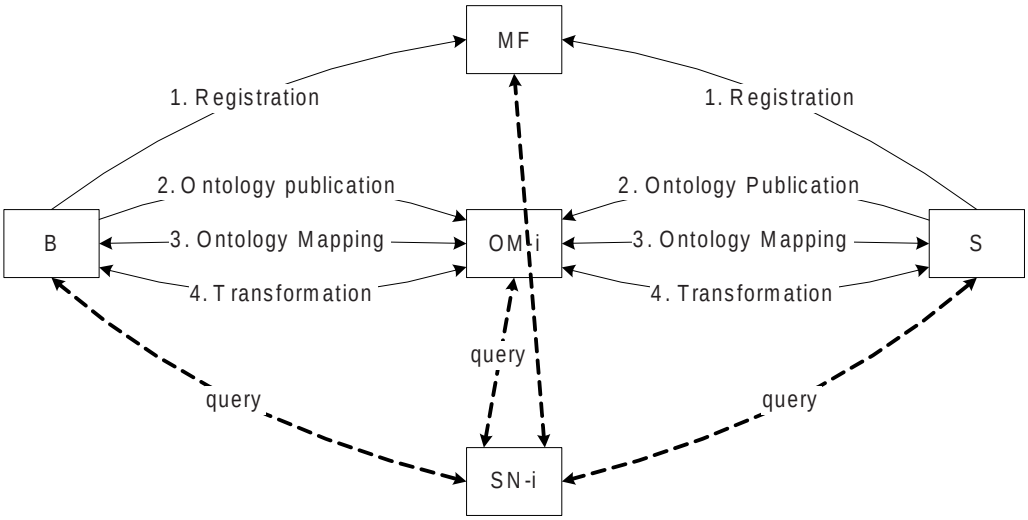
The first proposed version of the protocol (Figure 4) models the simplest interactions between the three types of agents, minimizing the changes in interactions between B (Buyer) and S (Seller) agents with the MF (Market Facilitator) agent, based on the ISEM protocol (Figure 2).

This minimal approach aims to overcome the simplest information integration problems mentioned above.

In this protocol, the MF agent is responsible for the specification of the mapping between the B and S ontologies. When the B requests a proposal (ReqProposal), the MF agent generates (or queries) the ontology mapping repository for a mapping between the two agents' ontologies, and proposes it to the agents. The agents are free to accept or reject the mapping, but in case both accept, the messages' content will be transformed between agents according to that ontology mapping specification.

The changes to the original protocol are minimal. In fact, most of the changes affect the MF agent only, in the sense that the ReqMapping and ReqTransf messages will trigger MF's internal processes.

Figure 5. Marketplace's actors and interactions



Extended Approach

However, this system infrastructure is too simple to be effective especially because:

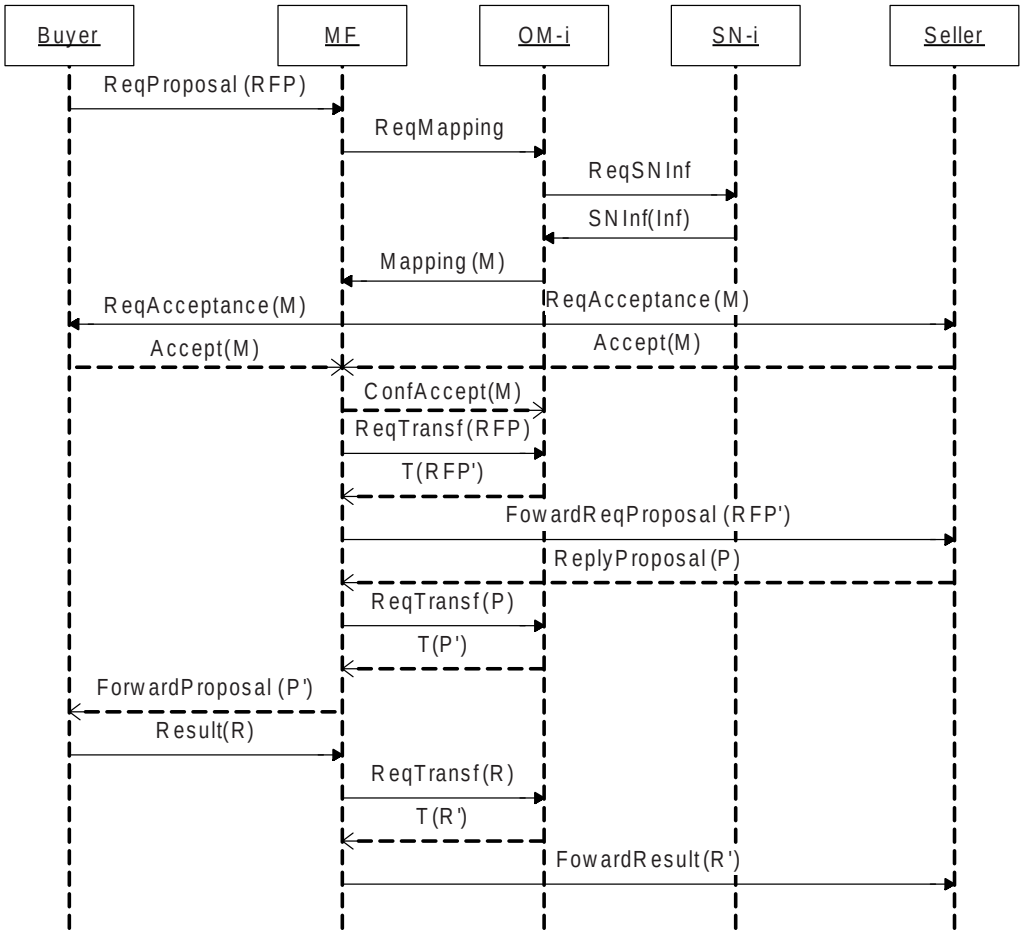
- It neglects the complexity and subjectivity of the ontology mapping process.
- It does not consider the social relationships emerging in the Web and their potential in disambiguation and decision-making processes.

As a consequence, a new system infrastructure is proposed, recognizing two new types of actors:

- *Ontology Mapping Intermediary* (OM-i), is the agent that supports the information integration process during the market interoperability, typically one per marketplace.
- *Social Networks Intermediary* (SN-i), is the agent that provides trust relationship information holding between B, S and other agents that undertake similar experiences (e.g. a trader agent), typically one per marketplace.

These actors deploy a set of relationship types whose goal is to automate and improve the quality of the results achieved in the e-commerce transactions. Figure 5 depicts the types of interactions

Figure 6. The integration protocol



between the marketplace supporting agents (i.e. MF, OM-i and SN-i agents) and the operational agents (i.e. B and S).

Considering the previous descriptions, a more complete and complex protocol is now detailed, including the OM-i and SN-i agents in the system (Figure 6).

The integration starts when the B agent sends a request for proposal message (ReqProposal) to the MF agent. In response, the MF sends to the OM-i a request for mapping message (ReqMapping) between B and S's ontologies.

Once OM-i receives the ReqMapping message, it will start the ontology mapping specification process, with the support of other entities, including matching agents, ontology mapping repositories and SN-i. SN-i is responsible for retrieving the relevant information from ontology mapping repositories and social networks. Past similar ontology mapping experiences undertaken by agents with trust relationships with B and S will be used by SN-i to compile the social network repository information (i.e. SNInf(Inf)). Because the ReqSNInf is the sole responsibility of OM-i, both B and S are advised to perform a similar verification (eventually using other SN-i) once the ontology mapping is submitted for acceptance (i.e. ReqAcceptance(M)). Despite the fact that figure 6 represents only the acceptance scenario, a rejection scenario is also possible, in which case no further interaction will occur between B and S. In case the mapping is accepted, MF resumes the protocol by requesting to OM-i the RFP data transformation. Using the ontology mapping document, RFP data represented according to B's ontology is transformed into data represented according to S's ontology. The transformed data (RFP') is forwarded to S, which will process it and will reply to MF. MF will then request the transformation of the proposal data (P) and will forward P' to B. B processes it and will accept or formulate a counter-proposal (CP). As can be seen, once a mutually acceptable ontology mapping is established between B's ontology and S's ontology, all messages exchanged between B and S through MF are forwarded to OM-i for transformation.

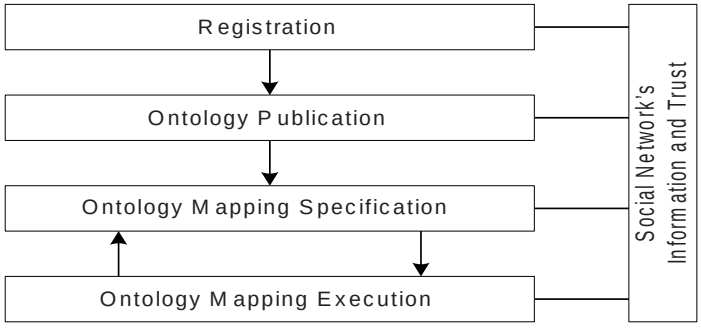
Notice that Figure 6 represents one single S in the system, but in fact multiple S's capable of replying to the request may exist in the marketplace. In such case, the protocol would replicate the previous protocol for as many capable S's. In order to decide which S's are capable of answering the request, a simple approach based on a keyword matching algorithm is taken. The B agent specifies a few keywords along with its formal request (RFP). The MF, with the aid of SN-i, matches this list against every S's publicized keyword list. In case the match succeeds to a certain level, the S is classified as capable.

The interaction protocol just described emphasises the information integration problem occurring in the negotiation stage of the CBB model. However, other interaction protocols are needed to improve the marketplace in accordance to the objectives stated in section 2.1. In fact, in every stage of the CBB model, both the SN-i and OM-i are major players in the proposed solution. Notice that the social network information and trust component of the system is orthogonal to previous processes, as depicted in Figure 5. Also notice that the trust component of the system is orthogonal to previous processes, as depicted in Figure 7.

In particular, the Social Network component is envisaged as a source of information for disambiguation and decision-making to the other processes, along with trust relationships between users and groups:

- The Registration process will profit from the Trust component in several ways. For example, the S agents can better decide which Services to provide in a marketplace, depending on the segment of customers traditionally found in specific marketplace. This is achieved by the social characterization of the B agents according to the social

Figure 7. Marketplace's ontology-based services



networks they belong to. In the same sense, B agents can more accurately choose the marketplaces to register to, depending on the social network advice, based on a social characterization of the other marketplace participants (i.e. Buyers and Sellers);

- During the Ontology Publication process, agents need to decide which ontologies are advisable in that particular marketplace (e.g. simple or more detailed). The agent is able to choose the ontology that conveniently describes the semantics of its data in a certain context. In order to decide the more convenient ontology, S agents require a social characterization of the marketplace. Similar decisions are taken by B agents. Notice however, that the agent's published ontology should not be understood as the complete representation of its internal data, but the semantics the agent intends to exteriorize through the Ontology Publication process. As a consequence, the agent should encompass the mechanisms allowing the internal transformation between the internal data semantics (e.g. data schema) and the external semantics (ontology), and vice-versa;
- The Ontology Mapping Specification process is typically very ambiguous, thus it can potentially profit from the social characterisation and social trust relationships provided by SN-i. This process is understood as a negotiation process, in which B and S try achieving a consensus about the ontology mapping. The SN-i agent participates in this process as an information provider to the OM-i in order to disambiguate the ontology mapping achieved through automatic mechanisms and protocols. A possible approach for this negotiation can be found in (Silva, Maio & Rocha, 2005);
- The Ontology Mapping Execution process is very systemic (in accordance to the ontology mapping specification document). Yet, the messages' data may be inconsistent in respect to the B's and S's data repository. In such cases, social knowledge is often required in order to decide/correct the consistency of the data. Through the use of social relationships, SN-i is a facilitator of this process.

ALTERNATIVES

Some approaches to agent-based applications for competitive electronic markets are more targeted or limited than our proposal: some of them not address a multi-issue negotiation type; do not consider behaviour dependent dynamic strategies, or expected future reactions. Others, although considering behaviour dependent dynamic strate-

gies, frequently assume that agents have complete information about market, such as the distribution of buyer preferences or its competitor's prices like (Dasgupta & Das, 2000) and (Sim & Choi, 2003), and, in general, all of them assume that actors be aware of understand each other. Nevertheless, we can find approaches where the semantic problems are been considered.

For example, in (Malucelli, Rocha & Oliveira, 2004) ontology-based services are proposed to be integrated in the ForEV architecture in order to help in the Virtual Enterprise formation process (B2B). ForEV is an appropriate computing platform that includes and combines a multi-issue negotiation method in the context of Multi-Agent Systems, which makes the platform more open, enabling the establishment of the negotiation process between agents with different ontologies although representing the same domain of knowledge. They propose an ontology-based services Agent which is responsible for providing the ontology-based services. However we are interested in studying Multi-agent systems for B2C domain where other stages, as advocated by the CBB model, need to be contemplated in order to represent real situations. We are interested in studying how the identification stage of the CBB model can be supported by using ontologies to construct the most accurate model of the consumer's needs. Moreover, at the product brokering, buyer coalition formation, merchant brokering and negotiation stages, the ontology mapping process will provide the integration of the seller and consumer's models and guarantee the agent's heterogeneity. On the other hand, they don't explore the repository of relationships provided by emergent social networks, which can carry out the establishment of more accurate trust relationships between businesses and customers, as well as providing a better alignment (mapping) between their models.

In (Paolucci, Sycara, Nishimura & Srinivasan, 2003) authors present a capabilities matchmaker for applications in the Web, wrapped by Web Services. DAML-S is an ontology of capabilities that is instantiated for describing the capabilities of Web Services, which are then matched against other Web Services' capabilities. Thus a semantic matchmaking approach is adopted, ahead of the syntactic description approach commonly adopted. A very important limitation of the approach is the lack of a decision-making mechanism. In fact, as stated by the authors, "DAML-S requires applications that look more like intelligent software agents than traditional e-commerce applications".

Instead, the approach proposed in this chapter suggests the adoption of an orthogonal information and trust and social relationship service, capable of supporting the overall e-commerce processes, especially the decision-making and disambiguation processes.

A similar concept is adopted by the on-going myOntology project (http://www.myontology.org), in which the ontologies are perceived as socially evolving descriptive artefacts of a domain of discourse, namely that of e-commerce. This project aims to create infrastructures capable of capturing the social interactions in developing/evolving the ontologies, and exploit such information "to improve the expressiveness and disambiguity of informal concept definitions in an ontology".

Therefore, this project is complementary to the approach we propose, in the sense that it is focused on capturing, populating and systematizing the repository with the ontology changes carried by the social groups. The approach suggested in this chapter will profit form this technology as a social information and Web-of-trust source of information. It is up to the SN-i to conveniently exploit such information in e-commerce scenarios.

COST AND BENEFITS

In an efficient agent-mediated electronic market it is necessary to have common standards: an

interaction protocol to achieve deals; a language for describing the messages' content; and ontology for describing the domain's knowledge.

The interaction protocol is usually announced at the marketplace and composed of a set of interaction phases. The language for describing the messages' content is usually provided by the communication platform used to develop the e-commerce application. Platforms for distributed agent's communication are also available, however, an ontology for describing the domain's knowledge needs to be incorporated and represent the best answer to the demand for intelligent systems that operate closer to the human conceptual level.

With regard to ontologies, many studies are being done but, currently, there is neither a standard ontology language nor a standard ontology knowledge representation. This lack of standardization, which hampers communication and collaboration between agents, is known as the interoperability problem. However, *ontologies per se* are not able to overcome information interoperability incompatibilities and it is hardly conceivable that a single ontology can be applied to all kinds of domains and applications. Information integration mechanisms are necessary in order to identify, represent and apply the diverse relationships between concepts and relations of two or more ontologies. Ontology Mapping is one of the most successful information integration paradigms used in the Internet and in particular in the Semantic Web. The use of a common ontology guarantees the consistency and the compatibility of the information present in the system.

In this work, the different actors involved in the marketplace are able to independently describe their universe of discourse, while the market is responsible for providing a technological framework that promotes the semantic integration between parties through the use of ontology mapping.

On the other hand, we propose an approach where the agents are free to accept or reject the mapping done by the marketplace. With this approach we can test several scenarios and explore a valid future agent-based platform for B2C. Additionally, through the integration information process carried out in the different stages, the searching space is enlarged, promoting the access to goods/services otherwise inaccessible.

Nevertheless, this solution has some drawbacks related to development and deployment. The development problems are specially related with the heterogeneity of the main technology requirements. The deployment issues are related with the configuration of the agent according to the user's constraints and strategic behaviours.

RISK ASSESSMENT

During this chapter, several assumptions were considered in order to support the proposed system. In particular, it is assumed that the social network paradigm will continue growing and that processing mechanisms will be provided. In particular it has been assumed that:

- The social participation in the Web will continue growing, namely tagging, social bookmarking and collaborative activism;
- The capabilities of the social network infrastructures will increase, providing and promoting services exploiting the information brought online by each individual;
- The relation (dependency) between Web-Of-Trust systems and Social Networks (based on FOAF or other representation mechanism) will tighten. This relation will be of fundamental importance in the effective development of the proposed infrastructure since it will provide the identification and trust infrastructure between communities and systems.
- The social network concept will evolve in order to encompass some explicit semantics, turning into semantic social networks. More than an enabler, such evolution would

facilitate the adoption of the proposed approach.

According to what has been stated, the proposed approach has several internal strengths and weaknesses.

Strengths:

- The CBB reference model is widely recognized as a correct interaction model in B2C.
- The proposed approach adopts concepts and behaviours not supported by traditional agent-based approaches.
- Permeation of ideas, domain modelling and content characterization between individuals and communities leads to socially supported decisions and disambiguation.

Weaknesses:

- Intrinsic semantic ambiguity of the proposed approach, which might lead to users having difficulty in accepting the decisions recommended by the system.
- Novelty of the suggested model, which might lead to poor results which are not fully geared and tweaked.

At the same time, the proposed approach profits and suffers from the technological, social and economic context.

Opportunities:

- Limitations of the user-driven portal-based and of the traditional agent-based B2C.
- Social network relationships, giving rise to communities of interests with some degree of mutual trust.
- SNA metrics and statistics, enabling the characterization of (implicit) communities, and hence promoting business activities according to target community.
- Social network engagement, promoting the cooperation between individuals, thus facilitating technological solutions, such as information integration and decision support.

Threats:

- Maturity of the Social Networks and related technologies.
- Lack of the supposed social network services.
- Lack of social network services accessibilities for software agents (e.g. API, Web Services).

FUTURE TRENDS

One of the largest problems this approach faces is related to the access and processing of social relationships. Despite the fact that the social networks and social network analysis are well established concepts, their implementation in the scope of the Internet started a couple of years only. As a consequence, some of the available services supporting online social communities (e.g. LinkedIn, MySpace, FaceBook) are now facing relevant problems, such as duplication of registers inside and across services, abandoned registers, ghost (virtual) registers. Instead, the evolution seems to follow the specialization trend (e.g. one service is used to create music relationships, another one is used for sports relationships) and respecting the age of user (i.e. one service is mostly used by young people, while another is used by middle age people). Additionally, organizations are now creating their own business-oriented social networks, either for employers or customers. Social networks are spreading on the Web, which will require an immense effort of integration. Our approach would profit from this characterization as it will be possible to better characterize the customer needs in respect to buyers and other customers.

We need to develop our approach and test several scenarios in order to obtain and explore new market information, which will be used to feed agent's knowledge-bases and the social networks repositories.

CONCLUSION

The meaningful interaction between distributed, heterogeneous computing entities (e.g. software agents), in order to be both syntactic and semantically compatible, needs to follow appropriate standards well understood by all participants. Some standards are being developed with regard to ontologies specially the Ontology Web Language (i.e. OWL) in the context of Word Wide Web Consortium.

Several problems involved in the overcoming of syntactic and semantic heterogeneity are difficult to be solved, at least nowadays. However, some efforts have been made in order to find possible ways to resolve parts of this complex problem.

A big challenge for communicating software agents is to resolve the problem of interoperability. Through the use of a common ontology it is possible to have a consistent and compatible communication. However, we maintain that each different actor involved in the marketplace must be able to independently describe their universe of discourse, while the market has the responsibility of providing a technological framework that promotes the semantic integration between parties through the use of ontology mapping. In addition, we think that the solution to overcome theses problems has to take into consideration the technological support already existent, namely a well-proven e-commerce platform, where agents with strategic behaviour represent consumers and suppliers.

This chapter has proposed the use of agents and Multi-Agents Technology as a platform for B2C. We propose an agent-based electronic market with an ontology-based information integration approach, exploiting the ontology mapping paradigm, by aligning consumer needs and the market capacities, in a semi-automatic mode, improved by the application and exploitation of the trust relationships captured by the social networks. Additionally we explore the repository of relationships provided by emergent social networks, which can carry out the establishment of more accurate trust relationships between businesses and customers, as well as providing a better alignment (mapping) between their models. Social networks form the basic infrastructure to encompass trust in many internet based activities, including B2C transactions and information integration.

REFERENCES

Aberer, K., Datta, A., & Hauswirth, M. (2005). A decentralized public key infrastructure for customer-to-customer e-commerce. *International Journal of Business Process Integration and Management*, 1, 26-33.

Dasgupta, P., & Das, R. (2000). Dynamic Service Pricing for Brokers in a Multi-Agent Economy. *Proceedings of the Third International Conference for Multi-Agent Systems* (ICMAS), pp. 375-76.

Fatima, S., Wooldridge, M., & Jennings, N. (2004). An agenda-based framework for multi-issue negotiation. *Artificial Intelligence*, 152(1), 1-45.

HarmoNet (2008). HarmoNET - the Harmonisation Network for the Exchange of Travel and Tourism Information. Retrieved January 11, 2008, from the World Wide Web: http://www.etourism-austria.at/harmonet.

Hepp, M. (2007). Possible Ontologies: How Reality Constrains the Development of Relevant

Ontologies. *IEEE Internet Computing*, 11(7), 96-102.

Huhns, M. N., & Singh, M. P. (1997). *Readings in Agents*. San Francisco, CA: Morgan Kaufmann Publishers.

Maedche, A., Motik, B., Silva, N., & Volz, R. (2002). MAFRA-A MApping FRAmework for Distributed Ontologies. *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management*, LNCS, 2473, 235-250.

Malucelli, A., Rocha, A., & Oliveira, E.. (2004). B2B Transactions enhanced with ontology-based services. *Proceeding of the 1st International Conference on E-business and Telecommunication Networks.* Setúbal, Portugal.

Obrst, L., Liu, H. & Wray, R. (2003). Ontologies for Corporate Web Applications. *AI Magazine*, 24(3), 49-62.

Osborne, M. J., & Rubinstein, A. (1994). *A Course in Game Theory.* Cambridge, MA: MIT Press.

Paolucci, M., Sycara, K., Nishimura, T., & Srinivasan, N. (2003). Toward a Semantic Web e-commerce. *Proceedings of the 6th International Conference on Business Information Systems.* Colorado Springs (CO), USA.

Pinto, H., Gómez-Pérez, A., & Martins, J. P. (1999). Some issues on ontology integration. *Proceedings of the Workshop on Ontology and Problem-Solving Methods: Lesson learned and Future Trends at IJCAI'99*, 18, 7.1-7.11.

Runyon, K., & Stewart, D. (1987). *Consumer Behavior* (3rd ed.). Merrill Publishing Company.

Silva, N., Maio, P., & Rocha J. (2005). An approach to ontology mapping negotiation. *Proceedings of the Third International Conference on Knowledge Capture Workshop on Integrating Ontologies.* Banff, Canada.

Sim, K. M., & Choi, C. Y. (2003). Agents that React to Changing Market Situations. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, 33(2), 188-201.

Siorpaes, K., & Hepp, M. (2007). myOntology: The Marriage of Ontology Engineering and Collective Intelligence. *Proceedings of the ESWC 2007 Workshop "Bridging the Gap between Semantic Web and Web 2.0"*. Innsbruck, Austria.

Ströbel, M. (2001). Communication Design for Electronic Negotiations on the Basis of XML Schema. *Proceedings of the Ten'th International Conference on World Wide Web.* Hong-Kong, pp. 9-20.

Studer, R., Benjamins, R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, 25(1), 161-197.

Tsvetovat, M., & Sycara, K. (2000). Customer Coalitions in the Electronic Marketplace. *Proceedings of the Fourth International Conference on Autonomous Agents*, pp. 263-264.

Van Damme, C., Hepp, M., & Siorpaes, K. (2007). FolksOntology: An Integrated Approach for Turning Folksonomies into Ontologies. *Proceedings of the ESWC 2007 Workshop "Bridging the Gap between Semantic Web and Web 2.0"*. Innsbruck, Austria.

Viamonte, M.J. (2004). *Mercados Electrónicos Baseados em Agentes – Uma Abordagem com Estratégias Dinâmicas e Orientada ao Conhecimento.* Doctoral dissertation, University os Trás-os-Montes e Alto Douro, Portugal.

Viamonte, M.J., Ramos, C., Rodrigues, F., & Cardoso, J.C. (2006). ISEM: A Multi-Agent Simulator For Testing Agent Market Strategies. *IEEE Transactions on Systems, Man and Cybernetics – Part C: Special Issue on Game-theoretic Analysis and Stochastic Simulation of Negotiation Agents*, 36(1), 107-113.

Yamamoto, J., & Sycara, K. (2001). A Stable and Efficient Buyer Coalition Formation Scheme for E-Marketplaces. *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 237-288.

This work was previously published in Semantic Web for Business: Cases and Applications, edited by R. García, pp. 150-169, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.6
Electronic Commerce Strategy in the UK Electricity Industry:
The Case of Electric Co and Dataflow Software

Duncan R. Shaw
University of Nottingham, UK

Christopher P. Holland
University of Manchester, UK

Peter Kawalek
University of Manchester, UK

Bob Snowdon
University of Manchester, UK

Brian Warboys
University of Manchester, UK

ABSTRACT

This paper investigates the collective use of a simple modeling technology by highly complex, heterogeneous and numerous groups of stakeholders who heavily depend upon it to mediate their interactions. We use economic theory, design theory, complex systems theory and business process modeling concepts to analyze deregulation and business to business interaction in the UK electricity industry, and the strategic business and IT response of Electric Co, a large electricity supply company. The relevance of this study comes from its investigation of a novel example of the shaping of a whole sector's e-business through regulatory law and thus we are concerned with enterprise and inter-enterprise systems not purely with ERP systems. We focus on model-based business interaction and its effect upon the business and consumer behaviors of a

whole country's electricity sector. This sector is a socio-technical system; so business processes and consumer behaviors are not only shaped by the regulator's legally enforced business to business process interaction model, but the opinions of businesses and the public also influence how the regulator updates its model. Thus business behaviors, consumer behaviors and the model interact to shape each other. By moving from intra to inter-organizational business processes we seek to demonstrate and explain the value of models in e-business where the complexity of interacting business systems involves many thousands of parameters. We show how developments in technical standards and business process management are related to inter-organizational interaction and coordination.

INTRODUCTION

Networks of businesses are complex systems that are joined by complex interrelations (Anderson, 1999). These interrelations take the form of business processes that join the elemental businesses. Business process modeling literature seeks to describe business processes from different perspectives, according to different objectives or goals. To do this, it uses modeling constructs such Curtis, Kellner, and Over's three constructs of agent, role and artifact to do so (Curtis, Kellner, & Over, 1992). This paper illustrates the use of regulatory power over modeling methodology sophistication. By moving from intra to inter-organizational business processes it demonstrates and explains the value of models in e-business where the complexity of interacting business systems involves many thousands of parameters (Scheer & Habermann, 2000). The importance of this study is that it moves away from the single firm Enterprise Resource Planning (ERP) system perspective. Here we focus upon enterprise and inter-enterprise systems and not purely upon ERP systems. Rather, this study focuses upon model-based business interaction and its effect upon the business and consumer behaviors of a whole country's electricity sector.

The relevance of this study comes from its investigation of a novel example of the shaping of a whole sector's e-business through regulatory law. It is also an example of a collective use of a simple modeling technology by highly complex, heterogeneous and numerous groups of stakeholders who heavily depend upon it to mediate their interactions. Most interestingly it is a socio-technical system; so not only are business processes and consumer behaviors shaped by this legally enforced model, but the opinions of business and the public also influence how the regulator updates the model. Thus business behaviors, consumer behaviors, and the model shape *each other.*

The business processes that embody business-to-business (B2B) interaction can be decomposed into sub-processes; and sub-processes into sub-sub-processes *ad infinitum* (Rescher, 2000). For example, the production process for a car can be decomposed into the main production processes of all the first tier suppliers to the car company, whose brand appears on the front of the car. These main production processes can be further decomposed into those of second tier suppliers and so on. Additionally, all the on site business processes of all these companies can be also be decomposed, seemingly, *ad infinitum* and so can the business processes that are the interactions between suppliers and customers. This has both theoretical and practical implications.

Theoretical Implications

The infinite decomposability of business processes is a possible reason for why there is no theoretical basis for business process model constructs. In the literature, these constructs use supporting theories from semiotics (Falkenberg et al., 1998; Stamper, 1987; Liu, 2000), Shannon's Communications

Theory (1948), Classification Theory (Parsons, 1996) and Ontology Theory (Wand & Weber, 2002; Green & Rosemann, 2000). Scheer's ARIS House Of Business Engineering, for example, is one of many modeling architectures (Scheer & Nüttgens, 2000); Hommes lists several hundred on his Web site (Hommes, 2005). ARIS' four levels provide four useful perspectives for abstracting salient real world properties. The first two levels, (1) process design and (2) process control, are process-centric (what happens) and its second two levels, (3) inter-application workflow coordination and (4) application used, are application-centric (by whom/what). Also, these perspectives are grouped into low and high specificity: levels (1) and (2) are general designs whilst levels (3) and (4) specify users (Scheer & Nüttgens, 2000). These are all useful perspectives for abstraction but because process modeling has no underlying theory other perspectives (e.g., regulatory, data quality, or even environmental) could equally be included. Such a theory may never be discovered because at some level processes are not pre-definable and their composition only becomes possible to know during enaction (ibid, p. 375). Complex systems are not modelable below a certain level of detail, or past a certain point in time, because emergence cannot be forecast (Checkland, 1981).

However, these are theoretical bases for the *communication* or *representation* of the process model constructs rather than for its *generation*. The model constructs in the business process modeling literature "merely list properties" (Lindland, Guttorm, & Solvberg, 1994), thus there is no way to judge the relative or absolute quality of different lists of model constructs. If processes are infinitely decomposable systems, as asserted by the Process Philosophy literature (Rescher, 2000), then an "atomic" process component does not exist and business process models can be compared to skyscrapers built upon mud. Their foundations do not rest upon bedrock but are deep enough to derive support from the mud itself. In modeling terms we can say that the model constructs only emulate properties of the subject in the domain of the modeler's concern. The lack of a theory also removes the possibility of a theory-derived heuristic.

Practical Implications

Secondly, businesses experience connection problems in practice. The composition of business processes from an immense number of elemental sub-processes enables many different configurations. This leads to many potential configuration mismatches between businesses (Scheer & Habermann, 2000). For example, car parts delivered to an assembly plant could arrive at the wrong time or in the wrong quantity, type, specification or batch size. Instead of "wrong" a more precise word is "unsynchronized". The delivered parts are not wrong for the deliverer but they hold properties that do not synchronize with the business processes that the receiver is enacting. The complex nature of B2B systems, and the business processes that join them, generates the potential for unsynchronized interdependencies. Synchronizing, or *managing interdependencies*, requires an act of *coordination* (Malone & Crowston, 1994) and one form of coordination uses standards (Langlois & Savage, 2001).

The objective of this study is to understand *how* and *why* the interdependencies (business processes) and interactions between businesses in a complex network can be managed, so as to reduce prices through increased competition while at the same time increasing interaction complexity, in other words, this is a *coordination* problem. Coordination involves business process composition and decomposition by all managers each time they make managerial choices, for example, choosing to incorporate one supplier's process output versus another's. We view business process coordination as the overall function of business process composition and decomposition. We attempt to do this by using theories from several different disciplines but which are *all* designed

to make sense of complex systems of one type or another. This includes design theory's and systems theory's intriguing *intersection* at the concept of "fit". We also develop a type-structural-volume flexibility model of inter-organizational interaction, which is build up gradually throughout the paper. It is grounded in the concepts of Requisite Variety, Bounded Rationality, Scale Economies, and Transaction Costs. It is illustrated using the case both at sector and firm level and its implications are set out for regulators and practitioners in our conclusion.

The contributions of our article apply both to academics and business practitioners, and are in its investigation of the use of business process standards to mitigate the complexities B2B interaction. In one such network, an electricity market, we illustrate how one company uses a business process standard at strategic management, business process and technical levels. This cross-disciplinary paper describes some of the problems of inter-organizational human interaction and how they have been solved by using a business process model.

RESEARCH METHOD

This paper illustrates a sector where very high consumer churn rates have forced very high volumes of business-to-business and business-to-consumer interaction. The paper analyzes how this produces data-exchange problems for electricity suppliers and how these problems are reduced by the regulator and then solved in detail by one particular electricity supplier. This solution is explained with a conceptual framework that is constructed using the theoretical concepts of *standards* from Economics, *fit* from Design Theory and Ashby's Law of Requisite Variety from Cybernetics. We apply this framework to the context of modeling the business processes that constitute the regulated interactions of the main actors in the UK electricity industry. Then we explore the practical implications of this by showing how the sector model is used by one electricity supplier to solve its own acquisition interconnection problems. Current literature does not adequately cover complex, high volume, politically sensitive, and high frequency business interactions across whole sectors.

We take an interpretive stance, because of the subjective nature of human interaction, and iterate around a hermeneutic circle, comparing sectoral *and* organizational perspectives so as to consider an interdependent whole joined by a standardized interaction model (Klein & Myers, 1999). We use a qualitative approach because our investigation is concerned with questions of "*how*" and "*why*" rather than of "how many". In seeking to answer "how" and "why"-type questions we follow Yin's recommendations (2003, pp 5) and use a case study approach: "how can a whole industry sector organizations be coordinated?" and "why do electricity prices reduce even when B2B interaction complexities seem to be greatly increased?", i.e., "how is it possible that standardization increases flexibility and why is this so?" A case approach is fitting because we are concerned with contemporary phenomena which we have no control over (ibid).

We are concerned with dynamic phenomena, so we have used several different data collection methods (Eisenhardt, 1989). Most data was obtained from interviews with members of "Dataflow Software's" board, marketing staff and design teams. Dataflow described the history of the use of modeling and modeling technology both in the sector and in helping their client "Electric Co". They also provided us with technical product information. Additionally, a full version of the regulator's industry business process model, the Master Registration Agreement (MRA) process diagrams were obtained from the MRA Service Company (MRASCo) Web site serviced by Gemserv Limited. Finally, other historical back-

ground data relating to government-led change in the sector was obtained from public Web sites as referenced.

STANDARDS FOR BUSINESS-TO-BUSINESS (B2B) INTERACTION

Kindleberger distinguishes between two types of standards: standards that create scale economies and standards that reduce transaction costs (Kindleberger, 1983; Langlois & Savage, 2001). Both types of standards act by reducing variety. Standards create scale economies by reducing the variety of *problem types* and *solution types* in a particular domain, which increases the volume of use of the remaining *problem types* and *solution types.* For example, consider a market with a size of 100 widgets per year where four different types of widgets are produced and consumed. If the market's customers change to a standard consisting of two widget types then the average volume of these two types could be expected be approximately double the volume of any one of the previous four solution types. This assumes an equal demand for all types and neglects reduction in demand from buyers who could only tolerate a discontinued widget type. The higher volumes permit economies of scale from longer production runs and reduced stock costs that are brought about by narrower product portfolios.

Transaction cost standards reduce the costs of a transaction between two people, or organizations, by aligning expectations and increasing the predictability of outcomes (Langlois & Savage, 2001). This acts to reduce the costs of the interaction by *limiting* the interaction, in other words, standards are a form of coordination (Schilling, 2000). Standards manage interdependencies by limiting their values to a subset of all possible values, for example, the use of standard shoe sizes increases the predictability of a fit when we try on a new pair of shoes in a shoe shop; so fewer shoes need to be tried on.

Standards can be technical (for example the Standard International units used in the physical sciences and engineering) or behavioral (such as taboos and traditions). Standards reduce "perceived" variety by improving communication precision, which leads to the increased alignment of expectations. By reducing the variety that has to be managed standards enable the *variety management* "capacity" of boundedly rational managers to be applied *elsewhere* (Newell, 1989). Standards are applicable whenever there are options or choices. They can also reduce variety when applied to the interdependencies between businesses and thus can be used to *synchronize* B2B processes. They reduce variety when applied to multiple occurrences of the same process, or even single instances, in a single business. For example, businesses may have standard delivery times; standard production processes may be reproduced many times on a production line; and a product specification is a design standard that is created only once because redesigned products are separate, unique and have different part numbers. Whether between businesses, between process instances or in the enaction of one process instance, all three examples contain options that have to be decided between.

Standards limit the requirement for *variety management* in a particular instance but this produces savings for either the producer (supply-side scale economies) or the consumer (demand-side transactions costs) only if the modular architecture of the chosen standard *fits* the architecture of the problem. Modules are components of a system that are arranged according to a set of rules, the *system architecture* (Schilling, 2000; Parnas, Clements, & Weiss, 1985). The system architecture is an abstraction of the full system based upon the concept of loose coupling between modules and tight coupling between sub-modular components within modules. Loose coupling is where the interdependencies between system elements are of low intensity (Weick, 1974; Orton & Weick, 1990). It is an example of Herbert Simon's

concept of near decomposability (Simon, 1969). In systems every element is connected to every other via interdependencies but the interdependencies between some elements are less intense than between others. For example, couplings are stronger within organizations than between organizations (depending upon the observation criteria). Thus organizations can be thought of as modules within socio-economic systems. The set of human rules or natural laws that govern the coupling tightness of elemental interdependencies is the *architecture of the system*. These same rules, or laws, also govern the *arrangement* of system modules.

In order for a standard to reduce variety, and so produce either scale economies or reduced transactions costs, there has to be *fit*. This concept of *fit* is a match between context and structure (Drazin & Van de Ven, 1985; Alexander, 1964). It is the fit between the architectures of the *problem system* and the *solution system* that join into a larger system when the solution is implemented. If the solution does not fully fit the problem then it is sub-optimal. The solution system may not address a subsystem in the problem system such as a product portfolio containing purely *desktop* PCs that fails to address a customer's *mobile* computing requirements. Another example would be a human designed standard for shoe sizes that must architecturally fit with the naturally evolved architecture of human foot growth and shoe use. Firstly, the architectural *cover* of the standard must include the cover of most of the human population. The subset of the population not covered by the standard would amend it or just not use it because it would not reduce variety for them. This concept of *cover* applies to both the *hierarchical levels* of the standard relative to the population of foot sizes and shoe use (the solution architecture) but also the *span* of the standard. A system can be decomposed conceptually into related subsystems and these subsystems can be further decomposed into related sub-subsystems and so on. The system, the subsystems and the sub-subsystems, and so forth, exist upon three different hierarchical levels. These are levels of recursion like national economies, businesses and departments; or departments, human bodies and cells. In order to reduce variety a standard must cover the appropriate hierarchical *levels* of the subject to be standardized (e.g., the feet and foot use). The *span* of a system relates to the sum of the properties of its subsystems on any one level (Simon, 1969). For a standard to fit it must be applicable across all the hierarchical levels and the full span of system properties that its users find *relevant*. These hierarchical concepts are similar to Allen, O'Neill, and Hoekstra's *grain* and *extent* (1984).

Secondly, the standard's classification system must *reflect* the modularity of the subject system. In this example the architectural structure of the standard's classes must follow the architectural structure of human feet. If the standard does not reflect the modularity of the subject system, in other words, if it includes classification boundaries in regions of tight coupling in the subject system, then variety would not be reduced and so choosing between classifications would not be made easier. For example, a standard with a classification system that specified sole size *independently* of upper shoe size would double the variety of choice for the buyer by decomposing a single class of "whole shoe size" into two sub classes. If human shoe use required similar uppers but with different soles then this added variety would be *requisite*, in other words, there would be enough solution variety to match the variety of the problem (Beer, 1979). But humans tend to change shoe uppers and shoe soles at the same time so in human shoe *use* upper variety is tightly coupled to sole variety and the problem variety does not require a standard that decouples shoe upper and shoe sole classifications. There may also be a lack of fit if the standard includes too *few* classification boundaries in regions of loose coupling in the subject system. For example, we tend to use shoes independently of clothes. Our use

of foot coverings and body coverings are loosely coupled, so clothes that incorporate foot coverings are rare. Some examples of tight coupling between foot coverings and body coverings are space suits, certain types of wet suits and anglers' waders. In these three examples the *purposes* of the foot coverings and body coverings are tightly coupled and thus so is the construction of the covering. The hierarchical level and the span granularity at which the standard's classification system best reflects the modular architecture of the subject system is a trade off between increased fit versus increased variety.

We also use fit to assess the compatibility between the industry process model and the regulator's objectives (i.e., does the market function? Is market efficiency improved?). There is an architectural fit because although the industry process model is merely an abstraction of all possible supplier-customer-regulator interactions it includes enough of their fully set of behaviors to allow each role to interact successfully.

CASE ANALYSIS OF ELECTRIC CO AND DATAFLOW SOFTWARE

The UK Electricity Market

In 1990, 12 regional electricity companies were formed to supply business and domestic customers in the UK. Each Regional Electricity Company (REC) controlled a geographically defined market. At the same time there were just three major electricity generating companies: BNFL; Powergen; and National Power. Powergen and National Power used fossil fuels to produce 80 percent of the electricity and BNFL produced the remaining 20 percent using nuclear energy technology. Now we have a situation where Credit Suisse First Boston can trade on the future prices of electricity in the electricity wholesale marketplace and a grocery firm, Sainsbury, can supply electricity to a domestic customer through cable owned by any of the distribution companies, such as Scottish Power. There is now competition at every stage of the supply chain in the electricity market and a wide variety of competitive strategies have emerged. These range from large organizations attempting to dominate a particular stage of the supply chain to niche operators that focus on a very specific customer set. A transition has occurred from a highly regulated set of regional monopolies to an open and competitive marketplace. For this to happen in practice, in tandem with de-regulation, individual companies have developed and used advanced information systems and common industry standards for information exchange. This information exchange has in turn enabled the free flow of information to support the rapid changes in industry structure and interactions between electricity suppliers and users that include large corporate companies, small and medium sized enterprises, and domestic users. The range of participants now includes traders, generators, distributors and suppliers.

Electric Co is one of the new breed of electricity companies that is competing in multiple geographic regions, and is supplying electricity to all types of customers from domestic to large industrial users. A major part of its success in competing to attract and retain new customers is its advanced use of information systems that support both its internal business processes and the exchange of market data with competitors. One of its key technology partners is Dataflow Software whose software forms an integral part of the IT infrastructure of the industry as well as the communication flows between Electric Co's internal systems and its economic partners, customers, competitors and the industry regulator.

Deregulation and New Trading Agreements

In March 2001 the UK government introduced New Electricity Trading Arrangements (NETA) in England and Wales (NAO, 2003). These arrange-

ments were designed to increase competition in the electricity wholesale market, to reduce the price of bulk electricity, and ultimately to reduce the prices that the end users would have to pay. Prior to this the arrangements for trading electricity, known as the Pool, were thought to be uncompetitive and open to manipulation. The Office of Electricity Regulation (OFFER) and the Office of Gas Supply (Ofgas) were combined in 1999 to form The Office of Gas and Electricity Markets (Ofgem). In October 1997 the Minister for Science, Energy and Industry asked OFFER to review the way that electricity was traded in England and Wales and suggest a number of improvements. The areas of the review included reducing the price charged to the user and improving choice, quality and security of supply for electricity users (Ofgem, 1997).

OFFER's review of the Pool pointed to problems of over-regulation that limited the effect of normal market forces such as competition in price setting together with supply-side and demand-side price influencing. The Pool, one of the first examples of a wholesale electricity market in the world, was limited by the complexity of its bidding and price setting mechanism, the inflexibility of its regulations and openness to manipulation of its payment rules (NAO, 2003). The main proposal from the review was that the trading of electricity should become much more market-based. OFFER recommended that the electricity market should operate more like other commodity markets, subject to special requirements for physically balancing supply and demand in order to maintain the security and quality of electricity supply. These special requirements came from the fact that electricity is a commodity that is very difficult and costly to store and whose supply cannot be varied instantaneously but whose demand can. So only balancing, rather than buffer stores, can be used to stop frequent small power cuts caused by high user demand volatility. In addition to this the government required that users should be shielded from short-term price volatility and the financial instruments that are used within other commodities markets can also help with this.

These recommendations led to the current trading arrangements known as NETA. With NETA the value chain for the industry is now divided up into two main marketplaces (see Figure 1). These are the wholesale marketplace and the supply marketplace. The wholesale marketplace is made up of the six main generators: Innogy, London Electricity, Powergen, Centrica, Scottish Power and Scottish & Southern Electricity. These generators are vertically integrated into the supply marketplace, in other words, they produce *and* sell electricity to end customers. The six generators have evolved from very different starting points. For example, Centrica was originally only a supplier of electricity and had no generation capabilities. Powergen started in the power generation field and has now moved into the supply marketplace. Other companies in the wholesale electricity marketplace are banks, which trade on the price and availability of electricity just as they do in other commodity markets. Many of the small electricity generators have particular niches, for example they may use green energy sources such as hydroelectric and wind power. The supply, or "retail", electricity marketplace is made up of the six large, vertically integrated generators, several very large supply-only companies and tens of smaller companies. These smaller companies are frequently associated with well-known brands, or extensive customer networks, from sectors such as media, banking and food retail.

One significant outcome of deregulation was market instability caused by artificially high retail electricity prices normalizing under more natural market conditions. This instability caused a significant drop in the wholesale price of electricity, which in turn led to financial problems for the two electricity generation companies, British Energy and TXU Energy. British Energy had to ask the UK government for financial assistance and TXU Energy sold its European subsidiary to Powergen. However, the UK's National Audit Office (NAO)

has reported that overall "NETA has facilitated lower wholesale prices", a decrease of "over 20% between the introduction of NETA in March 2001 and October 2002". In the same period the NAO reported that electricity prices for industrial and commercial customers had fallen 18 percent. Prices for domestic customers fell less than this but still in line with the reductions in suppliers' overall costs. Customers who switched suppliers secured an average price reduction of 17 percent, but greater price reductions for domestic customers have been prevented by increases in supplier's environmental costs and "the substantial costs of processing changes of supplier" (NAO, 2003).

The Challenge to Electric Co and Other Electricity Suppliers

From the viewpoint of the customer one of the biggest changes caused by NETA is to enable customers to change their electricity supplier whenever they wish to do so. This has created responsiveness to demand-side market pressures and has vastly increased the number of customers who switch suppliers, which has produced an associated increase in the economic costs of market change, for example, customer churn: losing existing customers and acquiring new ones. The UK's National Audit Office found that from May 1999 to June 2000, "6.5 million customers — one in four — had saved money by changing supplier, and customers were changing at the rate of 400,000 a month" (NAO, 2004). The right of electricity users to easily change their supplier and the resulting increase in customer churn has forced Electric Co to develop systems that can cope with large numbers of customer registrations and deregistrations.

The Office of Gas and Electricity Markets (Ofgem), the electricity regulator, specifies a series of service level agreements that electricity suppliers must meet in this new open marketplace. In this industry Ofgem specifies how every organization operates. This specification describes all business processes and their input and outputs; the exchange of information and the information's meaning; and the timescales of the exchange. The model of the whole industry's business processes is described in the Master Registration Agreement (MRA) using dataflow diagrams that are extremely complicated. Over 96 detailed diagrams describe more than 21 main processes and many sub-processes that contain main varied process steps.

The MRA process diagrams are managed and maintained by the MRA Service Company (MRASCo, 2003). The business processes are

Figure 1. Elements of the English and Welsh electricity industry value chain in 2003

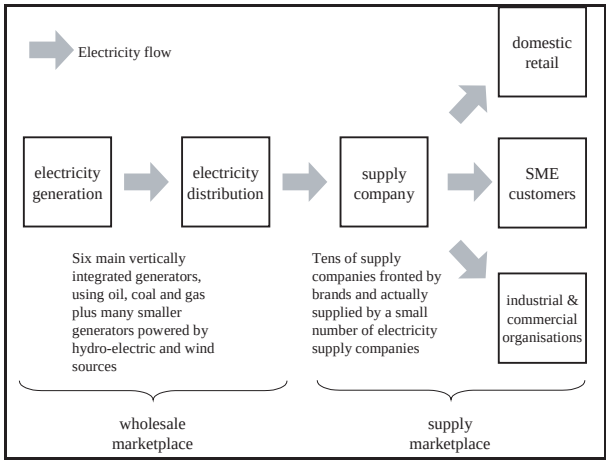
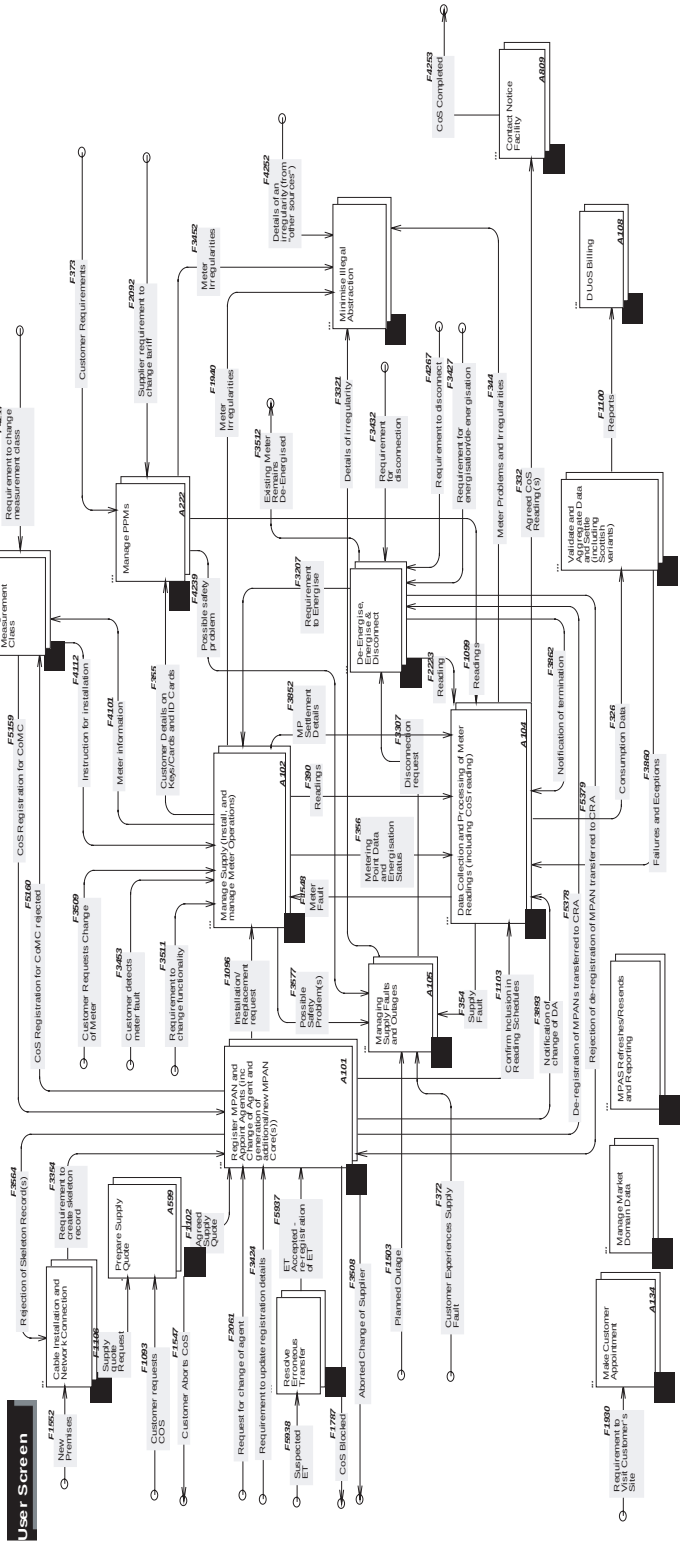


Figure 2. Overview diagram of the UK electricity industry key business processes (MRASCo, 2003)



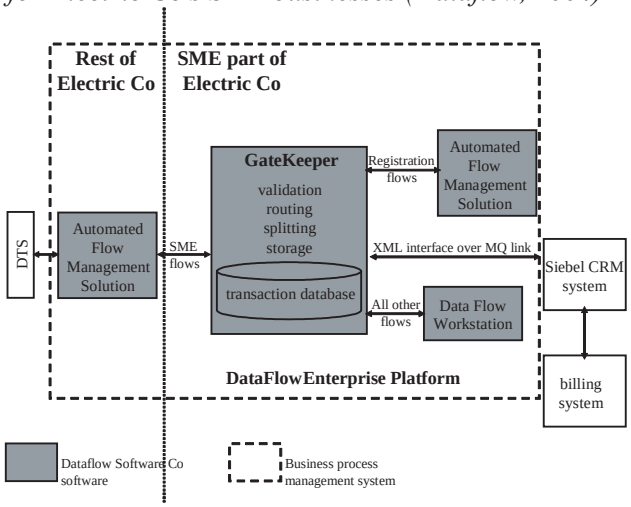
modeled in Enterprise Modeler by Enterprise Modeler Solutions Limited. An overview of the electricity industry's key business processes is shown in Figure 2. We use this to illustrate how complicated the MRASCo model is rather than for analysis purposes. Each change in supplier starts a formal deregistration process in one supplier and a formal registration process in another supplier. Each deregistration and registration process requires a complicated set of interrelated activities to take place in a pre-defined format, sequence and time. Although in concept this is straightforward, in practice the data structures are quite complicated and the high volumes of data exchanged exacerbate the information exchange and internal process problems. Every month 400,000 customers change suppliers; so suppliers have to process 9,600,000 deregistration and registrations per year. On a process level the cost of this from labor and potential errors is immense.

For example, each registration or deregistration incurs labor costs from taking customer telephone calls to maintaining the information systems that process each transaction. The potential for damage to the supply company's reputation from errors that reduce service levels is considerable because the registration process will fix a customer's first impressions and a customer that has already switched supplier will have few worries about switching again. Even deregistration is dangerous to the suppler because a customer may try to deregister in order to miss paying a bill. A supplier that allows a customer to transfer to another supplier with an outstanding debt may then find it harder to get the customer to pay.

Within this overall framework, Ofgem define what they call "Golden Threads". These are all the data flows and data processing operations that need to take place in order to enact a specific process, such as switching suppliers. The process diagram for switching suppliers is just one of the processes contained in MRA process diagrams and it includes interaction and service level specifications such as maximum times for the whole user disconnection-reconnection process.

Electric Co is greatly helped by software developed by Dataflow Software. For the parts of Electric Co acquisitions that service Small to Medium Sized Enterprises (SME) we can see (in Figure 3) that Dataflow Software's software manages the flows of data from and to the Data Transfer Service (DTS); between the SME parts of Electric Co; and between Electric Co's single Customer Relationship Management (CRM)

Figure 3. The data flows for Electric Co's SME businesses (Dataflow, 2004)



systems and multiple billing systems. The DTS is an Information and Communications Technology network, that is operated by Electralink, which enables business-to-business data communications between any electricity supplier that is connected to it (Electralink, 2004). Dataflow Software's GateKeeper software uses the DTS and provides data validation, routing, splitting and storage services together with a transaction history database.

Electric Co's Business and IT Strategy

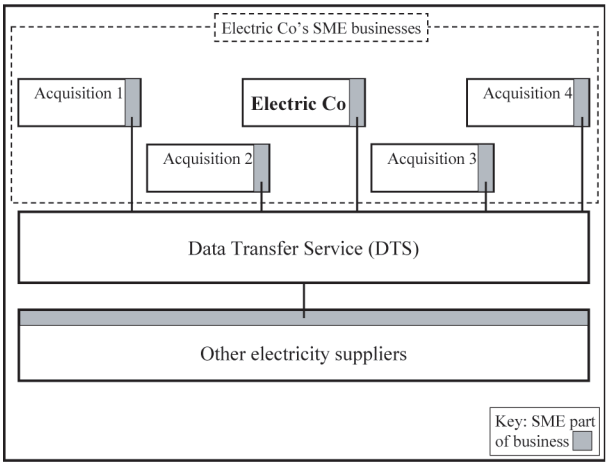
Electric Co has grown rapidly since 1995 through the acquisition of four large electricity suppliers composed of two original Regional Electricity Companies and two newer suppliers. Electric Co had the architectural legacy problem shown in Figure 4. Taking Electric Co's SME business as an example, we can see that each of the four acquisitions had their own information systems supporting their own SME businesses.

Upon acquisition Electric Co connected the separate SME systems via the Data Transfer Network. The DTN provided the interface for internal message passing in the same way that it was used as a medium for message passing to and from systems that were *external* to Electric Co. Whilst this did support business processes, such as the registration and deregistration processes examined above, the issues associated with increased external data flows concerning customer and supplier changes still needed to be resolved for the *internal* business processes.

Dataflow Software's IT solution strategy was to use software to integrate the SME components of the various legacy systems through a single communications and validation point. The actual software used was a product called Gatekeeper Data Exchange, a part of their Dataflow Enterprise Platform (Dataflow, 2004). Gatekeeper integrated valid dataflows with different back office applications, such as Credit Checking, CRM, Billing, and Registration, which run on different systems. Gatekeeper was able to do this because it was designed to use the governing industry data transfer rules to translate between information systems in different organizations and different systems in the same organization. This allowed Electric Co to consolidate its IT systems from a geographic architecture to a product/market architecture. This gives Electric Co the strategic benefits of being able to operate as a single entity, and has led to significant cost savings in IT expenditure.

It also improves the levels of customer service

Figure 4. Internal legacy system connection problem



because the *automation* of business processes and the associated data flows reduces errors and increases the speed of information flows to and from customers and competitors. This is a great benefit to Electric Co, given the huge volume of messaging associated with the customer churn for all electricity suppliers. After using the governing industry data rules to integrate the dataflows transmitted between the different industry organizations *and their software applications* Dataflow Software took the next logical step and created their Enterprise Automated Flow Management software. This software is able to automatically enact key processes because of its ability to process dataflows between the different back office applications used in the industry. The back-office software already obeys the governing industry data rules and Gatekeeper integrates valid dataflows. Dataflow Software were thus able to create a set of Automated Applications for customer registration and deregistration, and other volume intensive business processes, that uses the data held within back-office applications such as Credit Checking, CRM, Billing and Registration. Electric Co successfully integrates its SME business using Dataflow Software's solution and will extend this competitive advantage to its Domestic and Industrial & Commercial businesses.

ANALYSIS AND DISCUSSION

The Enterprise Automated Flow Management software is an example of a Business Process Management System (BPMS) (Aalst, Hofstede, & Weske, 2003; Basu & Kumar, 2002; Smith & Fingar, 2003; Sayal, Casati, Dayal, & Shan, 2002; Hollingsworth, 2004). BPMS, e.g., Intalio's N3 (Intalio, 2005), are distinct from ERP systems; i.e., business process management is only one component in some more sophisticated ERP systems. One example of this is SAP's Netweaver, which is *only part* of its larger ERP product offer (SAP, 2005). BPMS are significant developments of Workflow Management Systems (WFMS) with a more developed process design, diagnosis, and redesign functionality (Aalst et al., 2003). Hollingsworth describes BPMS as supporting a similar process design-execution-redesign cycle via an evolution of WFMS and its convergence with Enterprise Application Integration and World Wide Web technologies (2004). Dataflow's BPMS called Gatekeeper Data Exchange uses a model in the form of the governing industry data rules to translate between information systems in different organizations and different systems in the same organization. As such it is not itself an ERP system with a vast library of standard processes options but rather it manages other system components by manipulating the flows of data between intra and inter-system components, i.e., it *manages* business processes which are enacted by other entities. This differentiates it from ERP systems, which normally span a single enterprise rather than as in this case several quasi-integrated acquisitions.

This BPMS *includes* middleware capabilities but critically *also* contains process management capabilities. Bernstein (1996) defines middleware as "middleware services and/or frameworks" and a middleware service as "a general purpose service that sits between platforms and applications" (p. 89). The objectives of a business process management system, and of process modeling, are to facilitate human understanding and communication, to support process improvement, to support process management, to automate process guidance and to automate execution support (Curtis et al., 1992). Bernstein classes database management software as a type of middleware, but BPMS software also includes an internal process model for manual or automated process manipulation (Warboys, Kawalek, Robertson, & Greenwood, 1999). BPMS software is reliant on middleware for aspects of its execution but it obviously constitutes much *more* in terms of its functional scope and objectives. Smith et al point out that the reengineering of business pro-

cesses and natural business process changes create highly complex and unmanageable topologies in the "point-to-point" solutions of middleware companies (2002, p. 5). Only a process-centric approach, modeled on the *actual* business processes concerned, can remove the need for redundant and complex "layers" of higher and higher order interface and protocol combinations.

This is a further contribution of BPMS to business to business (B2B) interaction: (1) BPMS operate through the direct human or machine manipulation of process models and (2) help integrate human or machine actors that are joined by business processes. Historically, middleware has been used to facilitate B2B interaction but each interaction between different software applications running upon different hardware platforms requires a different interface and protocol combination, i.e., different middleware (Bernstein, 1996). These interface and protocol combinations have built up in layers in parallel to the hierarchies of interacting platforms, e.g., computer to computer, LAN to LAN, WAN to WAN, and so on . By focusing upon the business processes model of the actual B2B interaction BPMS bypass the layers of interface and protocol combinations, which differ by hierarchical level as well as by depending upon the interfacing party's hardware and software. This applies even more strongly to human to human interactions which are commonly less formal than technological interactions. Given an adequate model a BPMS can naturally execute across different systems and across different organizations but, unlike layers of interaction protocols, they are a direct abstraction of the user's focus of business concern rather than a 'fossilized' record of the negotiation between systems vendors.

The business process model at the heart of Dataflow Software's BPMS is a good *fit* with the reality of business in this sector because it is based upon the regulator's behavioral *standard* for electricity supply companies; the governing industry data rules in the MRA business process diagrams (see Figure 2). The regulator enforces this behavioral standard and so enforces a good fit between the behavior of the BPMS and its environment. It is this fit that enables the automation of many of Electric Co's business processes, which in turn enables it to deal effectively with the complexity of its environment. The usefulness of applying the concept of fit to businesses interactions is that it describes and explains the joining of two roles into a single system of "role components" while still preserving the role players' identities and supporting their business goals. Thus it is a *compatible* interaction. This can be contrasted with an incompatible interaction where either or both party's goals or even identities are not supported. Interaction compatibility is enforced in a one-to-many "top-down" sense as opposed to being "organically" built up in a series of one-to-one "bottom-up" negotiation processes. Interactions between the electricity suppliers in our case fit *by design* rather than by negotiation, i.e., forced interaction fit allows cost effective high volume data exchange between many agents

The regulator forces supplier-supplier interaction compatibility by defining a standard for supplier-supplier interaction, e.g., the standard includes the design of an electricity user's customer record and how and when this is exchanged between suppliers. Such records contain all the information that the new supplier needs in order to do business with the user, e.g., billing, usage, tariffs and customer requirements. The standard is represented using simple dataflow diagrams. This type of modeling notation does not provide a *natural architectural fit* with the complex marketplace of electricity companies and users. For example, it is *static* whereas the modeled subject is *dynamic*; the number of *hierarchical levels* is few in the model and infinite in the subject; and the *span* of this behavioral standard does not include organizations that support the suppliers. Dataflow Software's BPMS is a novel solution to a frequent post-merger problem because it makes used of predefined (regulated) interactions rules

and because it primarily manages inter-organizational interactions rather than intra-organizational interactions like most BPMS. We explore these three points next.

Firstly we can see that the *static* nature of the dataflow diagrams does not reflect the changing regulatory nature of the marketplace. The regulator attends to this by employing the MRA Service Company to update and publicize changes to the process standards as needed (MRASCo, 2003). In this case the UK government, via Ofgem the industry regulator, controls changes to the behavioral standard. This is in contrast to other industry standards whose definition is a *competitive* change process between organizations (Garud, Jain, & Kumaraswamy, 2002; Shapiro & Varian, 1999).

Secondly, the finite number of *hierarchical levels* in these dataflow diagrams does not reflect the infinite decomposability of a complex subject system such as the *actual* electricity marketplace of a national economy. This is true for all models, which are always abstractions of subjects. But in order for it to be a *useful* abstraction it has to abstract *salient* properties. In this case the salient properties are the descriptions of how the electricity suppliers must behave towards each other, the electricity users and the regulator from the perspective of data transferred. Modeling the non-salient properties of this system, such as the low level modeling of how they operate their businesses internally, is not required in this *interaction* standard. If we view the whole marketplace as our focal system then the standard achieves architectural fit, whilst using a only limited number of hierarchical levels by concentrating upon behavioral outcomes between high-level system elements (the market participants). The marketplace participants are natural modules so their *internal* complexities can be ignored with little reduction in the accuracy of the model.

Thirdly, the *span* of this behavioral standard does extend to specifications for organizations that support the electricity suppliers such as recruitment agencies, catering subcontractors, consultants, builders and lawyers. The behavior of the electricity suppliers is regulated but the behavior of the organizations that they depend upon to operate (in this constrained way) is not. The reason for this is again saliency. By concentrating on the behaviors of the electricity suppliers the standard leaves the management of these supporting organizations to them. These supporting organizations could be regulated, such as in a centrally planned economy, but the support organizations affect the marketplace only *through* the supply companies and as companies are natural modules this can be ignored just like internal complexities.

The Data Transfer Service (DTS) is another standard that Electric Co uses in the implementation of its acquisitive growth strategy. The DTS is an interconnectivity standard for data transfer between Electric Co and other electricity suppliers. It is an *information infrastructure* that supports data transfer between all electricity suppliers in this marketplace (Ciborra& Associates, 2001). For this reason Electric Co can also use it to connect between the legacy systems of its SME businesses. Thus the MRA is a behavioral, or business process, standard and the DTS is a interconnection technology standard.

CONCLUSION

The problem is that electricity supply companies are legally required to exchange vast quantities of customer data, which is costly. At the same time as they are under intense competitive pressures to reduce prices because of deregulation. The solution is a simplification of B2B interaction rules, which is embodied in the MRASCo model and which produces practical implications that Electric Co takes advantage of in the form of a Business Process Management System (BPMS) created by Dataflow Software.

Our theoretical argument is that human interaction on the scale of a whole industry, its consumers and its government can be standardized in *some* characteristics in order to promote flexibility in *others*, e.g., innovation-led consumer price reductions. Standardization is operationalized as a very large but unsophisticated business process model. The case phenomenon is market adjustment to deregulation and the theoretical basis of explanation is Ashby's Law of Requisite Variety applied to a network of boundedly rational actors. As a whole the electricity suppliers are *flexible* enough to reduce their prices from pre-deregulation levels, even though they are forced to interact with many different competitors using limited resources. This is because the complexity of supplier-supplier interactions is also limited by a regulatory standard and so flexibility is 'reserved' for price reductions rather than for interaction.

Models are architectures abstracted *from* a subject and standards are architectures projected *onto* a subject. The regulatory standard in our case fits even though its architecture is much simpler than that of the subject system because the subject system (the electricity marketplace participants) is legally *made* to fit it. This *reduces* requisite variety (interaction transaction costs) for all the electricity suppliers not just (Electric Co). The deregulation of the electricity marketplace can be viewed as environmental flexibility demands that can be categorized into three broad areas:

- Type flexibility — from the variety of different information types that is exchanged between suppliers;
- Structural flexibility — from the variety of other suppliers that suppliers have to connect with ;
- Volume flexibility — from the amount of information that is exchanged between suppliers.

The regulatory standard limits type flexibility requirements, because data types are standardized, and the DTS limits structural flexibility requirements because all suppliers connect via the same DTS. Even volume flexibility is limited because their service to the electricity users is standardized, so user demand aggregates, which smooths demand spikes. In this way the standard reduces requisite variety for all the electricity suppliers. For Electric Co, there are additional benefits from automating business processes whose variety is reduced to the extent that they can be fully *described* by a business process model that is based upon the business process standard itself.

We have shown that the case phenomena are linked to our conceptual framework as follows: (a) Standardization is used to explain the savings to electricity suppliers by following a common business process interaction model; and (b) The concept of system architecture from System Theory is used in conjunction with the concept of architectural fit from Design Theory to explain the prerequisites of such a model, i.e., dynamic, level and span fit. Theoretically we can see that this use of a whole industry interaction model works because the Requisite Variety of the sector's inter-organizational business processes is limited by legislation to those behaviors contained in the model. This allows Boundedly Rational electricity suppliers to concentrate their limited flexibility upon a finite subset of all competitor and consumer behaviors. Thus electricity suppliers are left with enough information processing capacity to reduce prices via innovation as well as deal with a very high consumer churn rate.

The most important benefit to Electric Co, from using these information systems, is achieving *structural* flexibility in compliance with industry regulations. These systems have also enabled Electric Co to grow its business and handle higher volumes of customer information; whether new customers have been won through competition with rivals or acquisition of competitors.

This paper draws upon multiple disciplines to make several theoretical and management contributions:

- it documents the changing environment of the UK electricity industry and the challenges facing an electricity supply company in terms of information management;
- it introduces, illustrates and explains a simple but widely applicable type-structural-volume flexibility model for examining business-to-business flexibility issues in e-commerce;
- it describes and explains an application of theoretical process modeling together with actual IT applications to manage business processes that cross between separate and newly merged organizations. This demonstrates an implementation of model-driven process reconfiguration that is supported by operationally integrating different software application packages.

We have used a parsimonious framework based on constructs from strategy, business process modeling and IT systems architecture, to explain an industry phenomenon of price reductions *at the same time* as increasing interaction complexity. We have also examined the implications of this in the form of Electric Co's strategic business and strategic IT responses to deregulation; and demonstrated the interrelationships between Electric Co's business strategy of growth by acquisition, its complex regulatory environment and its use of a business process management system to implement its operational systems. The high level strategy and the complexity of the operational requirements are clearly related to each other through the business process linkage.

The limitations of our approach are in its focus upon one industry in one country and in the enforcement of the business process model. Firstly this implies a limit upon generalization; a possible restriction upon the application of our model for examining business-to-business flexibility to this industry only. However, we can generalise to theory and abstract useful and insightful properties, as explained by Klein and Myers (1999). This is dependent upon the quality of our reasoning rather than purely on how representative our chosen case is. In fact our reasoning, linking interaction rules to theory from the domains of Economics, Cybernetics and Design, implies that our findings could be generalized to other B2B interactions (and, possibly, complexity management as a whole) because the theoretical framework itself is generalizable. So we suggest that this model-based approach could be applied to other B2B interaction problems and interaction flexibility problems. Secondly, the regulatory enforcement of the MRASCo model removes from the electricity suppliers any choice in how to interact other than to leave the industry. Thus we can make no general comment upon how to persuade many companies to follow a common interaction framework.

ACKNOWLEDGMENTS

We would like to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) in funding our xxx blinded xxx project. We would also like to thank the IJTHI reviewers and the reviewers and attendees at ECIS 2004 for helping us improve this article; and the staff and directors of "Dataflow Software" for access.

REFERENCES

Alexander, C. (1964). *Notes on the synthesis of form*. Harvard University Press.

Allen, T. F. H, O'Neill, R. V., & Hoekstra, T. W. (1984). *Interlevel relations in ecological research and management: Some working principles from*

hierarchy theory (General Technical Report RM-110, 11). USDA Forest Service. Rocky Mountain Forest and Range Experiment Station, Fort Collins, CO.

Anderson, P. (1999). Complexity theory and organization science. *Organization Science*, (10), 3.

Basu, A., & Kumar, A. (2002). Research commentary: Workflow systems. *Information Systems Research, 13*(1).

Beer, S. (1979). *The heart of enterprise*. John Wiley & Sons.

Bernstein, P. (1996). Middleware. *Communications of the ACM*, *39*(2).

Checkland, P. (1981). *Systems thinking, systems practice*. Chichester: John Wiley & Sons.

Ciborra, C. (2001). *From control to drift — The dynamics of corporate information infrastructures*. Oxford University Press.

Curtis, B., Kellner, M., & Over, J. (1992). Process modeling. *Communications of the ACM*, 35 (9).

Dataflow. (2004). Retrieved from http://www.Dataflow.com

Drazin, R., & Van de Ven, A. (1985). Alternative forms of fit in contingency theory. *Administrative Science Quarterly, 30*(4), 514-39.

Electralink. (n.d.). Retrieved September 27, 2004, from http://www.electralink.co.uk

Eisenhardt, K. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532-550.

Falkenberg, E., Hesse, W., Lindgreen, P., Nilsson, B., Oei, J., Rolland, C., et al (1998). *A framework of information system concepts* (The FRISCO Report) (electronic version). IFIP.

Garud, R., Jain, S., & Kumaraswamy, A. (2002). Institutional entrepreneurship in the sponsorship of common technological standards: The case of Sun Microsystems and Java. *Academy of Management Journal, 45*(1), 196-214.

Green, P. & Rosemann, M. (2000). Integrated process modeling: An ontological evaluation, *Information Systems, 25*(2), 73-87.

Hollingsworth, D. (2004). The workflow reference model: 10 years on. In L. Fischer (Ed.), *Workflow handbook, the workflow management coalition*. Retrieved August 12, 2004, from, http://64.70.135.73/information/handbook04.htm

Hommes, B. (2005). Retrieved February 21, 2005, from http://is.twi.tudelft.nl/~hommes/scr3tool.html

Intalio. (2005). *LexisNexis success story*. Retrieved February 12, 2005, from http://www.intalio.com/customers/successes/resources/documents/LexisNexis-Success-Story.pdf

Klein, H., & Myers, M.. (1999). A set of principles for conducting and evaluating interpretive field studies in information system. *MIS Quarterly, 23*(1), 67-93.

Kindleberger, C. (1983). Standards as public, collective and private goods. *Kyklos, 36*(3).

Langlois, R., & Savage, D. (2001). Standards, modularity, and innovation: the case of medical practice. In R. Garud, & P. Karnøe (Eds.), *Path dependence and path creation* (pp.149-168). Hillsdale: Lawrence Erlbaum. Retrieved August 13, 2004 from, http://www.sp.uconn.edu/~langlois/medical.html

Lindland, O., Guttorm, S., & Solvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software, 11*(2), 42-49.

Liu, K. (2000). *Semiotics in information systems engineering*. Cambridge University Press.

Malone, T. W., & Crowston, K. (1994). The interdisciplinary theory of coordination. *ACM Computing Surveys, 26*(1), 87-119.

MRASCo. (n.d.). *Master Registration Agreement (MRA) process diagrams.* Retrieved August 1, 2004 from, http://www.mrasco.com

NAO. (2003). *The new electricity trading arrangements in England and Wales* (Report by The National Audit Office for the House of Commons). London: The Stationery Office.

NAO. (2004). Retrieved July 5, 2005, from http://www.nao.gov.uk/publications/electricity1.htm

Newell, A. (1989). Putting it all together. In D. Klahr, & K. Kotovsky (Eds.), *Complex information processing: The impact of Herbert A. Simon.* Hillsdale, NJ: L.L. Erlbaum Associates.

Orton, J., & Weick, K. (1990). Loosely coupled systems: A reconceptualization. *Academy of Management Review, 15*(2), 203-223.

Ofgem. (1997). *What are the new electricity trading arrangements in England and Wales?* Office of Gas and Electricity Markets (Ofgem). Retrieved April 18, 2005 from, http://www.ofgem.gov.uk/elarch/retadocs/golive_explained.pdf

Parnas, D., Clements, P., & Weiss, D. (1985). The modular structure of complex systems. *IEEE Transactions on Software Engineering, SE11*(3), 259-266.

Parsons, J. (1996). An information model based on classification theory. *Management Science, 42*(10), 1437.

Rescher, N. (2000). *Process philosophy: A survey of basic issues.* University of Pittsburgh Press.

SAP. (2004). *Business process management with SAP netweaver.* Retrieved February 12, 2005 from http://www.sap.com/solutions/netweaver/pdf/BWP_NetWeaver_BPM.pdf

Sayal, M., Casati, F., Dayal U., & Shan, M-C. (2002). Integrating workflow management systems with business-to-business interaction standards. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)* (p. 287).

Scheer, A., & Habermann, F. (2000). Making ERP a success. *Communications of the ACM, 43*(4) 57-61.

Scheer, A., & Nüttgens, M. (2000). ARIS architecture and reference models for business process management. In W. van der Aalst, J. Desel, & A. Oberweis (Eds.), *Business process management: Models, techniques, and empirical studies. Lecture Notes in Computer Science, 1806*, 376.

Schilling, M. (2000). Towards a general modular systems theory and its application to inter-firm product modularity. *Academy of Management Review, 25*(2), 312-334.

Shannon, C. (1948, July and October). A mathematical theory of communication. *Bell System Technical Journal, 27,* 379-423 and 623-656.

Simon, H. (1969). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Smith, H., Neal, D., Ferrara, L., & Hayden, F. (2002). *The emergence of business process management.* Computer Sciences Corporation.

Shapiro, C., & Varian, H. R. (1999). The art of standard wars. *California Management Review, 42*(2) 8-32.

Smith, H., & Fingar, P. (2003) *Workflow is just a pi process.* Computer Sciences Corporation. Retrieved August 12, 2004, from http://www.bpm3.com/picalculus

Stamper, R. (1987). *Semantics.* In R. J. Bolland, & R. A. Hirschheim (Eds.), *Critical issues in information systems research.* John Wiley & Sons.

van der Aalst, W. M. P., ter Hofstede, A. H. M., & Weske, M. (2003). Business process management: A survey. In W. P. M. van der Aalst, A. H. M. ter Hofstede, & M. Weske (Eds.), *Proceedings of BPM 2003, Lecture Notes in Computer Science, 2678* (pp. 1-12).

Wand, Y., & Weber, R. (2002). Research commentary: Information systems and conceptual

modeling — A research agenda. *Information Systems Research, 13*(4), 363-376.

Warboys, B., Kawalek, P., Robertson, I., & Greenwood, M. (1999). *Business information systems: A process approach.* McGraw Hill.

Weick, K. (1974, March). Educational organizations as loosely coupled systems. *Administrative Science Quarterly, 21.*

This work was previously published in International Journal of Technology and Human Interaction, Vol. 2, Issue 3, edited by B. Stahl, pp. 38-60, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 4.7
IT and Software Industry in Vietnam

Yuko Iwasaki
Yokkaichi University, Japan

ABSTRACT

Vietnam has been advancing toward a market economy since 1986. Industrialization has progressed with a high rate of growth. One of the factors of the economic growth of Vietnam has been FDI. Japanese companies are among those that have a strong interest in Vietnam. Japanese companies are recently taking note of Vietnam's IT and software industries. Now, however, interest is increasing in offshoring as a means for developing in this sector.

INTRODUCTION

Vietnam has been advancing toward a market economy since 1986. After the Asian currency crisis in 1997, foreign direct investment (FDI) decreased and the growth rate slowed down, but since then, the country has followed a trend of recovery and industrialization has progressed with a high rate of growth.

One of the factors of the economic growth of Vietnam has been FDI. Japanese companies are among those that have a strong interest in Vietnam.

Up until now, the manufacturing industry has been at the center of Japanese companies' investment in Vietnam. However, Japanese companies are recently taking note of Vietnam's IT and software industries. Now, interest is increasing in offshoring as a means for developing in this sector.

In this chapter, the Vietnamese IT and software industries are surveyed, and the relations between Japan and Vietnam that have led to offshoring are considered.

RECENT ECONOMIC DEVELOPMENT

Since its adoption of the Doi Moi policy in 1986, Vietnam has promoted a market economy while also maintaining a socialist system. Due to the effort of the Doi Moi policy, FDI increased up to the first half of the 1990s, and high growth was recorded. However, after the Asian currency crisis of 1997, FDI decreased again, and the growth rate became slow. Since then, the present tendency has been maintained and the growth rate in 2005 showed a high 8.4% rate of growth with the progress of industrialization. Domestic demand was strong, and exports also remained strong.

While the manufacturing and service sectors expanded, the share of agriculture in the economy continued to decline (although it is still a major sector in terms of employment). Exports of electric goods continued to drive growth. Good performance was recorded.

FDI has been one of the factors of economic growth since 2000. Recently, with the increasing risks posed by the worsening business environment in China, Vietnam, with its low labor costs and stable political climate, is highly appreciated by investors anxious to spread their risks. Investment is increasing mainly in the manufacturing industry. Major investors are Japanese companies related to computers, electrical parts, automobile parts, motorbikes, and printers. Also, the Vietnamese government has improved businesses conditions for foreign firms.

Vietnam enjoys an especially high estimate among Japanese companies, whose direct investment in the country is increasing. In a medium-term survey conducted by the Japan Bank for International Cooperation (JBIC, 2005), Vietnam was ranked fourth in the world as a promising site for enterprise development (in approximately the next 3 years). For the longer term (approximately 10 years), it was ranked third by Japanese companies (Table 1).

Table 1. Promising countries and regions for overseas business operations over the medium and long terms (Source: JBIC, 2005)

Rank	Medium Term (next 3 years or so)	Long Term (next 10 years or so)
1	China	China
2	India	India
3	Thailand	**Vietnam**
4	**Vietnam**	Russia
5	U.S.A.	Thailand
6	Russia	U.S.A.
7	South Korea	Brazil
8	Indonesia	Indonesia
9	Brazil	South Korea
10	Taiwan	Malaysia

Among the reasons given for this were the cheap labor force, the country's attractiveness as a catcher for the risk, and its outstanding human resources. On the other hand, problems still to overcome include the insufficient infrastructure and the inadequacies and lack of transparency in the legal system.

As for its external economic relations, Vietnam participates in such organizations of regional integration as the Association of Southeast Asian Nations (ASEAN), the ASEAN Free Trade Area (AFTA), and the Asia Pacific Economic Cooperation (APEC). Its affiliation with the World Trade Organization (WTO) is expected to come about in 2006. Once Vietnam is affiliated with WTO, trade liberalization and the opening of the domestic market to foreign companies will be pursued.

In Vietnam, state-owned enterprises have so far borne an important role in achieving national prosperity through industrial development. However, with the current trend toward trade liberalization seen in economic globalization, the affiliation with WTO, and so on, competition with other countries has become very severe. For Vietnam, it is now vital to press on with the reform of state enterprises and to strengthen its competitiveness in the international market.

THE LEVEL OF IT IN VIETNAM

Vietnam is behind in its IT progress by any world standard. For example, in *The 2006 E-Readiness Rankings* (Economist Intelligence Unit), which investigated the diffusion of IT and the state of the fulfillment of businesses based on digital environments in countries around the world, Vietnam is placed at the very low level of 15th out of the 16 Asian nations surveyed, and 66th among 68 nations worldwide. (In 2005, it was ranked 15th out of 16 Asian nations and 61st among 65 nations worldwide.) This is a report that summarizes statistics on the diffusion of PCs (personal computers) and the Internet, the use of IT by firms and consumers, the digital response level of legal facilities and educational organizations, and so forth in various countries.

However, the diffusion of main and cellular phones and the Internet is now picking up in pace. Table 2 gives the diffusion of phones, the Internet, and PCs in Vietnam and other Asian countries. The data show the IT diffusion on an upward trend for Asian countries. In Vietnam, the diffusion of main phone lines was 18.8% and that of mobile phones was 9.54% (in 2005). The Internet user data of Vietnam and China was behind ASEAN4 countries in 2000, but are picking up. Vietnamese IT infrastructures have improved rapidly, especially for main phones and the Internet. However, most Internet users are concentrated in Hanoi and Ho Chi Minh City (approximately 70%). The Ministry of Posts and Telecomatics continues to lower charges for Internet and telecommunications services to expand the number of Internet users.

IT DEVELOPMENT IN ASIA

Asian countries have played an important and active role in IT development, especially in the PC industry. The newly industrialized economies (NIEs) and ASEAN countries showed rapid growth in IT manufacturing and trading since 1980s. These economies have made significant contributions to the global supply of IT products and components. As products matured, their production was relocated to low-cost areas.

During the last 10 years, China and India have joined in IT development. China has become a major production hub for IT products, and now the IT software industry has developed in both India and China. China's software exports have expanded rapidly. In 2005, China's software exports were valued at $3.6 billion, up from $7 million in 2001. Major export partners are Japan (approximately 60%) and the Unites States and Europe (20%).

Table 2. IT indicators in Vietnam and Asian countries (Source: International Telecommunication Union [ITU])

	Main Telephone Lines per 100 Inhabitants		**Cellular Mobile Subscribers per 100 Inhabitants**	
	2000	2005	2000	2005
Vietnam	3.23	18.81	1.00	9.54
China	11.18	26.63	0.35	29.90
India	3.20	4.51	6.58	8.16
Asia	9.73	15.77	6.63	23.22

	Internet Users per 100 Inhabitants		**PCs per 100 Inhabitants**	
	2000	2005	2000	2005
Vietnam	0.25	12.72	0.76	1.26
China	1.74	8.44	1.59	4.08
India	0.25	5.44	0.76	1.54
Asia	3.10	9.78	3.24	6.51

	Main Telephone Lines per 100 Inhabitants		**Cellular Mobile Subscribers per 100 Inhabitants**	
ASEAN4	2000	2005	2000	2005
Indonesia	3.23	5.73	1.78	21.06
Malaysia	19.92	16.79	22.01	75.17
Philippines	4.00	4.16	8.44	39.5
Thailand	9.10	10.95	4.97	42.98

	Internet Users per 100 Inhabitants		**PCs per 100 Inhabitants**	
ASEAN4	2000	2005	2000	2005
Indonesia	0.92	7.18	1.02	1.36
Malaysia	21.39	42.37	9.45	19.16
Philippines	2.01	5.32	1.93	4.46
Thailand	3.74	11.03	2.79	5.58

India is now a major IT software producer and exporter in the world. IT development in India has differed from that in the other Asian countries. Highly trained scientific and technical personnel from India have been migrating to the United States since the 1960s, and many of them were engaged in the development of computers and the communications industry. The high-level skills of India supplied qualified personnel to do jobs on site (in the United States) at first. After that, major IT producers set up subsidiary software centers in India (Braum, 2001).

As mentioned, India has highly skilled IT workers, and the business form of India is converting from that of onshore to offshore. While operating onshore means doing business in the customer's country (for example, in the United States or Europe), offshoring means producing software in India by Indian personnel. Lower prices and high-level skills have made the Indian software industry very competitive.

THE MARKET SCALE OF THE VIETNAMESE IT INDUSTRY

The market volume of the IT industry in Vietnam in 2005 was $828 million ($630 million in hardware and $190 million in software), an increase of 20.9% compared with the year before. In 1996, the market had only been worth $150 million, rising to $300 million in 2000 (Table 3).

IT-RELATED POLICY

The 2005 Plan for IT Use and Development was announced as a national policy in 2001. This plan proposed the following basic targets in order to bring the country's IT level up to the global standard by 2005:

- The diffusion of IT across the whole of Vietnam.
- 4 to 5% of the population as Internet users.
- An annual growth rate of 20 to 25% in the IT industry.
- The training of 50,000 IT specialists.

The putting in place of a telecom infrastructure was achieved ahead of the target. However, the training of competent personnel fell short of the target, not going above 20,000 employees. IT specialists are in short supply in Vietnam.

The announcement followed in 2005 of the Development Strategy for Information Communication Technology up to 2010 and of the Policy Plan up to 2020. This strategy and plan will serve as the foundation of Vietnam's future IT policy. In them, the most effective tool for national information development and modernization, and the importance of utilizing IT in economics and industry were brought out (e-citizenship, e-government, e-enterprises, e-transactions, and e-commerce). Moreover, high priority was given to the nurturing of an IT industry, especially the software and information-content industries, as a means of encouraging the creation and development of an information society. The policy also attaches importance to the laying of an IT infrastructure and the training of component specialists. The targets up to 2010 and 2015 for the promotion of IT use in economics and industry are as follows:

- By 2010, the IT industry will be promoted to the country's top industry, with a year-to-year growth rate of 20 to 25% and gross annual earnings of $6 to 7 billion.
- Through promotion of the IT infrastructure, the target for 2010 is to ensure 30 to 42 telephones per 100 people, 8 to 12 Internet service accounts (30% of them broadband), an Internet diffusion rate of 25 to 30%, and personal computer ownership of 10 per 100 people.

Table 3. Trends of the market scale (millions USD; Source: Ho Chi Minh City Computer Association, 2006)

Year	Total	Hardware	Software
2000	300	250	50
2001	340	250	60
2002	400	325	75
2003	515	410	105
2004	685	545	140
2005	828	630	178

- By 2015, the diffusion of fixed telephones will be lowered to 20 per 100 people, while that of cellular phones will be raised to 30.

To ensure that the targets for 2010 and 2015 are attained, a range of definite measures have been specified:

- Diffusion of IT knowledge.
- Promotion of practical uses for IT.
- Reinforcement of state's IT administration capacity.
- Ensuring of attractive conditions for foreign capital.
- Increasing of quantity and quality of IT labor.
- Strengthening of research and development system.
- Enhancement of legal environment in support of IT.
- Domestic and international cooperation.
- Integration with international IT market, especially for software and contents.

THE SOFTWARE INDUSTRY AND OFFSHORING

Within the industry, one of the fields that the government sees as very important is the software industry. The value of the software industry market in Vietnam in 2004 was $140 million. This marked a spectacular increase from $75million in 2002 and $105 million in 2003. 25% of the profits in the software industry are obtained from exports overseas. Vietnam started exports of software in 1997.

OFFSHORING

Behind the recent market expansion in Vietnam, one factor that can be mentioned is an increase in outsourcing (offshoring) for overseas products. Offshoring is the entrusting of part of a system development to a firm in another country (Figure 1). It has the advantage when using cheap and plentiful workforces.

Inspired by the example of India, which achieved rapid growth through offshoring from America and Europe, moves have been made by Vietnam to make use of its trained labor force to build up a software industry. Firms have also increased to about 600 by the end of 2005. Looking at them regionally, about 30% of these firms are located in Ho Chi Minh City or Hanoi. Corporate competition is intensifying with the increase in the number of companies. The following are some examples of the main software companies.

- FTP Software (Hanoi)
- TMA Software (Ho Chi Minh City
- Paragon Solutions Vietnam (PSV; Ho Chi Minh City)
- Silk Road (Ho Chi Minh City)
- Global Cybersoft (Ho Chi Minh City)

These companies have earned a name for reliable quality by acquiring CMM (capability maturity model) and CMMI (capability maturity model integration) under a quality-control stan-

Figure 1. An example of offshoring (Source: Ministry of Economy, Trade, and Industry [MEIT], 2006)

□ → Proposal	□ → System planning	□ → System design	□ → System manufacture	□ Operations and maintenance
Firm placing original order	Firm placing original order	Partly entrusted to overseas firm	Partly entrusted to overseas firm	Firm placing original order

dard code that objectively shows their capacity in developing software. Vietnam enjoys the following advantages in its software industry. First, its labor costs are cheap (about 60 to 70% of the cost in China), and the proportion of young people in the population is high. Second, the will to study is high and levels are excellent in the scientific and math fields. The potential for competitiveness in the software industry is ample.

However, there are few software companies of a scale exceeding 100 employees, and a lot of small and medium-sized enterprises exist. This poses the problem of a shortage of outstanding human resources in the industry as a whole. The training of competent personnel will be a challenge for the future. The government is setting preferential measures of taxation for the promotion of the software industry.

The market value of offshoring in 2005 was $70 million. In 2002 the market had only been worth $20 million.

OFFSHORING BY JAPANESE COMPANIES

Offshoring by Japanese companies can be seen to have increased rapidly in the past several years. The scale of Japanese offshoring, which was ¥20 billion in 2002, rose to ¥48.9 billion and then ¥52.7 billion in 2004 (Table 4). The background cause for this increase is that while the scale of software development is growing, the shortage of trained labor in Japan is serious, and in order to cope with shortening development times and the need for cost cutting, firms are therefore seeking a cheep and abundant labor force overseas.

With their cheap and plentiful workforces, India and China both make attractive offshoring sites, but for firms in Japan, China is the country most usually chosen on account of the geographical nearness and the fact there is a high level of aptitude for mastering Japanese.

Table 4. The scale of offshoring by Japanese companies (millions of Yen; Source: Japan Information Technology Services Industry Association [JISA], 2005)

	2004	Share (%)	Compared with 2003 (%)
China	33,241	63.1	+26.5
U.S.A.	5,147	9.8	+3.2
India	4,255	8.1	-32.6
Australia	3,133	5.9	+19.3
U.K.	2,126	4.0	+16.4
Philippines	2,117	4.0	-15.1
South Korea	1,415	2.7	-24.4
France	548	1.0	-34.3
Canada	262	0.5	-57.5
Vietnam	216	0.4	+620.0
Others	237	0.4	-78.1
Total	52,697	100.0	+28.1

The technical levels of companies in China and India are improving with the increase in the offshoring from Japan and the West. Points particularly attended to by Japanese companies when choosing an offshoring partner are the quality and quantity of technical workers available, whether Japanese can be understood, and whether the dealing prices are right. Although the volume of business with Vietnam is still small, Japanese companies are considering it as a new offshoring site. The scale of offshoring in Vietnam in 2004 showed a rapid seven-fold increase compared with the year before. Behind this growth lies the fact that the Vietnamese software industry recognizes Japan as an important market and has expanded its business in that direction. In 2005, the software association of Vietnam (Vietnam Software Association, VINASA) announced its 5-year plan for the development of the software industry (2006 to 2010). This plan sets a target of $1 billion in annual sales for software by 2010, with $400 million heading for Japan. Japanese companies are keenly observing Vietnam as a country where they can rely on finding excellently skilled workers at low cost.

As we have seen, Vietnam's purpose is changing to a market economy and introducing foreign capital to promote the IT industry with the aim of rapidly catching up with its nearby neighbors. With affiliation with WTO scheduled for 2006, this trend toward open markets and economic reform will progress further. Furthermore, there will also no doubt be improvement in the investment climate for foreign companies. For relations between Japan and Vietnam with regard to IT, while it is important to continue with the economic relations built up in the past through official development assistance (ODA) and FDI, it is also especially vital that the Japanese government and private firms should cooperate in the training of Vietnamese personnel.

CONCLUSION

For the ordering company, offshoring has merits, such as a reduction in development costs and the shortening of development times. On the other hand, for the country receiving orders, the government and private firms put more importance on policies for the fostering of information technology industries, both for the sake of economic growth and secure employment, and also for the furtherance of the information society. The proof of the effectiveness of this approach has already been seen in India.

In offshoring activities of Japanese companies, the values recognized with regard to China and India are well-established, and the time is now ripe to move on to the next stage. As a new location for offshoring, firms are taking close note of Vietnam, although many challenges exist in the present development of Vietnamese enterprises. One of the problems is a shortage of outstanding human resources in the industry as a whole. The training of competent personnel will be a challenge for the future. This will create a pool of highly skilled personnel that will be a foundation for the industry.

For Japan and Vietnam, which is still a developing country, it is important on both sides to work for an expansion of business by building a partnership for such aims as the training of IT personnel.

REFERENCES

Asia Development Bank (ADB). (2000). *Asian development outlook 2000 update*. Retrieved August 1, 2006, from http://www.adb.org/

Asia Development Bank (ADB). (2006a). *Asian development outlook 2006*. Retrieved August 1, 2006, from http://www.adb.org/

Asia Development Bank (ADB). (2006b). *Asian development outlook 2006 update*. Retrieved August 31, 2006, from http://www.adb.org/

Asia Development Bank (ADB). (2006c). *Key indicators 2006*. Retrieved August 1, 2006, from http://www.adb.org/

Braum, P. (2001). *Information and communication technology in developing countries of Asia*. Retrieved August 1, 2006, from http://www.adb.org/

Center of the International Cooperation for Computerization (CICC). (2005). *Asia computerization report (Vietnam).* Tokyo.

Center of the International Cooperation for Computerization (CICC). (2006). *Asia computerization report (Vietnam).* Tokyo.

Erran, C., & Paul, T. (2005). *Offshoring information technology*. Cambridge: Cambridge University Press.

Ho Chi Minh City Computer Association. (2006). *Vietnam ICT outlook 2006*. Retrieved August 1, 2006, from http://www.hca,org.vn/

Japan Bank for International Cooperation (JBIC). (2005). Survey report on overseas business operations by Japanese manufacturing companies. *JBICI Review, 13.* Retrieved August 1, 2006, from http://www.jbic.go.jp/english/research/report/review/

Japan Information Technology Services Industry Association (JISA). (2005). *Survey of overseas dealings in the computer software field 2005.* Retrieved August 1, 2006, from http://www.jisa.or.jp/

Ministry of Economy, Trade, and Industry (MEIT). (2006). *2006 white paper on international economy and trade*. Retrieved August 31, 2006, from http://www.meti.go.jp/

Ministry of Internal Affairs and Telecommunication. (2006). *2006 White paper on information and communications in Japan*. Retrieved August 31, 2006, from http://www.soumu.go.jp/menu_05/hakusyo/index.html

This work was previously published in Information Technology and Economic Development, edited by Y. Kurihara, S. Takaya, H. Harui, & H. Kamae, pp. 155-163, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 4.8
Utilizing Semantic Web and Software Agents in a Travel Support System

Maria Ganzha
EUH-E and IBS PAN, Poland

Maciej Gawinecki
IBS PAN, Poland

Marcin Paprzycki
SWPS and IBS PAN, Poland

Rafał Gąsiorowski
Warsaw University of Technology, Poland

Szymon Pisarek
Warsaw University of Technology, Poland

Wawrzyniec Hyska
Warsaw University of Technology, Poland

ABSTRACT

The use of Semantic Web technologies in e-business is hampered by the lack of large, publicly-available sources of semantically-demarcated data. In this chapter, we present a number of intermediate steps on the road toward the Semantic Web. Specifically, we discuss how Semantic Web technologies can be adapted as the centerpiece of an agent-based travel support system. First, we present a complete description of the system under development. Second, we introduce ontologies developed for, and utilized in, our system. Finally, we discuss and illustrate through examples how ontologically demarcated data collected in our system is personalized for individual users.

In particular, we show how the proposed ontologies can be used to create, manage, and deploy functional user profiles.

INTRODUCTION

Let us consider a business traveler who is about to leave Tulsa, Oklahoma for San Diego, California. Let us say that she went there many times in the past, but this trip is rather unexpected and she does not have time to arrange travel details. She just got a ticket from her boss' secretary and has 45 minutes to pack and catch a taxi to leave for the airport. Obviously, she could make all local arrangements after arrival, but this could mean that her personal preferences could not be observed and also that she would have to spend time at the airport in a rather unpleasant area where the courtesy phones are located or spend a long time talking on the cell phone (and listen to call-waiting music) to find a place to stay, and so forth. Yes, one could assume that she could ask her secretary to make arrangements, but this would assume that she does have a secretary (which is now a rarity in the cost-cutting corporate world) and that her secretary knows her personal preferences well.

Let us now consider another scenario. Here, a father is planning a family vacation. He is not sure where they would like to go, so he spends countless hours on the Web, going over zillions of pages, out of which only few match his preferences. Let us note here, that while he will simply skip pages about the beauty of Ozark Mountains—as his family does not like mountains, but he will "have to" go over a number of pages describing beach resorts. While doing this he is going to find out that many possible locations are too expensive, while others do not have kitchenettes that they like to have—as their daughter has special dietary requirements, and they prefer to cook most of their vacation meals themselves.

What do we learn from these two scenarios? In the first case, we have a traveler who, because of her unexpected travel, cannot engage in e-business as she does not have enough time to do it, while she could definitely utilize it. Yes, when in the near future airplanes will have Internet access, she will possibly be able to make the proper arrangements while traveling, but this is likely going to be an expensive proposition. Furthermore, the situation when a traveler is spending time on the plane to make travel arrangements is extremely similar to the second scenario, where the user is confronted with copious volumes of data within which he has to find few pertinent gems.

What is needed in both cases is the creation of a travel support system that would work as follows. In the first case, it would know personal preferences of the traveler and on their basis, while she is flying and preparing for the unexpected business meeting, would arrange accommodations in one of her preferred hotels, make a dinner reservation in one of her favorite restaurants, and negotiate a "special appetizer promotion" (knowing that she loves the shrimp cocktail that is offered there). Upon her arrival in San Diego, results would be displayed on her personal digital assistant (PDA) (or a smart cell phone) and she could go directly to the taxi or to her preferred car rental company. In the second case, the travel support system would act as an interactive advisor—mimicking the work of a travel agent—and would help select a travel destination by removing from considerations locations and accommodations that do not fit the user profile and personalizing content delivery further—by prioritizing information to be displayed and delivering one that would be predicted to be most pertinent first. Both these scenarios would represent an ideal way in which e-business should be conducted.

The aim of this chapter is to propose a system that, when mature, should be able to support the needs of travelers in exactly the previously described way. We will also argue that, and illustrate how, Semantic Web technologies combined with

software agents should be used in the proposed system. We proceed as follows. In the next section we briefly discuss the current state of the art in agent systems, Semantic Web, and agent-based travel support systems. We follow with a description of the proposed system illustrated by unified modeling language (UML) diagrams of its most important functionalities. We then discuss how to work with ontologically demarcated data in the world where such resources are practically nonexistent. Finally, we show how resource description framework (RDF) demarcated data is to be used to support personal information delivery. We conclude with a description of the current state of implementation and plans for further development of the system.

BACKGROUND

There are two main themes that permeate the scenarios and the proposed solution presented previously. These are: *information overload* and need for *content personalization*. One of the seminal papers that addresses exactly these two problems was published by Maes (1994). There she suggested that it will be intelligent software agents that will solve the problem of information overload. In a way it can be claimed that it is that paper that grounded in computer science the notion of a *personal* software *agent* that acts on behalf of its user and autonomously works to deliver desired personalized services. This notion is particularly well matching with travel support, where for years human travel agents played exactly the role that *personal agents* (PAs) are expected to mimic. Unfortunately, as it can be seen, the notion of intelligent personal agent, even though extremely appealing, does not seem to materialize (while its originator has moved away from agent research into a more appealing area of ambient computing).

What can be the reason for this lack of development of intelligent personal agents? One of them seems to be the truly overwhelming amount of available information that is stored mostly in a human consumable form (demarcated using hypertext markup language (HTML) to make it look "appealing" to the viewer). Even a more recent move toward the extensible markup language (XML) as the demarcation language will not solve this problem as XML is not expressive enough. However, a possible solution to this problem has been suggested, in the form of semantic demarcation of resources or, more generally, the Semantic Web (Berners-Lee, Hendler, & Lassila, 2001; Fensel 2001). Here it is claimed that when properly applied, demarcation languages like RDF (Manola & Miller, 2005), Web ontology language (OWL) (McGuinness & Van Harmelen, 2005) or Darpa agent markup language (DAML) (DAML, 2005) will turn human-enjoyable Internet pages into machine-consumable data repositories. While there are those who question the validity of optimistic claims associated with the Semantic Web (M. Orłowska, personal communication, April 2005; A. Zaslavsky, personal communcation, August 2004) and see in it only as a new incarnation of an old problem of unification of information stored in heterogeneous databases—a problem that still remains without general solution—we are not interested in this discussion. For the purpose of this chapter we assume that the Semantic Web can deliver on its promises and focus on how to apply it in our context.

In our work we follow two additional sources of inspiration. First, it has been convincingly argued that the Semantic Web and software agents are highly interdependent and should work very well together to deliver services needed by the user (Hendler, 1999, 2001). Second, we follow the positive program put forward in the highly critical work of Nwana and Ndumu (1999). In this context we see two ways of proceeding for those interested in agent systems (and the Semantic Web). One can wait for all the necessary tools and technologies to be ready to start developing and implementing agent systems (utilize ontological

demarcation of resources), or one can start to do it now (using available, however imperfect, technologies and tools)—among others, to help develop a new generation of improved tools and technologies. In our work we follow Nwana and Ndumu in believing that the latter approach is the right one. Therefore, we *do not* engage in the discussion *if* concept of a software agent is anything more but a new name for old ideas; *if* agents should be used in a travel support system; *if* agent mobility is or is not important, *if* JADE (2005), Jena (2005), and Raccoon (2005) are the best technologies to be used, and so forth.. Our goal is to use what we consider top-of-the-line technologies and approaches to develop and implement a complete skeleton of an agent-based travel support system that will utilize semantically demarcated data as its centerpiece.

Here an additional methodological comment is in order. As it was discussed in Gilbert et al. (2004); Harrington et al. (2003); and Wright, Gordon, Paprzycki, Williams, and Harrington (2003) there exists two distinct ways of managing information in an *infomediary* (Galant, Jakubczye, & Paprzycki, 2002) system like the one discussed here (with possible intermediate solutions). Information can be *indexed*—where only references to the actual information available in repositories residing outside of "the system" are stored. Or, information can be *gathered*—where actual content is brought to the central repository. In the original design of the travel support system (Angryk, Galant, Gordon, & Paprzycki, 2002; Gilbert et al., 2004; Harrington et al., 2003; Wright et al., 2003) we planned to follow the indexing path, which is more philosophically aligned with the main ideas behind the Semantic Web. It can be said metaphorically, that in the Semantic Web *everything is a resource* that is located somewhere within the Web and can be found through a generalized resource locator. In this case indexing simply links together resources of interest. Unfortunately, the current state of the Semantic Web is such that there are practically no resources that systems like ours could use. To be able to develop and implement a working system "now" we have decided to gather information. More precisely, in the central repository we will store sets of RDF triples (*tokens*) that will represent travel objects (instances of ontologies). We will also develop an agent-based data collection system that will transform Web-available information into such tokens stored in the system.

Obviously, our work is not the only one in the field of applying agents and ontologies to travel support, however, while we follow many predecessors, we have noticed that most of them have ended on a road leading nowhere. In our survey conducted in 2001 we have found a number of Web sites of agent-based travel support system projects that never made it beyond the initial stages of conceptualization (for more details see Paprzycki, Angryk, et al., 2001; Paprzycki, Kalczyński, Fiedorowicz, Abramowicz, & Cobb, 2001 and references presented there). The situation did not change much since. A typical example of the state of the art in the area is the European Union (EU) funded, CRUMPET project. During its funded existence (between approximately 1999 and 2003) it resulted in a number of publications and apparent demonstrations, but currently its original Web site is gone and it is really difficult to assess which of its promises have been truly delivered on.

Summarizing, there exists a large number of sources of inspiration for our work, but we proceed with development of a system that constitutes a rather unique combination of agents and the Semantic Web.

System Description

Before we proceed describing the system let us stress that what we describe in this chapter is the core of a much larger system that is in various stages of development. In selecting the material to be presented we have decided first, to focus on the parts under development that are finished or almost

finished. This means that a number of interesting agents that are to exist in the system in the future and that were proposed and discussed in Angryk et al. (2002); Galant, Gordon, and Paprzycki (2002b); and Gordon and Paprzycki (2005) will be omitted. Furthermore, we concentrate our attention on these parts of the system that are most pertinent to the subject area of this book (Semantic Web and e-business) while practically omitting issues like, for instance, agent-world communication (addressed in Galant, Gordon, & Paprzycki, 2002a; Kaczmarek, Gordon, Paprzycki, & Gawinecki, 2005) and others.

In Figures 1 and 2 we present two distinct top level views on the system. The first one depicts basic "interactions" occurring in the system as well as its main subsystems. It also clearly places the repository of semantically demarcated data in the center of the system. More precisely, starting from right to left, we can see that *content* has been divided into (a) *verified content providers* (*VCP*) that represent sources of trusted content that are consistently available and format of which is changing rarely and not "without a notice" and (b) *other sources* that represents all of the remaining available content. Interested readers can find more information about this distinction in Angryk et al. (2002) and Gordon and Paprzycki (2005).

While the dream of the Semantic Web is a beautiful one indeed, currently (outside of a multitude of academic research projects) it is almost impossible to find within the Web large sources of clean explicitly ontologically demarcated content (in particular, travel related content). This being the case, it is extremely difficult to find actual data that can be used (e.g., for testing purposes) in a system like the one we are developing. Obviously, we could use some of the existing text processing techniques to classify pages as relevant to various travel topics, but this is not what we attempt to achieve here. Therefore, we will, for the time being, omit the area denoted as *other sources* that contains mostly weakly structured and highly volatile data (see also Nwana & Ndumu, 1999, for an interesting discussion of perils of dealing with dynamically changing data sources). This area will become a source of useful information when the ideas of the Semantic Web and ontological content demarcation become widespread.

Since we assume that VCPs carry content that is structured and rarely changes its format (e.g., the Web site of Hilton hotels), it is possible

Figure 1. Top level view of the system

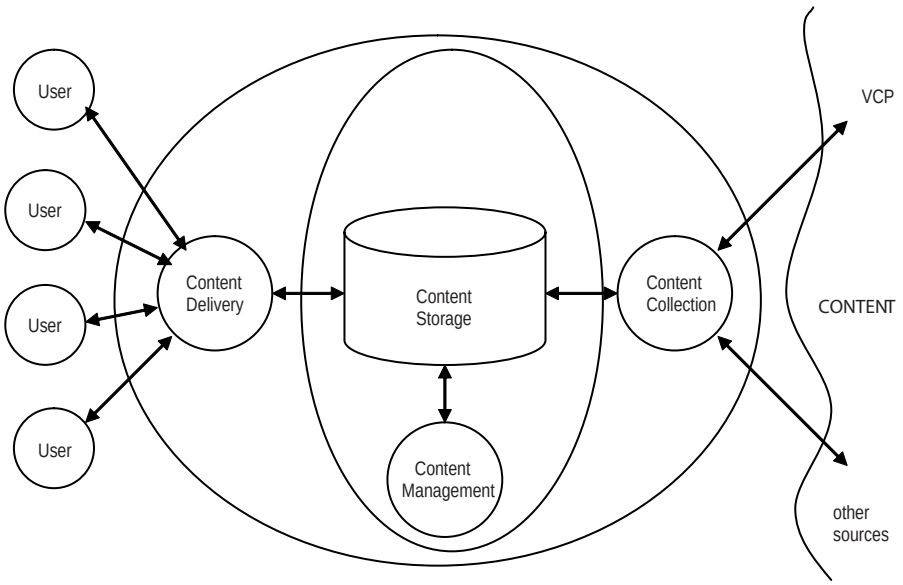
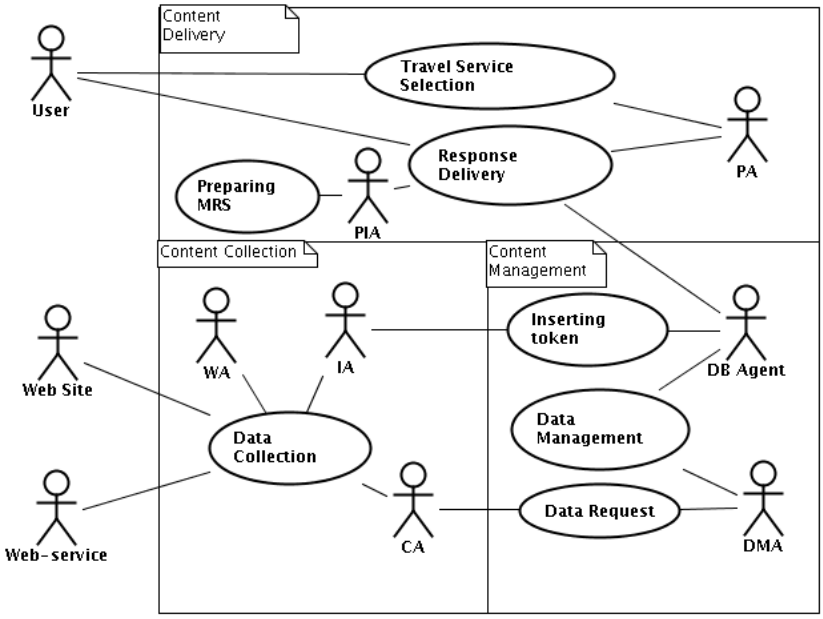


Figure 2. Top level use case diagram



to extract from them information that can be transformed into a form that is to be stored in our system. More precisely, in our system, we store information about travel objects in the form of instances of ontologies, persisted in a Jena (2005) repository. To be able to do this, in the *content collection subsystem* we use *wrapper agents* (*WA*) designed to interface with specific Web sites and collect information available there (see also Figure 2). Note that currently we have no choice but to create each of the WAs manually. However, in the future, as semantic demarcation becomes standard, the only operation required to adjust our system will be to replace our current "static WAs" with "ontological WAs." This is one of the important strengths of agent-based system design, pointed to in Jennings, 2001 and Wooldridge, 2002.

As mentioned, the *content storage* is the Jena repository, which was designed to persist RDF triples (RDF is our semantic demarcation approach of choice). The *content management subsystem* encompasses a number of agents (considered jointly as a *data management agent* [*DMA*]) that work to assure that users of the system have access to the best quality of data. These agents, among others deal with: time sensitive information (such as changes of programs of movie theaters), incomplete data tokens, or inconsistent information (Angryk et al., 2002; Gordon & Paprzycki, 2005).

Content delivery subsystem has two roles. First it is responsible for the format (and syntax) of interactions between users and the system. However, this aspect of the system, as well as agents responsible for it, is mostly outside of scope of this chapter (more details can be found in Galant et al., 2002a and Kaczmarek et al., 2005). Second, it is responsible for the semantics of user-system interactions. Here two agents play crucial role. First, the *personalization infrastructure agent* (*PIA*) that consists of a number of extremely simple rule-based "RDF subagents" (each one of them is a class within the PIA) that extend the

set of travel objects selected as a response to the original query to create a *maximum response set* (*MRS*) that is delivered to the PA for filtering and ordering. Second, the PA that utilizes *user profile* to filter and hierarchically organize information obtained from the PIA as the MRS. It is also the PA that is involved in gathering explicit user feedback (see section "RDF Data Utilization: Content Personalization") that is used to adjust user profile.

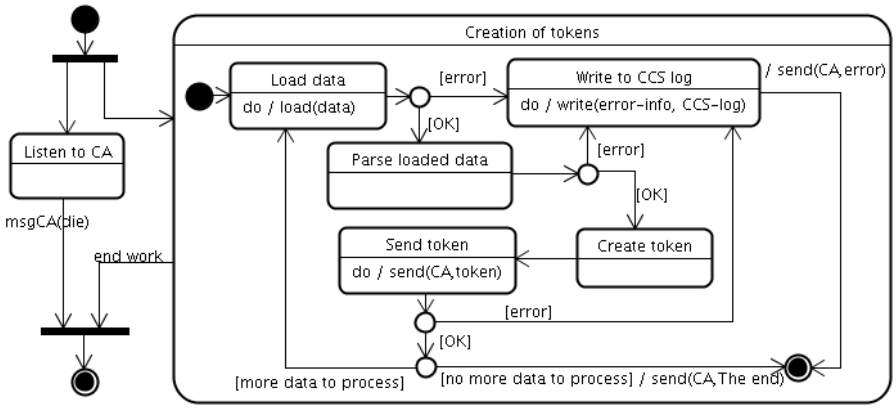
In Figure 2 we represent, in the form of a UML use case diagram, the aforementioned agents as well as other agents that are a part of the central system infrastructure. This diagram should be considered together with the system visualization found in Figure 1.

Since we had to abandon, hopefully temporarily, *other sources*, in Figure 2 we depict only Web sites and Web services that belong to the VCP category. They are sources of data for the function *Data Collection* that is serviced by WAs, *indexing agents* (*IA*), and a *coordinator agent* (*CA*). The IA communicates with the *DB agent (DBA)* when performing the *Inserting tokens* function. Separately, the CA receives data requests from the DMA. These data requests represent situations when data tokens were found to be potentially obsolete or incomplete (as a part of the *Data Management* function) and a new token has to be delivered by an appropriate WA to refresh/complete data available in the system. The DMA and the DBA are the only agents that have a direct access to the Jena database. In the content delivery subsystem we have three functions specified. The *Travel Service Selection* function is related to User(s) querying the system (information flow from the User to the central repository), while the *Response Delivery* function involves operations taking place between the time when the initial response to the query is obtained from Jena and when the final personalized response is delivered to the user (information flow from the central repository to the *User*). During this process the PIA performs the *Preparing MRS* function. Let us now discuss in some detail agents and their interactions. Before we proceed let us note that we omit a special situation when the system is initialized for the very first time and does not have any data stored in the Jena repository. While this situation requires agents started in a specific order, since it is only a one-time event it is not worthy of extra attention. We therefore assume that there is already data stored in the system and focus on interactions taking place in a working system.

The WA interfaces with Web sites, mapping XML- or HTML-demarcated data into RDF triples describing travel objects (according to the ontology used in our system [Gawinecki, Gordon, Nguyen, Paprzycki, & Szymczak, 2005; Gawinecki, Gordon, & Paprzycki, et al., 2005; Gordon, Kowalski, et al., 2005]). It is created by the CA on the basis of a *configuration file*. The configuration file may be created by the system administrator and sent to the CA as a message from the *graphical user interface (GUI) agent* or may be contained in a message from the DMA that wants to update one or more tokens. Each completed token is time stamped and priority stamped and send back to the CA. Upon completion of its work the (or in the case of an error) WA sends an appropriate message to the CA and self-destructs. A new WA with the same functionality is created by the CA whenever needed. Note that to simplify agent management we create instances of WA for each "job," even though they may produce tokens describing the same travel resource. For instance, when one WA is working on finding information about *all* Westin Hotels in Central Europe (task assigned by the system administrator), another WA may be asked to find information about Westin Hotel in Warszawa (job requested by the DMA). It is the role of the IA to assure that the most current available token is stored in the repository (see Figure 3). An UML statechart of the WA is contained in Figure 3.

CA manages all activities of the content collection subsystem. When started, it creates a certain number of IA (specified by the system

Figure 3. Statechart of the WA



administrator—*Servicing agent management request* function in Figure 4) and enters a listening state. There are six types of messages that may be received: (1) a self-destruction order received from the GUI Agent (send by the system administrator)—resulting in the CA killing all existing WAs and IAs first, and then self-destructing; (2) message from the WA that it encountered an error or that it has completed its work and will self-destruct—resulting in appropriate information being recorded; (3) message from the WA containing a token—to be inserted into the priority queue within the CA*;* (4) message from one of the IAs requesting a new token to be inserted into the repository—which results in the highest priority token being removed from the priority queue and send to the requesting IA. When the queue is empty, a message is send to the IA informing about this fact (as seen in Figure 5, IA will retry requesting token after some delay); (5) message from the DMA containing a request (in the form of a configuration file) to provide one or more tokens—resulting in creation of an appropriate WA (or a number of WAs); and, finally, (6) message from the GUI Agent ordering adjustment of the number of IAs in the system. A complete statechart of the CA is depicted in Figure 4.

IA is responsible for inserting tokens into the central repository as well as initial pre-processing of tokens to facilitate cleanness of data stored in the system. For the time being the IA performs the following simple checks: (1) time consistency of tokens to be inserted—since it is possible that multiple WAs generate tokens describing the same travel resource (see above), the IA compares time stamps of the token to be inserted with that in the repository and inserts its token only when it is newer; (2) data consistency—token to be used to update/append information has to be consistent with the token in the repository (e.g., the same hotel has to have the same address); and (3) inconsistent tokens are marked as such and they are to be deconflicted (Angryk et al., 2002). In the case when the priority queue is empty, request will be repeated after delay *T.* The statechart of the IA is represented in Figure 5 (top panel presents the overall process flow, while the bottom panel specifies processes involved in servicing tokens).

Let us now briefly describe the next three agents visible in Figure 2. The DBA represents interface between the database (in our case the Jena repository) and the agent system. It is created to separate an agent system from an "outside technology" in such a way that in case of changes in the repository all other changes will be localized to that agent, while the remaining parts of the system stay unchanged.

Figure 4. Statechart of the CA

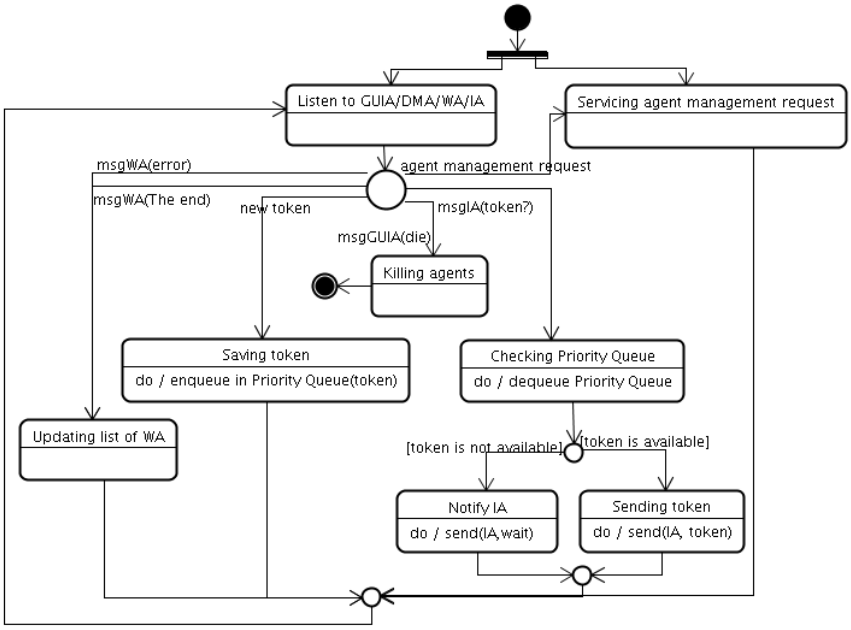
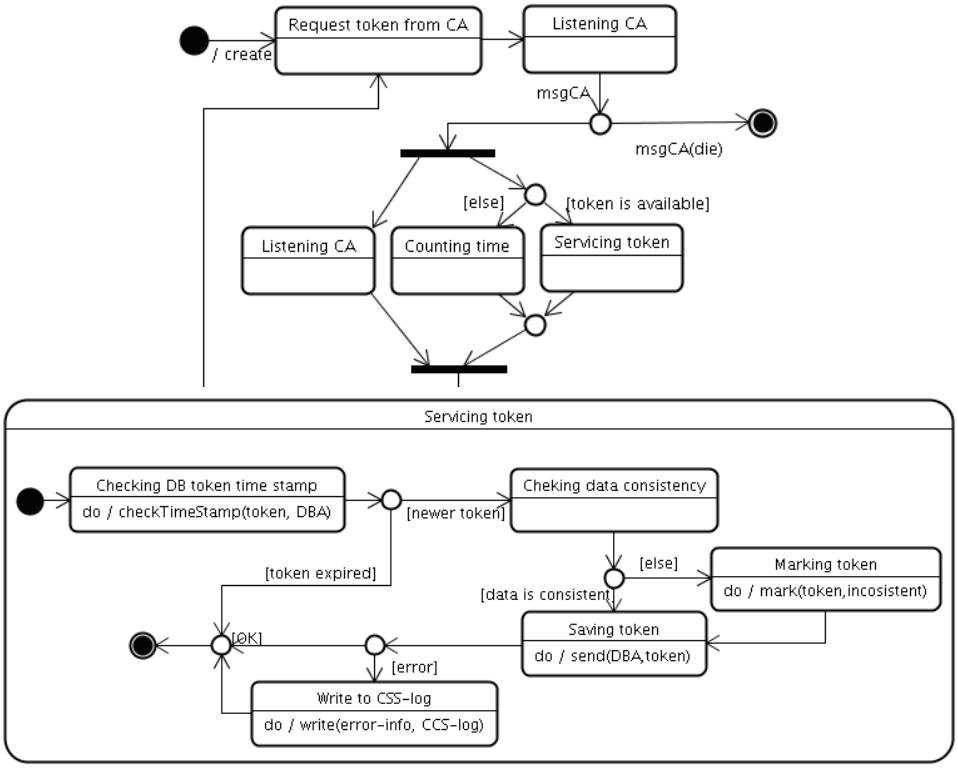


Figure 5. Statechart of the IA

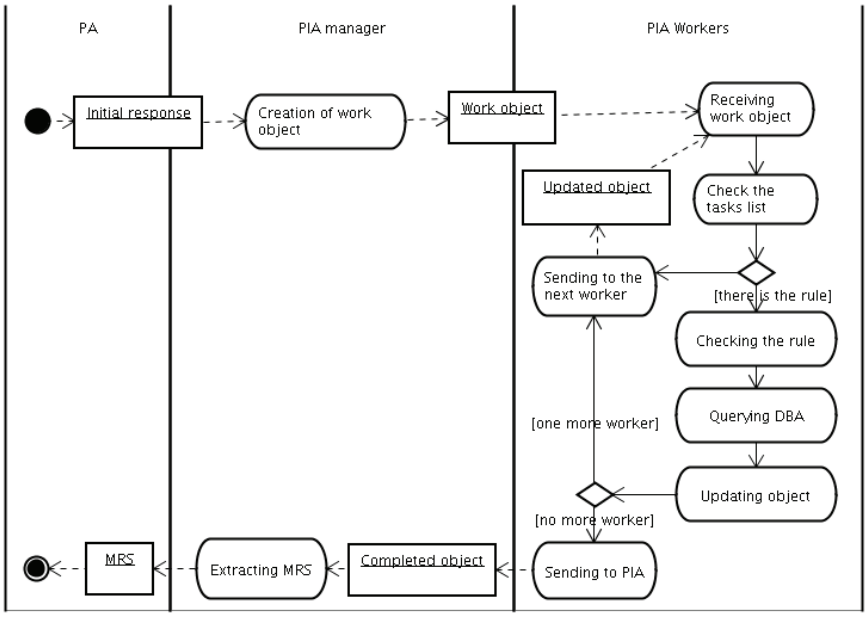


In the current system the DMA is a simple one. A number of agents of this type, responsible for different travel objects, are created upon system startup. Their role is to "traverse" the repository to find outdated and incomplete tokens and request new/additional ones to be generated to update/complete information stored in the repository. To achieve this goal DMAs generate a configuration file of an appropriate WA and send them to the CA for processing. In the future DMAs will be responsible for complete management of tokens stored in the repository to assure their completeness, consistency, and freshness.

The PIA consists of a manager and a number of "RDF subagents" (*PIA workers* in Figure 6). Each of these subagents represents one or more of simple rules of the type "Irish pub is also a pub" or "Japanese food is Oriental food." These rules are applied to the set of RDF triples returned by the initial query. Rule application involves querying the repository and is expected to expand the result set (e.g., if the user is asking for a Korean restaurant then other Oriental restaurants are likely to be included). The PIA subagents operate as a team passing the result set from one to the next (in our current implementation they are organized in a ring), and since their role is to maximize the set of responses to be delivered to the user no potential response is removed from the set. Final result of their operation is the MRS that is operated on by the PA. Action diagram of the PIA is depicted in Figure 6.

A separate PA will be created for each user and will play two roles in the content delivery subsystem. First, it is the central coordinator—for each user query it directs it from one agent to the next, constantly monitoring processing progress. Second, it utilizes user profile to filter and order responses that are to be sent to the user. More precisely, the user query, after being pre-processed and transformed into an RDQL query (see Kaczmarek et al., 2005 for more details), is being sent to the DBA. What is returned is the initial response consisting of a number of tokens that satisfy the query. This response is being redirected (by the PA) to the PIA to obtain the MRS. Then the PA utilizes the user profile to: (1) remove from the set responses that do not belong there (e.g., user

Figure 6. Action diagram of the PIA



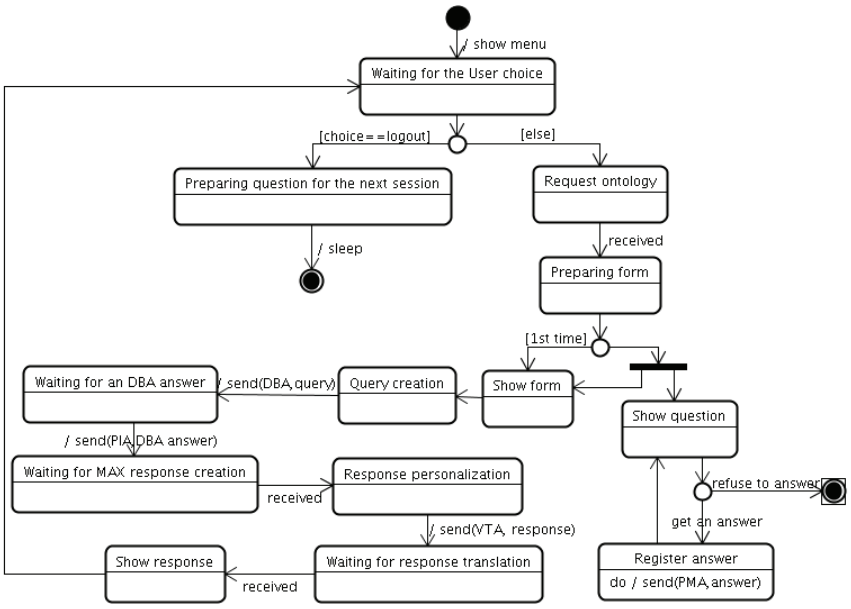
is known to be adversely inclined toward Italian food, and pizza in particular, and thus all of the Italian food serving restaurants have to be excluded); (2) order the remaining selections in such a way that those that are believed to be of most interest to the user will be displayed first (e.g., if user is known to stay in Hilton hotels, they will be displayed first). The statechart diagram of the PA is contained in Figure 7.

As we can see the PA behaves differently depending if the user is using the system for the first time or if it is a returning user. In the latter case, the PA will attempt at gathering explicit feedback related to the information delivered to the user during the previous session. This will be done through a generation of a questionnaire that will be shown to the user, who may decide to ignore it (see also Galant & Paprzycki, 2002). Obtained responses will be used to adjust the user profile. We can also see how the PA plays the role of response preparation orchestrator by always receiving responses from other agents and forwarding them to the next agent in the processing chain. We have selected this model of information processing so that "worker agents" like the DBA or the PIA know only one agent to interact with (the PA). Otherwise, an unnecessary set of dependencies would be introduced to the system making it substantially more difficult to manage (any change to one of these agents would have to be propagated to all agents that interact with it—while in our case only a single agent needs to be adjusted).

Replacing Semantic Web with a Semantic Database

As noted before, currently the Semantic Web is an attractive idea that lacks its main component—large repositories of semantically demarcated (in particular travel-related) data. This was one of the important reasons to change the design of our systems from data indexing into data gathering. As a result we are able to create our own "mini Semantic Web" (in the form of a semantic database) and store there information that in the future will allow us to extend our system beyond the basic skeleton described here, and start experimenting with its true projected functionalities—like content personalization.

Figure 7. Statechart of the PA



Let us describe how the HTML-demarcated information available on the Web is turned into semantic tokens representing travel objects in our repository. Before proceeding let us discuss briefly ontologies utilized in the system. As reported in Gawinecki, Gordon, Nguyen, et al., 2005; Gawinecki, Gordon, Paprzycki, et al., 2005; and Gordon, Kowalski, et al., 2005, while there exists a large number of attempts at designing ontologies depicting various aspects of the world, we were not able to locate a complete ontology of the most basic objects in the "world of travel" such as a *hotel* and a *restaurant*. More precisely, there exists an implicit ontology of restaurants utilized by the Chef Moz project (Chef Moz, 2005), but it cannot be used directly as a Semantic Web resource, due to the fact that data stored there is infested with bugs that make its automatic utilization impossible without pre-processing that also involves manual operations (see Gawinecki, Gordon, Paprzycki, et al., 2005 and Gordon, Kowalski, et al., 2005 for more details).

This being the case we have proceeded in two directions. First, as reported in Gawinecki, Gordon, and Paprzycki, et al. (2005) and Gordon, Kowalski, et al. (2005) we have reverse engineered the restaurant ontology underlying the Chef Moz project and cleaned data related to Polish restaurants. Separately we have proceeded with designing hotel ontology using a pragmatic approach. Our hotel ontology is to be used to represent, manipulate, and manage hotel information actually appearing within Web-based repositories (in context of travel; i.e., not hotels as landmarks, or sites of historical events). Therefore we have studied content of the 10 largest Internet travel agencies and found out that most of them describe hotels using very similar vocabulary. Therefore we used these common terms to shape our hotel ontology and the results of this process have been reported in Gawinecki, Gordon, Nguyen, et al. (2005); Gawinecki, Gordon, Paprzycki, et al. (2005); and Gordon, Kowalski, et al. (2005). As an outcome we have two fully functional, complete ontologies (of a hotel and of a restaurant) that are used to shape data stored in our Jena repository.

In this context, let us illustrate how we transform the VCP featured data into travel tokens. As an example we will utilize the Web site belonging to Hilton hotels (www.hilton.com). More precisely, let us look at some of the information that is available at the Web site of Hilton Sao Paulo Morumbi depicted in Figure 8.

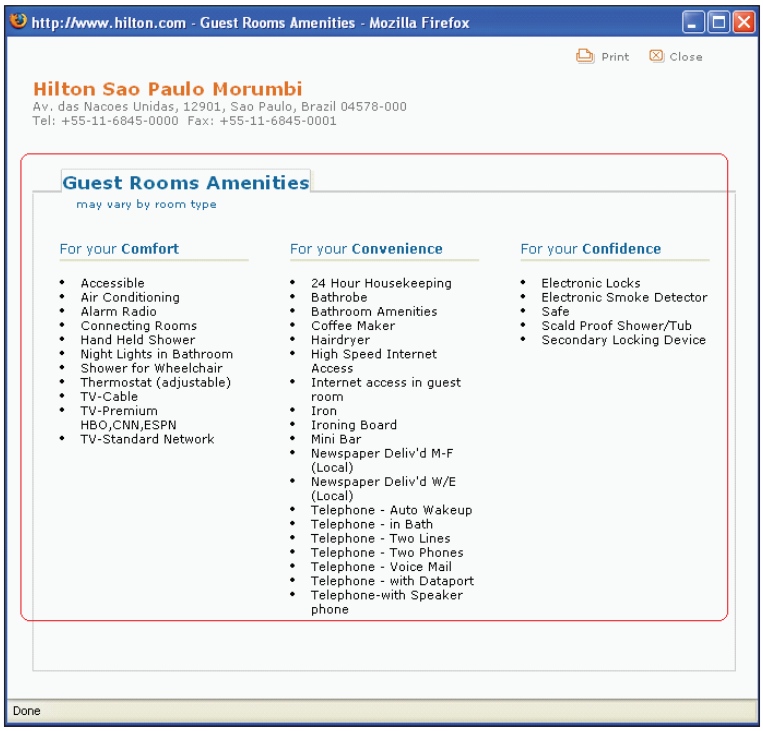
As clearly seen, from this page we can extract information such as the hotel name, address, and phone numbers. This page would also have to be interacted with, in case we planned to utilize our travel support system to make an actual reservation (which is only in very long-term plans and out of scope of this chapter). To find the remaining information defined by the hotel ontology requires traversing the Web site deeper. Therefore, for instance, the WA has to go to the page contained in Figure 9, to find information about hotel amenities.

As a result the following set of RDF triples (in XML-based notation) will be generated:

```
<rdf:Description
rdf:about="http://www.agentlab.net/travel/hotels/Hilton/
      SAOMOHI">
   <j.1:roomAmenit rdf:resource="http://.../hotel.
      rdf#AccessibleRoom"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#AirConditioning"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#ConnectingRooms"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#Shower"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#CableTelevision"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#CNNavailable"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#Bathrobe"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Bathro
      omAmenities"/>
 <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Coffee_
      TeaMaker"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#Hairdryer"/>
 <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#HighSpeedInternetC
      onnection"/>
   <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#InternetAccess"/>
```

Figure 8. Hilton Sao Paulo Morumbi main page

Figure 9. Hilton Sao Paulo Morumbi amenities page

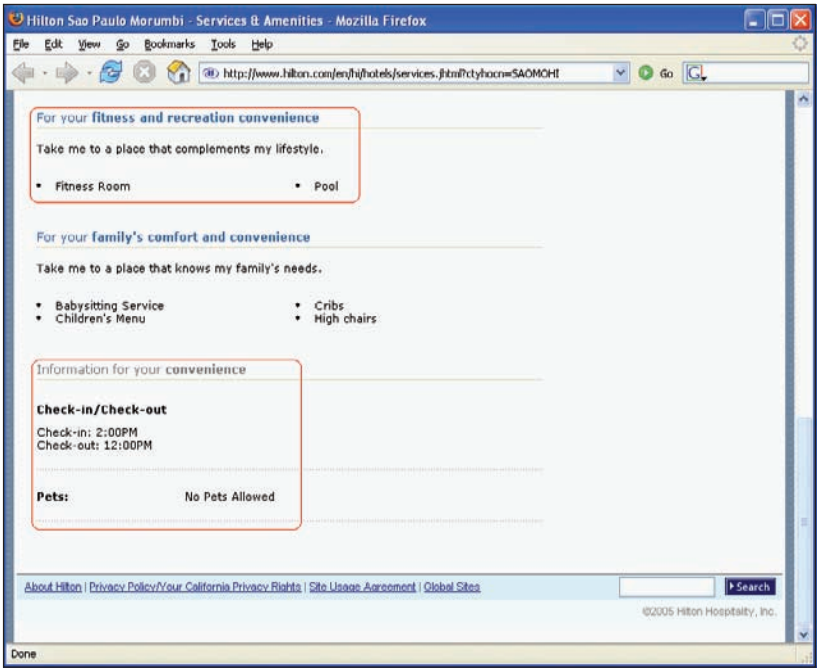


```
<j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Iron"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#IroningBoard"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#Minibar"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#Newspaper"/>
 <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Wake-
      upCalls"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Two-
      linePhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#VoiceMail"/>
<j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#TelephoneWithData
      Ports"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#SpeakerPhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.
      rdf#SmokeDetektors"/>
 <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Safe"/>
</rdf:Description>
```

These RDF triples represent a part of our hotel ontology, but this time they became its instance representing a given Hilton hotel (values of various aspects of the hotel are filled-in). Our WA will then continue traversing the hotel site to find, for instance, information about fitness and recreation as well as check-in and check-out times. An appropriate page belonging to the same hotel is depicted in Figure 10 while the resulting set of RDF triples follows.

```
<rdf:Description
rdf:about="http://www.agentlab.net/travel/hotels/Hil-
      ton/SAOMOHI">
 <j.1:recreationService
      rdf:resource="http://.../hotel.rdf#FitnessCenter
      Onsite"/>
 <j.1:recreationService
rdf:resource="http://.../hotel.rdf#IndoorOrOutdoorCo
      nnectingPool"/>
     <j.1:petsPolicy rdf:resource="http://.../hotel.
      rdf#NoPetsAlowed"/>
 <j.1:additionalDetail
```

Figure 10. Hilton Sao Paulo Morumbi fitness and recreation and check-in andcheck-out information



```
rdf:resource="http://www.agentlab.net/travel/hotels/Hil-
      ton/SAOMOHI/CheckIn-CheckOut"/>
</rdf:Description>

<rdf:Description
rdf:about="http://www.agentlab.net/travel/hotels/Hil-
      ton/SAOMOHI/CheckIn-CheckOut">
 <j.1:detail>Check-in: 2:00PM, Check-out: 12:00PM</
      j.1:detail>
</rdf:Description>
```

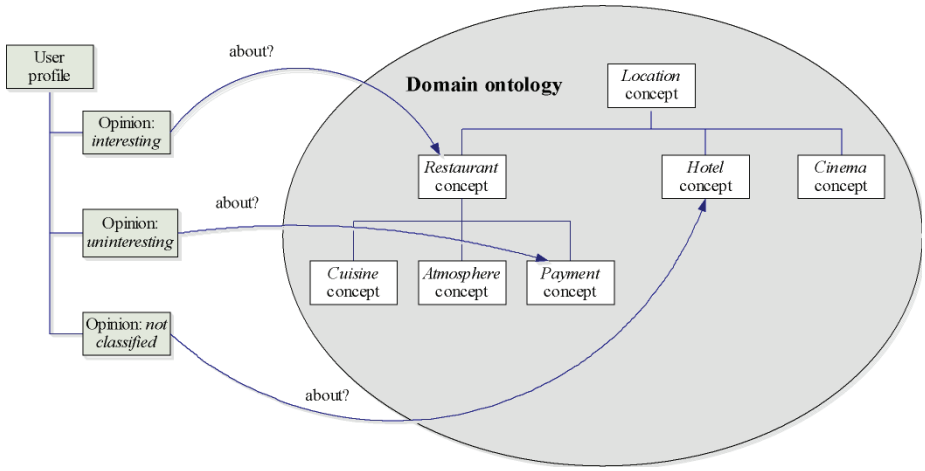
In this way the WA processes all necessary pages belonging to the Hilton Sao Paulo Morumbi and as a result obtains a set of RDF triples that constitute its complete definition (from the point of view of ontology utilized in our system). This set of RDF triples is then time and priority level stamped, packed into an ACL message and send to the CA that inserts it into the priority queue—to be later inserted, by the IA, to the semantic database. Depending on the assignment the WA may continue producing tokens of other Hilton hotels or, if work is completed, it informs the CA about this fact and self-destructs. In this way in our system, by manually creating WAs for a variety of travel information sources, we can collect real-life data representing actual travel objects.

RDF Data Utilization: Content Personalization

Let us now discuss how the data stored in the system is used to deliver personalized responses to the user. While our approach to user profile construction and utilization is based on ideas presented in Burke (2002); Fink and Kobsa (2002); Galant and Paprzycki (2002); Kobsa, Koenemann, and Pohl (2001); Montaner, López, and De La Rosa (2003); Rich (1979) and Sołtysiak and Crabtree (1998), however utilization of these methods in the context of ontologically demarcated information is novel and was proposed originally in Gawinecki, Vetulani, Gordon, and Paprzycki (2005).

To be able to deliver personalized content to the user of the system, we have to be able to represent the user in the system first—define user profile. Furthermore, the proposed user profile has to be created in such a way to simplify interactions in the system. Since our system is oriented toward processing of ontologically demarcated data, it is very natural to represent user preferences in the same way. Thus we adapted an *overlay model* of user profile, where opinions are "connected" with appropriate concepts in the domain ontology. This approach is also called a *student model*, since it has been found useful to describe knowledge of the student about specific topics of the domain

Figure 11. Overlay model utilized to represent user profile



(Greer & McCalla, 1994). Basic tenets of the overlay model are depicted in Figure 11.

For instance, let us consider our hotel ontology and assume that the user likes to stay in hotels that have both a *pool* and *fitness center*. Both these features are subclasses of the concept *amenities*. We can represent user interest by assigning weight to each amenity (the larger the weight the more important the given feature is to the user). In case of our hypothetical customer, both pool and exercise room will be assigned large weights, while features that user is not particularly interested in (e.g., availability of ironing board—see Figure 9) will be assigned small weight—the lesser the interest is the closer to 0 the value will be. In the case of features about which we do not know anything about users' preferences, no value will be assigned (see Figure 11). Let us observe that in this approach we are mimicking the notion of probability—all assigned values are from the interval (0, 1). This means that even in the case of strong counter preference towards a given feature we will assign value 0 (there are no negative values available). Proceeding in this described way, we will create a special instance of hotel ontology, one that represents *user-hotel-profile*. The following fragment of an instance of hotel ontology (this time represented in an N3 notation) depicts user (Karol) profile as it is represented in our system:

```
:KarolOpinions    a sys:OpinionsSet;
  sys:containsOpinion
    [sys:about hotel:Pool;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.89].
    [sys:about hotel:ExerciseRoom;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.84].
    [sys:about res:AirConditioning;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.89].
    [sys:about hotel:BathroomAmenities;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.73].
    [sys:about hotel:IroningBoard;
      sys:hasClassification sys:NotInteresting;
      sys:hasNormalizedProbability 0.11].
    [sys:about hotel:Iron,
      sys:hasClassification sys:NotInteresting;
      sys:hasNormalizedProbability 0.15].
```

The previous hotel profile of Karol, tells us that he likes to stay in hotels with swimming pool and exercise room, while the availability of an iron and ironing board is inconsequential to him.

Obviously, somewhere in the system we have to store, in some form, information about the user. To assure consistency across the system, this is done in the form of a simplistic user ontology. Next, we present a fragment of such ontology:

```
:hasDress a rdf:Property ;
    rdfs:range :Dress ;
    rdfs:domain :UserProfileData.

:hasAge a rdf:Property ;
        rdfs:range :Age ;
        rdfs:domain :UserProfileData .

:hasWealth a rdf:Property ;
        rdfs:range :Wealth ;
        rdfs:domain :UserProfileData .

:hasProfession a rdf:Property ;
        rdfs:range :Profession ;
        rdfs:domain :UserProfileData .
```

Let us now assume that Karol is a 24-year-old painter, who has enough money to feel rich and whose dressing style is a natural one, then his profile would be represented as:

```
:KarolProfile a sys:UserProfile;
  sys:hasUserID 14-32-61-3456;
  sys:hasUserProfileData :KarolProfileData;
  sys:hasOpinionsSet :KarolOpinions.

:KarolProfileData a sys:UserProfileData;
  sys:hasAge 24;
  sys:hasWealth sys:Rich;
  sys:hasDress sys:NaturalDress;
  sys:hasProfession sys:SpecialistFreeLancer.
```

Rather than keeping them separate, we combine instances of user ontology with the previously described user profile into a complete ontological description—a comprehensive user profile. This user profile is then to be stored in the Jena repository.

One of the important questions that all recommender systems have to address is, how to "introduce" new users to the system (Galant & Paprzycki, 2002). In our system we use stereo-

typing (Rich, 1979). Obviously, we represent stereotypes the same way we used to represent user profiles, with the difference that instead of specific values representing preferences of a given user, we use sets of variables of nominal (to represent categories—e.g., profession), ordinal (e.g., low income, medium income, high income), and interval (e.g., age between 16 and 22) types. For values of nominal and ordinal types we have established sets of possible values, while for the values of interval types, we defined borders of intervals considered in the system. Using results of a survey and expert knowledge, we were able to create restaurant-related stereotypes (one instance of restaurant ontology of each identified stereotype). To illustrate such a case, here is a fragment of artistic profile in the area of restaurants:

```
:ArtistStereotypeOpinions    a sys:OpinionsSet;
 sys:containsOpinion
   [sys:about res:CafeCoffeeShopCuisine;
     sys:hasClassification sys:Interesting;
     sys:hasNormalizedProbability 1.0].
   [sys:about res:CafeteriaCuisine;
     sys:hasClassification sys:Interesting;
     sys:hasNormalizedProbability 0.75].
   [sys:about res:TeaHouseCuisine;
     sys:hasClassification sys:Interesting;
     sys:hasNormalizedProbability 0.9].
   [sys:about res:WineBeer;
     sys:hasClassification sys:Interesting;
     sys:hasNormalizedProbability 0.8].
   [sys:about res:WineList;
     sys:hasClassification sys:Interesting;
     sys:hasNormalizedProbability 1.0].
   [sys:about res:HotDogsCuisine;
     sys:hasClassification sys:NotInteresting;
     sys:hasNormalizedProbability 0.0].
```

In this stereotype we can see, among others, that an *artist* has been conceptualized as a person who likes *coffee houses* a bit more than *tea houses* and is willing to eat in a *cafeteria*, likes *wine* (a bit more than *beer*), but does not like *hot dogs* (*fast food*). Other stereotypes have been conceptualized similarly and their complete list and a detailed description of their utilization can be found in Gawinecki, Kruszyk, and Paprzycki (2005).

When a new user logs onto the system he/she will be requested to fill out a short questionnaire about age, gender, income level, occupation, address (matching user features defined by the user ontology), as well as questions about travel preferences. While the basic user ontology-based data will be required, answering questions about travel preferences will be voluntary. Personal data collected through the questionnaire will be used to match a person to a stereotype. More precisely, we will calculate a distance measure between user-specified characteristics and these appearing in stereotypes defined in the system and find one that matches his/her profile the closest. To achieve this we will use the following formula:

$$d\left(\hat{S},\hat{u}\right)=\frac{\sum_{f=1}^{k} w^{f}\delta_{\hat{S}\hat{u}}^{f}d_{\hat{S}\hat{u}}^{f}}{\sum_{f=1}^{k} w^{f}\delta_{\hat{S}\hat{u}}^{f}}$$

Where: w^f – weight of attribute, $d^f_{S,u}$ – distance between values of the attribute in the stereotype S and user's data u, $\delta^f_{S,u}$ – Boolean flag that informs whether attribute f appears in both: stereotype's data (S) and user's data (u).

To illustrate this, let us consider Karol, the painter, again. In the Table 1 we present Karol's data and the artist stereotype data and show how the closeness between Karol and that stereotype is calculated.

The same process is then repeated comparing Karol's data against all other stereotypes to find the one that fits him the best. In the next step this stereotype is joined with his user data to become his *initial profile*. In the case when he answers any domain-specific questions (recall, that he may omit them), this data will be used to modify his user profile. For example, let us assume that he has been identified as student stereotype, but he has also specified that he does not like coffee houses (while in the student stereotype coffee

Table 1. Calculating closeness between user profile (Karol) and a stereotype (artist)

Attribute (f)	Attribute weight (w^f)	Data of artist stereotype (comma means OR relation): (S)	Karol's Data: (u)	Distance between value of attribute: ($d^f_{S,u}$)	Weighted distance: ($w^{f*} d^f_{S,u}$)
Age	2	20-50	24	0.00	0.00
Wealth	4	Not Rich, Average Rich	Rich	0.33	1.33
Dress	1	Naturally, Elegantly	Naturally	0.00	0.00
Profession	2	Student/Pupil, Scientist/ Teacher, Specialist/ FreeLancer Unemployed/WorkSeeker	Specialist/ FreeLancer	0.00	0.00
			COMBINED		1.3(3) / (2+4+1+2)= 0.14(6)

houses have been assigned a substantial positive weight). Obviously, in his profile, this positive value will be replaced by zero—as explicit personal preferences outweigh these specified in the stereotype (see also Nistor, Oprea, Paprzycki, & Parakh, 2002):

```
:KarolOpinions a sys:OpinionsSet;
  sys:containsOpinion
    [sys:about res:CafeCoffeeShopCuisine;
      sys:hasClassification sys:Interesting;
      sys:hasNormalizedProbability 0.0].
```

Observe that as soon as the system is operational we will be able to store information about user behaviors (Angryk et al., 2003; Galant & Paprzycki, 2002; Gordon & Paprzycki, 2005). These data will be then used not only to modify individual user profiles, but also mined (e.g., clustered) to obtain information about various group behaviors taking place in the system. This information can be used to verify, update, or completely replace our initial stereotypes. Such processes are based on the so-called implicit relevance feedback (Fink & Kobsa, 2002; Kobsa et al., 2001). As described earlier (see Figure 7) we will also utilize explicit feedback based on user responses to subsequent questionnaires. Currently as explicit feedback we utilize only a single question: "Did you like our main suggestion presented last time?" but a more intricate questionnaire could also be used. Specifically, at the end of each user system interaction, on the basis of what was recommended to the user, a set of questions about these recommendations could be prepared. When the user returns to the system, these questions would be then asked to give him/her opportunity to express his/her direct opinion. Both implicit and explicit feedbacks are used to adjust user profile (see also Gawinecki, Vetulani, et al., 2005). Note here, that in most recommender systems stereotyping is *the* method of information filtering (demographic filtering); thus making such systems rather rigid—in this case individual user preferences cannot be properly modeled and modified (Kobsa et al., 2001). In our system we use stereotyping only to solve the cold-start problem—and modify them over time—and thus avoid the rigidity trap.

User profile is utilized by the PA to rank and filter travel objects. Let us assume that after the query, the response preparation process has passed all stages and in the last one the PIA agent has completed its work and the MRS has been delivered to the PA. The PA has now to compute a *temperature* of each travel object that is included in the MRS. The temperature represents the "probability" that a given object is a "favorite" of the user. This way of calculating the importance of selected objects was one of the reasons for the way that we have assigned importance measures to individual features (as belonging to the interval

[0,1]). Recall here that the DBA and the PIA know nothing about user preferences and that the PIA uses a variety of general rules to increase the response set beyond that provided as a response to the original query.

To calculate the temperature of a travel object (let us name it an *active object*) three aspects of the situation have to be taken into account. First, features of the active object. Second, user interests represented in the user profile—if a given feature has no preference specified then it cannot be used. In other words, for each token in the MRS we will crop its ontological graph to represent only these features that are defined in user profile. Third, features requested in user query. More specifically, if given keywords appear in the query (representing explicit wishes of the user), for example, if the query was about a restaurant in Las Vegas, then such restaurants should be presented to the user first. Interactions between these three aspects are represented in Figure 12.

Here we can distinguish the following situations:

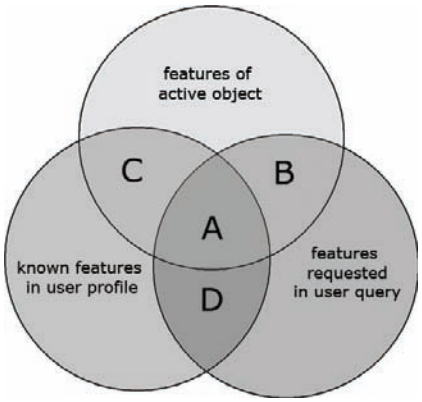
A. Features explicitly requested by the user that appear in the active object as well as in the user-profile;
B. Features requested by the user and appearing in the active object;
C. Features not requested that are a part of the user profile and that appeared in the active object; and
D. Features that do not appear in the active object (we are not interested in them).

Ratings obtained for each token in the MRS represent what the system believes are user preferences and are used to filter out these objects temperatures of which are below a certain threshold and rank the remaining ones (objects with highest scores will be displayed first). We will omit discussion of a special case when there is no object above the threshold. The MRS is processed in the following way:

1. Travel objects are to be returned to the user in two groups (buckets)
   a. Objects requested explicitly by the user (via the query form) – Group I
   b. Objects not requested explicitly by the user but predicted by the system to be of potential interest to the user – Group II

   Thus, for each active object we divide features according to the areas depicted in Figure 11. Objects for which at least one feature is inside of either area A or B belong to Group I, objects with all features inside area C belong to Group II, while the remaining objects are discarded.

Figure 12. Construction of final response: Interactions between features



2. Inside of each bucket travel objects are sorted according to their temperature computed in the following way: for a given object *O* its temperature

$$temp(0) = \sum_{f \in O} temp(f)$$

where $temp(f) = 1$ if $f \in A \cup B$, or $p_n(f)$ if $f \in C$, while $temp(f)=temp(f) - 0.5$. This latter calculation is performed to implicate that these features that are not of interest to the user (their individual temperatures are less than 0.5) reduce the overall temperature of the object. Function $p_n(f)$ is a normalized probability of feature *f*, based on the user profile.

Let us consider Karol, who is interested in selecting a restaurant. In his query he specified that this restaurant has to serve Italian cuisine and has to allow smoking. Additionally, we know, from Karol's profile, that he does not like *coffee* (weight 0.1) and *outdoor dining* (weight 0.05). Thus for the restaurant X:

```
:RestaurantX a res:Restaurant;
      res:cuisine res:ItalianCuisine;
      res:cuisine res:PizzaCuisine;
      res:cuisine res:CafeCoffeeShopCuisine;
      res:feature res:Outdoor.
```

the overall score will be decreased due to the influence of *Outdoor* and *CafeCoffeeShopCuisine* features, but will receive a "temperature boost" because of the *ItalianCuisine* feature (explicitly specified feature). However, the restaurant X it won't be rated as high as the restaurant Y:

```
:RestaurantY a res:Restaurant;
      res:cuisine res:ItalianCuisine;
      res:smoking res:PermittedSmoking.
```

which serves *ItalianCuisine*, where smoking is also permitted. To be more specific, let us consider these two restaurants and the third one described by the following features:

```
:RestaurantZ a res:Restaurant;
      res:cuisine res:WineBeer;
      res:smoking res:PermittedSmoking.
```

Then Table 2 represents the way that temperatures of each restaurant will be computed.

As a result, restaurants X and Y belong to the first bucket (to be displayed to the user as they both have features that belong to area B). However,

Table 2. Computing temperature of a restaurant

Restaurant N3 descriptions (bold – requested by the user, underlined – in the user profile; could be conjunctive)	Calculations
:RestaurantX a res:Restaurant; res:cuisine **res:ItalianCuisine**; res:cuisine res:PizzaCuisine; res:cuisine res:CafeCoffeeShopCuisine; res:feature res:Outdoor.	+0.5 (=1-0.5) requested; B +0 -0.49 (=0.01-0.5) profile -0.45 (=0.05-0.5) profile **= -0.44**
:RestaurantY a res:Restaurant; res:cuisine **res:ItalianCuisine**; res:smoking **res:PermittedSmoking**.	+0.5 (=1-0.5) requested; B +0.5 (=1-0.5) requested; B **= 1**
:RestaurantZ a res:Restaurant; res:cuisine res:WineBeer; res:smoking res:PermittedSmoking.	+0.3 (=0.8-0.5) not requested; profile; C +0.5 (=1-0.5) not requested; profile; C **= 0.8**

while restaurant Y has high temperature (1) and definitely should be displayed, restaurant X has very low temperature (-0.44) and thus will not likely be displayed at all. Interestingly, restaurant Z, which belongs to the second bucket (belongs to area C), has an overall score of 0.8 and is likely to be displayed. This example shows also the potential adverse effect of lack of information (e.g., in the ChefMoz repository; but more generally, within the Web) on the quality of content-based filtering (at least done in a way similar to that proposed previously). Simply said, what we do not know cannot decrease the score, and thus a restaurant for which we know only address and cuisine may be displayed as we do not know that it allows smoking on the premises (which would make it totally unacceptable to a given user).

RDF Data Utilization: Content Delivery

Let us now present in more detail how the delivery of content to the user is implemented as an agent system. To be able to do this we need to briefly introduce additional agents (beyond these presented in Figure 2) and their roles (using Prometheus methodology [Prometheus, 2005])—as represented in Figure 13.

In addition to the PA (described in details in Figure 7) and the DBA, we have also: (1) *view transforming agent* (*VTA*) responsible for delivering response in the form that matches the user I/O device; (2) *proxy agent* (*PrA*) that is responsible for facilitating interactions between the agent system and the outside world (need for these agents as well as a detailed description of their implementation can be found in Kaczmarek et al. (2005); (3) *session handling agent* (*SHA*), which is responsible for complete management and monitoring of functional aspects of user interactions with the system; and (4) *profile managing agent* (*PMA*) which is responsible for (a) creating profiles for new users, (b) retrieving profiles of returning users and (c) updating user-profiles, based on implicit and explicit relevance feedback. Let us now summarize processes involved in content delivery through a UML action diagram. While rather complex, descriptions contained in Figure 14 represent a complete conceptualization of actions involved in servicing user request from the moment that the user logs on to the system, to the moment when he/she obtains response to their query.

State of the System

As indicated earlier in this chapter, we have concentrated on these features of our system that are currently being implemented and close to being ready, while omitting the features that

Figure 13. Content delivery agents and their roles

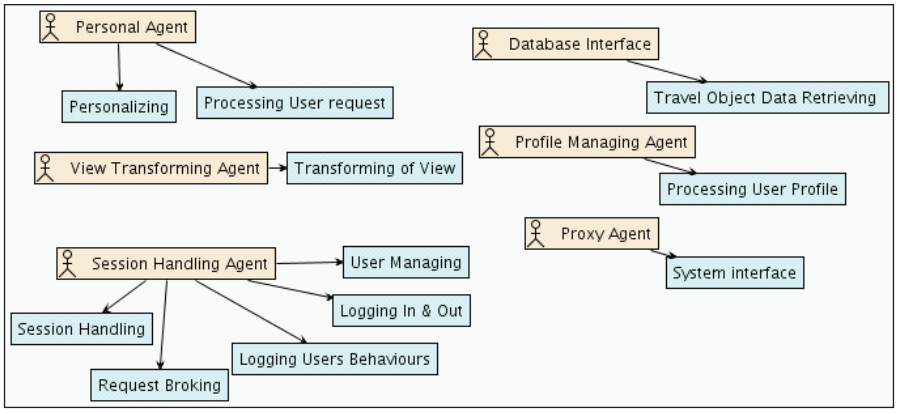
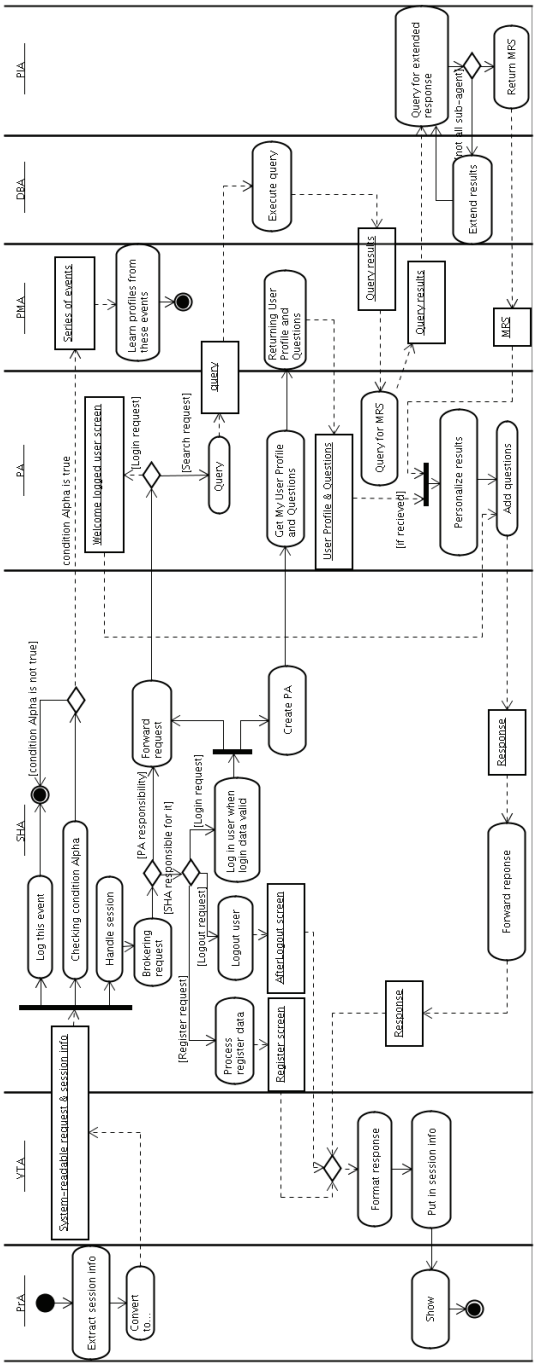


Figure 14. Content delivery action diagram



we would like to see developed in the future. While the interface to the system is still under construction, it is possible to connect to it from a browser. Furthermore, we have emulated WAP-based connectivity. As of the day this chapter is being written, we have implemented a function-complete content collection subsystem consisting of: (1) a number of hotel wrappers (WA) that allow us to feed hotel data into the system; (2) CA and IA agents that collaborate with the WAs to insert data into Jena-based repository; and (3) an initial version of the DMA and the PIA. For the CCS we have semi-automatically cleaned-up subsets of Chef Moz data, describing selected restaurants. We have also a relatively complete content delivery subsystem. In particular, (1) the PrA, the SHA, and the VTA that facilitate user-system interactions have been implemented and tested; (2) the PA is working as described in this chapter (with the PIA working in the case of restaurants only); (3) the PMA has only limited capacity, it is capable of creating and managing a single user profile; (4) while the existing set of stereotypes involves only restaurants. Let us briefly illustrate the work of the system, by screen-shots of the query (Figure 15) and the response (Figure 16). The query was a general query about restaurants in Greensboro, NC; note the box that attempts at asking a question about *Bistro Sophia* that was suggested to the user in the previous session (Figure 16).

One of the biggest problems related to testing our system is the fact that, being realistic, no user would be interested in a system that only provides a few hotel chains and restaurants (e.g., in Poland). This being the case we can ourselves test features of the system like: (1) is the user query handled correctly, that is, do the returned results represent the correct answer taking into account the current state of the system; (2) do the WAs correctly deliver and the CA and IAs accurately insert tokens into the system; and (3) are agent communication and interactions proceeding without deadlocks and does the system scale. Unfortunately, it is practically impossible to truly test adaptive features

Figure 15. System query screenshot

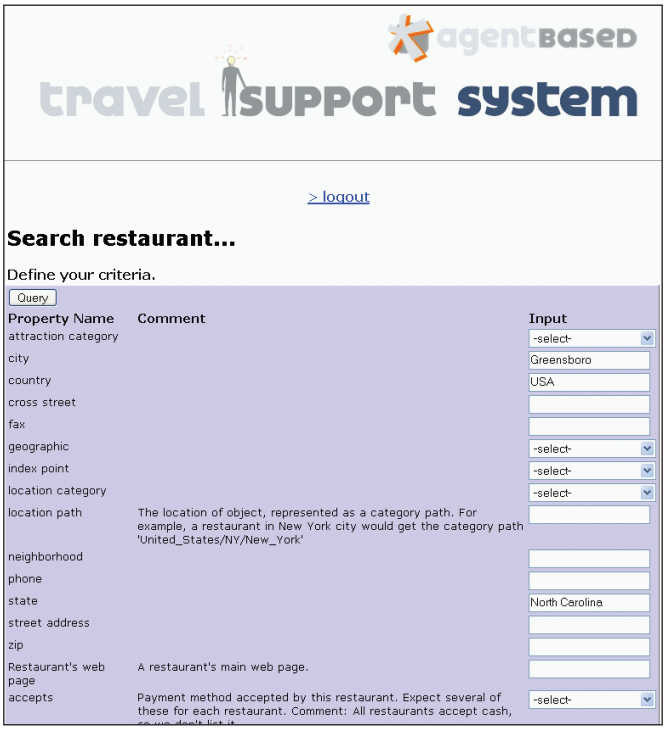
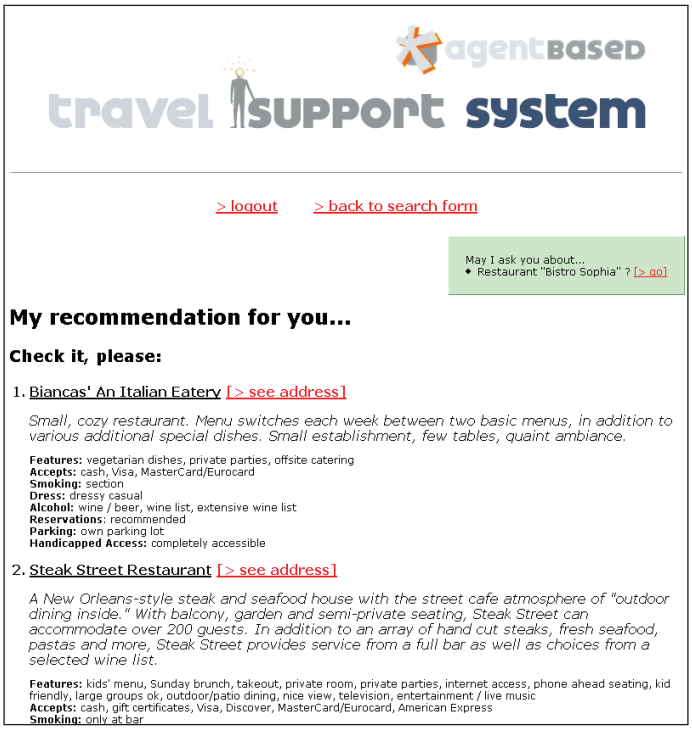


Figure 16. System response screenshot



of the system. Without actual users utilizing the system to satisfy their real travel needs, all the work that we have done implementing the system cannot be practice verified.

This is a more general problem of the chicken-and-egg type that is facing most of Semantic Web research (regardless of its application area). Without real systems doing real work and utilizing actual ontologically demarcated data on a large scale (to deliver what users need) it is practically impossible to assess if the Semantic Web, the way it was conceptualized, is the way that we will be able to deal with information overload, or is it just another pipe dream like so many in the past of computer science.

FUTURE DEVELOPMENTS

As described previously, it seems to be clear what the future of the development of Semantic Web technologies applied in context of e-business (or in any other context) has to be. It has to follow the positive program put forward by Nwana and Ndumu (1999). The same way as agent systems and a large number of systems utilizing Semantic Web technologies have to be implemented and experimented with. Furthermore, it is necessary to develop tools that are going to speed up ontological demarcation of Web content. Here, both the content that is about to be put on the Web as well as tools supporting demarcation of legacy content need to be improved and popularized. Only then, we will be able to truly assess the value proposition of the Semantic Web. Furthermore, since software agents and the Semantic Web are truly intertwined, the development of the Semantic Web should stimulate development of agent systems, while development of agent systems is likely to stimulate development of the Semantic Web.

To facilitate these processes we plan to continue development of our agent-based travel support system. The first step will be to complete integration and testing of the aforementioned described system skeleton. We will proceed further by: (1) developing ontologies of other important travel objects, for example, movie theaters, museums, operas, and so forth; (2) fully developing and implementing the PIA and the DMA infrastructures—according to the previously presented description; (3) continuing implementing WAs to increase the total volume of data available in the system; (4) adding a geographic information system (GIS) component to the system, to allow answering queries like: which restaurant is the closest one to that hotel?; (5) developing and implementing an agent-based collaborative filtering infrastructure; and (6) investigating the potential of utilizing text processing technologies for developing new generation of adaptive WAs.

REFERENCES

Angryk, R., Galant, V., Gordon, M., & Paprzycki, M. (2002). Travel support system: An agent based framework. In H. R. Arabnia & Y. Mun (Eds.), *Proceedings of the International Conference on Internet Computing (IC'02)* (pp. 719-725). Las Vegas, NV: CSREA Press.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web. *Scientific American.* Retrieved May, 2001, from http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction, 12*(4), 331-370.

ChefMoz. (2005). *ChefMoz dining guide*. Retrieved November, 2004, from http://chefdmoz.org

Darpa Agent Markup Language (DAML). (2005). *Language overview.* Retrieved October, 2005, from http://www.daml.org/

Fensel, D. (2001). *Ontologies: A silver bullet for knowledge management and electronic commerce*. Berlin: Springer.

Fink, J., & Kobsa, A. (2002). User modeling for personalized city tours. *Artificial Intelligence Review, 18*, 33-74.

Galant, V., Gordon, M., & Paprzycki, M. (2002a). Agent-client interaction in a Web-based e-commerce system. In D. Grigoras (Ed.), *Proceedings of the International Symposium on Parallel and Distributed Computing* (pp. 1-10). Iasi, Romania: University of Iaşi Press.

Galant, V., Gordon, M., & Paprzycki, M. (2002b). Knowledge management in an Internet travel support system. In B. Wiszniewski (Ed.), *Proceedings of ECON2002, ACTEN* (pp. 97-104). Wejcherowo: ACTEN.

Galant, V., Jakubczyc, J., & Paprzycki, M. (2002). Infrastructure for e-commerce. In M. Nycz & M. L. Owoc (Eds.), *Proceedings of the 10th Conference Extracting Knowledge from Databases* (pp. 32-47). Poland: Wrocław University of Economics Press.

Galant, V., & Paprzycki, M. (2002, April). Information personalization in an Internet based travel support system. In *Proceedings of the BIS'2002 Conference* (pp. 191-202). Poznań, Poland: Poznań University of Economics Press.

Gawinecki, M., Gordon, M., Nguyen, N., Paprzycki, M., & Szymczak, M. (2005). RDF demarcated resources in an agent based travel support system. In M. Golinski et al. (Eds.), *Informatics and effectiveness of systems* (pp. 303-310). Katowice: PTI Press.

Gawinecki, M., Gordon, M., Paprzycki, M., Szymczak, M., Vetulani, Z., & Wright, J. (2005). Enabling semantic referencing of selected travel related resources. In W. Abramowicz (Ed.), *Proceedings of the BIS'2005 Conference* (pp. 271-290). Poland: PoznaD University of Economics Press.

Gawinecki, M., Kruszyk, M., & Paprzycki, M. (2005). Ontology-based stereotyping in a travel support system. In *Proceedings of the XXI Fall Meeting of Polish Information Processing Society* (pp. 73-85). PTI Press.

Gawinecki, M., Vetulani, Z., Gordon, M., & Paprzycki, M. (2005). Representing users in a travel support system. In H. Kwaśnicka et al. (Eds.), *Proceedings of the ISDA 2005 Conference* (pp. 393-398). Los Alamitos, CA: IEEE Press.

Gilbert, A., Gordon, M., Nauli, A., Paprzycki, M., Williams, S., & Wright, J. (2004). Indexing agent for data gathering in an e-travel system. *Informatica, 28*(1), 69-78.

Gordon, M., Kowalski, A., Paprzycki, N., Pełech, T., Szymczak, M., & Wasowicz, T. (2005). Ontologies in a travel support system. In D. J. Bem et al. (Eds.), *Internet 2005* (pp. 285-300). Poland: Technical University of Wrocław Press.

Gordon, M., & Paprzycki, M. (2005). Designing agent based travel support system. In *Proceedings of the ISPDC 2005 Conference* (pp. 207-214). Los Alamitos, CA: IEEE Computer Society Press.

Greer, J., & McCalla, G. (1994). *Student modeling: The key to individualized knowledge based instruction* (pp. 3-35). NATO ASI Series. Springer-Verlag.

Harrington, P., Gordon, M., Nauli, A., Paprzycki, M., Williams, S., & Wright, J. (2003). Using software agents to index data in an e-travel system. In N. Callaos (Ed.), *Electronic Proceedings of the 7th SCI Conference* [CD-ROM, file: 001428].

Hendler, J. (1999, March 11). Is there an intelligent agent in your future? *Nature*. Retrieved March, 2004, from http://www.nature.com/nature/webmatters/agents/agents.html

Hendler, J. (2001). Agents and Semantic Web. *IEEE Intelligent Systems Journal, 16*(2), 30-37.

JADE. (2005). (Version 3.4) [Computer software]. Retrieved from http://jade.tilab.com/

Jena. (2005, March). *A Semantic Web framework* (Version 2.4) [Computer software]. Retrieved from http://www.hpl.hp.com/semweb/jena2.htm

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM, 44*(4), 35-41.

Kaczmarek, P., Gordon, M., Paprzycki, M., & Gawinecki, M. (2005). The problem of agent-client communication on the Internet. *Scalable Computing: Practice and Experience, 6*(1), 111-123.

Kobsa, A., Koenemann, J., & Pohl, W. (2001). Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review, 16*(2), 111-155.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM, 37*(7), 31-40.

Manola, F., & Miller, E. (Eds.). (2005). *RDF primer*. Retrieved from http://www.w3.org/TR/rdf-primer

McGuinness, D. L., & Van Harmelen, F. (Eds.). (2005, February 10). *OWL Web ontology language overview*. Retrieved December, 2004, from http://www.w3.org/TR/owl-features/

Montaner, M., López, B., & De La Rosa, J. L. (2003). A taxonomy of recommender agents on the Internet. *Artificial Intelligence Review, 19,* 285-330.

Nistor, C. E., Oprea, R., Paprzycki, M., & Parakh, G. (2002). The role of a psychologist in e-commerce personalization. In *Proceedings of the 3rd European E-COMM-LINE 2002 Conference* (pp. 227-231). Bucharest, Romania: IPA S. A.

Nwana, H., & Ndumu, D. (1999). A perspective on software agents research. *The Knowledge Engineering Review, 14*(2), 1-18.

Paprzycki, M., Angryk, R., Kołodziej, K., Fiedorowicz, I., Cobb, M., Ali, D., et al. (2001) Development of a travel support system based on intelligent agent technology. In S. Niwiński (Ed.), *Proceedings of the PIONIER 2001 Conference* (pp. 243-255). Poland: University of PoznaD Press.

Paprzycki, M., Kalczyński, P. J., Fiedorowicz, I., Abramowicz, W., & Cobb, M. (2001) Personalized traveler information system. In B. F. Kubiak & A. Korowicki (Eds.), *Proceedings of the 5th International Conference Human-Computer Interaction* (pp. 445-456). Gdańsk, Poland: Akwila Press.

Raccoon. (2005). (0.5.1) [Computer software]. Retrieved November 2005, from http://rx4rdf.liminalzone.org/Raccoon

Rich, E. (1979). User modeling via stereotypes. *Cognitive Science, 3,* 329-354.

Prometheus. (2005). *Prometheus methodology*. Retrieved from http://www.cs.rmit.edu.au/agents/prometheus/

Sołtysiak, S., & Crabtree, B. (1998). Automatic learning of user profiles—towards the personalization of agent service. *BT Technological Journal, 16*(3), 110-117.

Wooldridge, M. (2002). *An introduction to multiAgent systems.* John Wiley & Sons.

Wright, J., Gordon, M., Paprzycki, M., Williams, S., & Harrington, P. (2003). Using the ebXML registry repository to manage information in an Internet travel support system. In W. Abramowicz & G. Klein (Eds.), *Proceedings of the BIS2003 Conference* (pp. 81-89). Poland: Poznań University of Economics Press.

This work was previously published in Semantic Web Technologies and E-Business: Toward the Integrated Virtual Organization and Business Process Automation, edited by A. Salam & J. Stevens, pp. 325-359, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 4.9
Online Synchronous vs. Asynchronous Software Training Through the Behavioral Modeling Approach:
A Longitudinal Field Experiment

Charlie C. Chen
Appalachian State University, USA

R. S. Shaw
Tamkang University, Taiwan

ABSTRACT

The continued and increasing use of online training raises the question of whether the most effective training methods applied in live instruction will carry over to different online environments in the long run. Behavior Modeling (BM) approach – teaching through demonstration — has been proven as the most effective approach in a face-to-face (F2F) environment. A quasi-experiment was conducted with 96 undergraduate students who were taking a Microsoft SQL Server 2000 course in a university in Taiwan. The BM approach was employed in three learning environments — F2F, online synchronous and online asynchronous classes. The results were compared to see which produced the best performance, as measured by knowledge near-transfer and knowledge far-transfer effectiveness. Overall satisfaction with training was also measured. The results of the experiment indicate that during a long duration of training no significant difference in learning outcomes could be detected across the three learning environments.

CONCEPTUAL FOUNDATIONS

The Internet's proliferation creates a wealth of opportunities to deploy alternative online learning environments to facilitate many users in their learning processes. The information technology (IT) skills training market represented 76% of the entire online learning market in year 2000, according to a Jupiter Research report (CyberAtlas, 2003). The worldwide corporate online learning market may grow to $24 billion ($18 billion in the U.S.) by 2006 with a compound annual growth

rate of 35.6% (IDC, 2002). The burgeoning online learning/training market, and the increasing training budgets of businesses and schools has provided these key users of online training and marketing tools with practical reasons, as well as compelling research motives, to investigate the effectiveness of training and education in different online formats.

Online learning differs primarily from the traditional face-to-face (F2F) learning in that it is a user-centered, rather than instructor-centered, learning mode. Other benefits of substituting online learning for F2F learning include (1) self-paced instruction; (2) the ability to incorporate text, graphics, audio and video into the training; (3) opportunity for high levels of interactivity; (4) a written record of discussions and instructions; (5) low-cost operation; and (6) access to a world-wide audiences (Aniebonam, 2000). In addition, online learning can remove a certain degree of space and time limitations, speed up the learning process for motivated learners, lower economic costs of attending F2F classes and have higher information accessibility and availability.

Although IT has changed the training and educational approaches and environments, the ultimate goal of learning has not changed, that is, to transfer knowledge to students and allow them to apply the acquired knowledge in real situations. In the field of IT, the success of software training can be assessed with a trainee's IT skills of, and knowledge of the use of, particular software to solve problems. Surprisingly, after attending a training session, very few students know how to properly apply the acquired knowledge and skills to real situations. This raises an important issue, that is, how to improve knowledge transfer capability of learners in different online learning environments.

The importance of knowledge transfer is self-evident. However, the knowledge transfer process does not occur naturally. There is a need to assist learners in transferring their acquired knowledge into future applications. One effective approach to assisting the learning transfer process is "behavior modeling" (BM). This approach teaches learners through demonstration and hands-on experience. Simon, Grover, Teng, and Whitcomb (1996) and Compeau and Higgins (1995) found that in the field of information technology, BM is the most effective approach compared to the other two knowledge transfer approaches: exploration — teaching through practice on relevant example, and instruction — teaching software characteristics.

Distance education is defined as "teaching through the use of telecommunications technologies to transmit and receive various materials through voice, video and data" (Bielefield & Cheeseman, 1997, p. 141). In the same token, Leidner and Jarvenpaa (1995) define distance learning as "the transmission of a course from one location to another" (p. 274). These definitions provide an analogy to distance learning in the field of information technology or online software training. Online software training can be the transmission of instructional IT programming or contents to geographically dispersed individuals or groups.

There are two general modes of online learning: synchronous and asynchronous modes. Each mode can be marshaled with IT tools to deliver software training. Case in point, audio and video conferences are two types of online synchronous training mediums. Online asynchronous training mediums range from Web pages, file download, e-mail, e-mail list, newsgroup, forum, chat, response pad, whiteboard and to screen sharing. Built on his personal distance training and education experiences since 197 -+ 1, Horton (2000) suggests that online synchronous and asynchronous learning and training be designed for different purposes. Incorporating synchronous learning demands the control of schedule, time, people, class size, video and audio equipment and place. These factors constrain the possibility of reaching large numbers of students at any given time and in any given place.

However, the BM approach for trainees can be a problem in online asynchronous and synchronous training. For instance, any demonstration presented by a live instructor would need to be replaced with a scripted or videotaped demonstration in asynchronous mode, and live transmission or Webcam in synchronous mode. This raises several important questions. Can the scripted, videotaped, live transmission or Webcam approach still be as effective as the traditional classroom? How receptive are students to different online learning environments with differential degrees of student-centered interaction compared to an instructor-centered F2F environment? Most importantly, it is an unknown but interesting question to ask whether knowledge can be effectively transferred in different online environments. This research is to address these important issues faced by any instructor who intends to apply the BM approach in either online synchronous and asynchronous environments.

Behavioral Modeling and Knowledge Transfer

Social learning theory is the basis of the behavior modeling approach. Therefore, it is important to assess the applicability of the theory and approach in the online learning environment. Learning outcomes can be measured by different types of knowledge transfer and end-user satisfaction.

Behavior Modeling in Online Environments

Bandura (1977) proposed the Social Learning Theory to explain the interactive learning process between individuals and their social environment. He asserted a series of social learning needs take place to direct an individual from biological and self-centered response to social and group behaviors. Since the social learning process takes place within a society, individuals learn to establish their behavior models by observing and imitating other individuals' behaviors or through the enforcement of the media and environment. Online learning in different environments needs to be delivered via different media. Different online learning environments, therefore, may have different degrees of enforcement to learners' individual behaviors.

Learning by modeling or observing people's behaviors may be more effective than learning by trial-and-error because the former approach can avoid unnecessary mistakes and harm. Modeling an instructor's behaviors empowers students to (1) learn new behavior from the instructor, (2) self-evaluate their behaviors against the instructor's and (3) enforce students' current behavior.

Learning by modeling takes place in four sequential steps: (1) attention, (2) retention, (3) motor reproduction and (4) motivation and reinforcement (Bandura, 1977). Enforcement forces, such as the duration of training, praise, motivation and attention of others, allows learning to move along these four steps against counter forces. Enforcement forces, such as retention enhancement and practice, can contribute to better cognitive learning (Yi & Davis, 2001).

Lewin (1951) argued that the effectiveness of Behavior Modeling is a function of people interacting within an environment. The BM approach is different from learning by adaptation. The former approach teaches through demonstration, while the latter approach influences the behaviors of learners by reward and punishment (Skinner, 1938). The BM approach was first applied in the training of interpersonal communication and management skills (Decker & Nathan, 1985). Gist, Schwoerer and Rosen (1989) further applied the training to the context of information technology.

BM may be readily employed in face-to-face instruction, but cannot be easily simulated in online asynchronous instruction, which lacks the interactive immediacy necessary for optimally effective instructor demonstration and correction. The richness of information media in online

synchronous instruction is another constraint and may also have less enforcement force than F2F instruction to the learning outcomes. For example, in a live training class, the instructor is able to demonstrate a software process and immediately ask the students to repeat the activity under the instructor's close supervision. However, in an online asynchronous situation where there is no live instructor, the demonstration loses the benefit of that immediate feedback. In the same token, in an online synchronous situation bandwidth constraints and compromised reciprocity may undermine the enforcement force of the demonstration. In both online environments, enforcement forces can be further compromised with the missing of "learning by doing," another key element of F2F BM training (McGehee & Tullar, 1978).

Therefore, there is a strong possibility that the BM approach cannot be fully replicated in either the online synchronous or asynchronous situation and will not be as effective a method in online training as in the traditional environment.

Knowledge Transfer

Knowledge transfer is the application of acquired skills and knowledge into different situations. Unless the transferring process occurs, learning has little value. The applied situations could be similar or novel to the learning situation. Depending on the situation, knowledge transfer can take place in different formats. In general, there are four different types of knowledge transfer.

Positive Transfer vs. Negative Transfer

Positive transfer of learning means that learning in one situation stimulates and helps learning in another situation. Negative transfer of learning hinders the application of learning in one situation to other situations. Positive learning experience can be enhanced via analogy, informed instruction (Paris, Cross & Lipson, 1984), tutorial (Morris, Shaw & Perney, 1990) and so forth. Learning effectiveness can be improved by triggering positive learning and mitigating negative learning experience.

Near Transfer vs. Far Transfer

Salomon and Perkins (1988) argued that transfer of learning could have a differential degree of transfer. The effectiveness of near-transfer learning depends on the learner's ability to solve problems similar to those encountered in the learning context. For instance, learning how to add two digit numbers allows learners to add three digit numbers. Near-transfer learning occurs in two similar situations and at a lower level. Therefore, the level of learning is more easily acquired and applied. In contrast, applying the acquired skills and knowledge in two dissimilar and sometimes novel situations is much harder to achieve. For instance, a table tennis player can apply skills of playing pinball to playing tennis. Although both sports look similar on the surface, the techniques to control pinballs and tennis balls are very different. The learning transfer is much harder to be acquired and retained. Therefore, the transfer is defined as far-transfer learning. Near-transfer and far-transfer of knowledge seem to be the most widely used measures of learning outcomes in the field of information technology since learners must utilize the knowledge learned in a computing environment.

Specific Transfer and General Transfer

Depending on learning content, there are two different learning transfers: specific transfer and general transfer (Bruner, 1996). The former refers to the extension and association of habit and skills. The latter refers to the transfer of principles and attitudes that can be used to deepen the understanding of basic concepts.

Lateral Transfer and Vertical Transfer

Gagne (1992) asserted that the transfer of learning includes lateral and vertical transfers. Lateral learning is to apply one domain of knowledge to another domain. Lateral learning does not follow step-by-step instruction and is considered as provocative learning. Vertical learning means that a higher level of learning needs to be created by integrating acquired skills, and experiences with new situations. Vertical transfer of learning is analytical and sequential.

Other Knowledge Transfer Theories

Theories related to knowledge transfer are not limited to the above mentioned ones. For instance, the theory of identical elements asserts that the more identical elements different learning contains, the more efficient the transfer of learning (Thorndike, 1949). Baldwin and Ford (1988) proposed a general training theory to classify three categories of factors affecting transfer of training: (1) training inputs, (2) training outputs and (3) conditions of transfer. The situated learning theory argues that individuals are affected by learning environment when trying to solve practical problems. Therefore, the interaction between learners and the environment is an important factor that needs to be taken into account when measuring the transfer of learning. Finally, the theory of formal discipline argues that knowledge transfer skills can be acquired by training learner's sensuality, such as thinking, judgment, classification, imagination, creation and so forth.

The objective of this study was to investigate the impacts of the learning environment in online and offline formats on the transfer of learning. The situational changes rationalize the adoption of situated learning theory. To accomplish this objective, we sought to train end-user to learn how to use Microsoft SQL server 2000 software. Therefore, we adopted the near-transfer and far-transfer measures of learning outcomes for our information technology related experiment.

Hypotheses

Hypotheses are formulated to investigate whether the BM approach is as effective in online synchronous and asynchronous environments as in the traditional face-to-face environment. We measured learning outcomes by trainees' performances in near-transfer and far-transfer tasks, as well as overall satisfaction levels. The study also considered the importance of time variant. Hence, training and performance measurement were conducted over five weeks.

Knowledge Near-Transfer (KNT) Tasks

H1: End-users trained using F2F behavior modeling perform near-transfer information system tasks better than those trained in asynchronous behavior modeling.

H2: End-users trained in F2F behavior modeling perform near-transfer information system tasks better than those trained in synchronous behavior modeling.

H3: End-users trained in synchronous behavior modeling perform near-transfer information system tasks better than those trained in asynchronous behavior modeling.

Knowledge Far-Transfer (KFT) Tasks

H4: End-users trained in F2F behavior modeling perform far-transfer information system tasks better than those trained in asynchronous behavior modeling.

H5: End-users trained in F2F behavior modeling perform far-transfer information system tasks better than those trained in synchronous behavior modeling.

H6: End-users trained in synchronous behavior modeling perform far-transfer information

system tasks better than those trained in asynchronous behavior modeling.

Overall Satisfaction

H7: End-users trained in synchronous behavior modeling have a higher overall satisfaction level than those trained in asynchronous behavior modeling.

Research Design

This study applied Simon, Grover, Teng and Whitcomb's (1996) well-constructed software training theory to experimentally test behavior modeling training in three learning environments — F2F, online asynchronous and online synchronous environments. In doing so, it should be possible to detect the effects of the single independent variable (training environment) on training outcomes. The experiment was conducted in a field setting that enabled the study to garner greater external validity than would be the case with a laboratory experiment. A field experiment methodology has the merits of "testing theory" and "obtaining answers to practical questions" (Kerlinger & Lee, 2000). The exploratory nature of the study requires that variables (e.g., training environments and subject areas of study) under investigation be manipulated.

Subjects Control

The setting for the field experiment was the Tamkang University in Taiwan. The experiment was prompted by the need of 96 college sophomores, who are Management Information Systems (MIS) majors, to learn a Microsoft SQL Server 2000 software program in a database processing course. The schedule agreed on with the faculty at Tamkang University was to run the experiment for an hour training each week for four weeks. The author's graduate assistant Ms. Lin helped administer the experiment to collect the data. The subject pool had a mean age of 22 years. Subjects who participated in the structured experiment had little database-related experience. Their intellectual levels are relatively the same because subjects scored the same range of scores in a national entrance exam. The national entrance exam system has been adopted for more than 40 years in Taiwan and is considered a relatively reliable test. Subjects' individual backgrounds should not have influence on learning outcomes.

For the purposes of this study, subjects were chosen if they lacked a theoretical and procedural understanding of the particular subject area being tested. Participants were given a pretraining questionnaire that includes important study units on Microsoft SQL Server 2000. Two experts of the domain administered the Delphi study to finalize the study units and questionnaires. This is to improve the content validity. The subjects voluntarily answered whether they knew those study units and answered their database-related experiences. Based on their answers, a correlation test of database and usage experience of the target system showed no significant differences among three experimental groups. Subjects of the study may be considered representative of novice end-users. Many studies (Ahrens & Sankar 1993; Santhanam & Sein, 1994) support using students as experimental subjects to represent the general populations. Hence, all subjects' questionnaires were used for further data analysis. This segmentation was used to mitigate the effects of computer literacy and experience on the findings, thereby improving the internal validity of the study.

Training Treatments

Face-to-face BM (FBM) is instructor-centered training while online Asynchronous BM (ABM) and Synchronous BM (SBM) are learner-centered training. Course materials used in online learning environments were created to properly reflect the key elements of a behavior modeling approach. AniCam simulation software was used to record

the demonstration of instruction. Hyperlink structure was used to help users assimilate nonlateral conceptual, and procedural knowledge.

Feedback activities of behavior modeling approach in online asynchronous environment are supported with e-mail and hyperlinks. SBM differs from ABM in providing feedback functions via real-time discussion forums. Training materials integrate key elements of behavior modeling approach: (1) control of three different learning environments, (2) demonstration of the instructor, (3) continuous feedback (verbal feedback in F2F and online synchronous environments; e-mail feedback in the online asynchronous environment). Three training environments were designed to maximize the effect of size on their differences (Figure 1).

Training Procedures

The experimental study lasted for four weeks. There was a 50 minute training session each week for each class. Figure 2 shows the experimental procedures used at each time period. The X's, Y's and Z's represent online asynchronous BM training, online synchronous BM training and F2F BM training methods, respectively. The subscripts next to each alphabet indicate the ith observation or training session, respectively. Before executing experimental treatments (the pretest period O1), the instructor asked the subjects to complete a short questionnaire soliciting demographic information, database software-related experience and attitudes towards learning in the subject's assigned online learning environment (Pretest). Approximately one-third of the subjects pooled received the same experimental treatment for four straight weeks (Week1 to Week4). The assigning process was random on the class basis. Randomizing the execution of O4 and O5 in Week2 and Week3 for Group A and Group B can help avoid possible confounding results from the interactive effects of the pretest of O1 and O3. This randomization process can further ensure that difference in learning outcomes of O6 is not possibly due to the sensitization of the participants after the pretest and the interaction of their sensitization, O4 and O5 (Kerlinger & Lee, 2000).

Before or after each training session, subjects were asked to complete database design tasks using the MS SQL commands to assess their prior knowledge in the trained subjects and immediate learning outcomes that involve both near-transfer and far-transfer knowledge. On week five, students were evaluated again for their attitude changes towards the e-learning sessions and performance

Figure 1. Differences of behavior modeling approach in three learning modes

Online Learning Environments		**Off-line Learning Environment**
Asynchronous BM (ABM)	**Synchronous BM (SBM)**	**Face-to-Face BM (FBM)**
• Scripted demonstration of step-by-step instructions • Deductive/inductive complementary learning • Trainees choose one of two relevant examples to practice • Without online reference sources • Trainee control	• Webcam-delivered demonstration of step-by-step instructions • Deductive/inductive complementary learning • Instructor chooses examples that are relevant to trainees' majors • Without online reference sources • Trainer/trainee partially control	• Demonstration of a live instructor to learn step-by-step • Deductive/inductive complementary learning • Live instructor chooses examples that are relevant to trainees' majors • Without online reference sources • Trainer control

Figure 2. Experimental procedures

GROUP	Pretest	Week1	Week2	Week3	Week4	Post-test
Group A	O1	O2 X1 O3	X2	O4 X3 O5	X4	O6
Group B	O1	O2 Y1 O3	O4 Y2 O5	Y3	Y4	O6
Group C	O1	O2 Z1 O3	O4 Z2 O5	Z3	Z4	O6

Oi Questionnaire and Tests
Xi ABM (Online Asynchronous BM Training)
Yi SBM (Online Synchronous BM Training)
Zi FBM (F2F BM Training)

in near and far-transfer tasks (Post-test). The final exam concludes the five-week training sessions.

Training materials were designed to integrate key elements of the three training environments, as illustrated in Figure 3. Course materials used in the online asynchronous training session were stored on the school's server for students to learn at their own pace after each training session was completed. At the end of the experiment, students were asked about their affect for their learning environments.

Outcomes Measurement

Regardless of the teaching environment, computer training is intended to instill in users a level of competency in using the system and to improve their satisfaction with the system. A user's competency in using a system is contingent upon the user's knowledge absorption capacity. Ramsden (1988) finds that effective teaching needs to align students with situations where they are encouraged to think deeper and more holistically. Kirkpatrick (1967) also suggests that learning effectiveness needs to be evaluated by students' reactions, learning and knowledge transfer. The levels of knowledge absorbed by students, Bayman and Mayer (1988) suggest, may include syntactic, semantic, schematic and strategic knowledge. Mennecke, Crossland and Killingsworth (2000) believe that experts of one particular knowledge domain possess more strategic and semantic knowledge than novices. Knowledge levels, as Simon, Grover, Teng and Whitcomb (1996) suggest, can be categorized as near-transfer, far-transfer or problem solving. Near-transfer knowledge is necessary for being able to understand software commands and procedures. This type of knowledge is important for a trainee to be able to use software in a step-by-step fashion. Far-transfer knowledge seeks to ensure that a trainee has the ability to combine two or more near-transfer tasks to solve more complicated problems.

Both the use of software and information systems and the satisfaction levels of using them are useful surrogates to properly measure the effectiveness of an information system (Ives, Olson, & Baroudi, 1983). The end-user satisfaction level has been widely adopted as an important factor contributing to the success of end-user software training. Since the study was to replicate Simon, Grover, Teng and Whitcomb's (1996) research in a dissimilar environment, near-knowledge and far-knowledge transfer, and end-user overall satisfaction levels were adopted in this study to measure training outcomes. Cronbach's alpha reliability for Simon et al.'s (1996) instrument to measure satisfaction is r = 0.98. Users need to use the Likert scale from one to five to answer 12 test items related to their satisfaction with the use of online system.

Data Analysis

Table 1 shows the means and standard deviations for the scores at each treatment period. Table 2

Figure 3. Delivery mechanisms of behavior modeling approaches

	FBM (F2F Behavior Modeling)	**ABM (Asynchronous Behavior Modeling)**	**SBM (Synchronous Behavior Modeling)**
Course Materials	Instructor demonstrates the use of software along with PowerPoint slides Covered three study subjects within forty five minutes each week	Course materials covered by FBM was pre-recorded and stored in a server. No instructor was present to assist the learning process of students. Students learned at their own path and completed their study within forty five minutes.	Instructor was present, but broadcasted steaming video from a broadcast room. Instructor conducted the real-time discussion with students on a BBS station.
Information Systems Tools	Instructor, PowerPoint, and PC	AniCam, PowerPoint and Acrobat Reader	AniCam, PowerPoint, Stream Author v.2.5 (Authoring Tool) and Acrobat Reader
Target System	SQL Server2000 Personal Edition	SQL Server2000 Personal Edition	SQL Server2000 Personal Edition
Pretest Questionnaire	Learning Experience and Style Questionnaires	Learning Experience and Style Questionnaires	Learning Experience and Style Questionnaires
The First Week	First Training Session First Learning Outcomes Test	First Training Session First Learning Outcomes Test	First Training Session First Learning Outcomes Test
The Second Week	Second Training Session Second Learning Outcomes Test	Second Training Session	Second Training Session Second Learning Outcomes Test
The Third Week	Third Training Session	Third Training Session Second Learning Outcomes Test	Third Training Session
The Fourth Week	Comprehensive Test (Third Learning Outcomes Test)	Comprehensive Test (Third Learning Outcomes Test)	Comprehensive Test (Third Learning Outcomes Test)
Post-test Questionnaire	Measure End-User Satisfaction	Measure End-User Satisfaction	Measure End-User Satisfaction

shows F and P values of the dependent variables (near-transfer and far-transfer task performances, and overall satisfaction) across treatment groups and in different times. Pretest scores (Q1, Q2 and Q4) in varying weeks were used to tell apart students with prior experiences and knowledge on the studied topics. After learning in a weekly session, students participated in a post-test. Their scores (Q3 and Q5) were used for KNT effectiveness comparison across training sessions. Scores of Q6 are KFT effectiveness and end-user satisfaction levels. A cursory examination of means (Table 1) indicates that no patterns can be identified for near-transfer performance from time Week1 to Week5. Subjects in ABM performed better than those in FBM, followed by SBM at Week1 while at Week2 and Week3 the order was changed to FBM>ABM>SBM and SBM>FBM>ABM, respectively. The findings are not in agreement with a consistent pattern as predicted by Hypotheses H1 and H2. For KFT tasks, subjects in ABM performed better than those in SBM, followed by FBM. This is the reversed order of a pattern as predicted by Hypotheses H3 and H4. The measurement of overall satisfaction level somewhat follows the predicted patterns of Hypotheses H5 and H6.

Table 1. Descriptive statistics - means (standard deviations)

	ABM (N=40)	SBM (N=26)	FBM (N=30)	Overall (N=96)
KNT (Week 1)	27.63 (5.77)	25.38 (7.06)	26.00 (7.24)	26.51 (6.61)
KNT (Week 2)	28.50 (3.43)	28.46 (3.09)	29.83 (0.91)	28.91 (2.83)
KNT (Week 3)	56.05 (14.06)	64.27 (17.52)	62.27 (12.71)	60.22 (14.98)
KNT (Week 5)	71.70 (16.21)	70.50 (19.78)	67.00 (17.27)	69.91 (17.49)
KFT (Post-test)	9.63 (1.33)	9.42 (1.63)	8.83 (2.15)	9.32 (1.72)
OS (Post-test)	38.10 (6.87)	39.38 (9.21)		38.61 (7.83)

Table 2. Performance on differentlLearning outcomes over five weeks

	F	p-value	Power
KNT (Week 1)	1.035	0.359	0.337
KNT (Week 2)	2.415	0.095*	0.605
KNT (Week 3)	2.891	0.061*	0.677
KNT (Post-test)	0.634	0.532	0.246
KFT (Post-test)	1.913	0.153	0.517
OS (Post-test)	0.420	0.519	0.169

We took a closer look at the mean difference at the significance level of 0.05. The study used one-way ANCOVA to analyze the effects of behavior modeling approach on learning outcomes over different time. Levene's Test (1960) was used to examine the variance homogeneity of three groups. Its F-statistics showed that KNT was 3.04 (p=0.053) at Week1, 13.01 (p=0.000) at Week2, 1.71 (p=0.187) at Week3, and 0.47 (p=0.627) at the Post-test. In contrast, the F-statistics of Levene's Test for KFT and OS were 7.64 (p=0.001) and 1.75 (p=0.191) at the Post-test. With the exception of KNT at Week2 and KNF at the Post-test, all dependent variables met the $p > 0.05$ criterion for assuring homogeneity of variances. The heteroscedasticity of variances for these two exceptions suggested that the statistical test results may not be valid. As such, the following discussion will ignore these two variances and focus on KNT and OS. For other effects that show significance, the study adopts the Scheffe post-test to analyze data. In addition, Pearson Correlation Analysis was used to assess the carry-over effects of different training sessions.

ANCOVA was performed using the general linear model approach; the results are presented in Table 3. It shows that the treatment effects are significant for KNT (Week2) and KNT (Week3) with F-statistics of 2.415 (p=0.095) and 2.891 (p=0.061), confirming a univariate treatment effect of learning environments on the dependent variable: KNT. However, the treatment effects are not salient for other dependent variables: KFT and OS. These lacks of effect may have been due to small effect sizes.

Least-Squares Deconvolution (LSD) was used to test cross-correlations for KNT (Week2) and KNT (Week3). LSD is a cross-correlation technique for computing average profiles. LSD is very similar to most other cross-correlation techniques, though slightly more sophisticated in the sense that it cleans the crosscorrelation profile from the autocorrelation profile (Donati, 2003). For KNT (Week 2), the LSD results indicate that subjects in FBM perform better than

Table 3. Results for training methods

Variable	*Hypothesis*	*Result in Correct Direction?*	*Significant p-value? (n.s. - not significant)*
Week1	H1: FBM > ABM	F	n.s. (p=0.312)
	H2: FBM > SBM	T	n.s. (p=0.729)
	H3: SBM > ABM	F	n.s. (p=0.182)
Week2	H1: FBM > ABM	T	p=0.051
	H2: FBM > SBM	T	p = 0.069
	H3: SBM > ABM	T	n.s. (p=0.956)
Week3	H1: FBM > ABM	T	p=0.083
	H2: FBM > SBM	F	n.s. (p=0.612)
	H3: SBM > ABM	T	p = 0.029
Week4	H1: FBM > ABM	F	n.s. (p=2.71)
	H2: FBM > SBM	F	n.s. (p=0.459)
	H3: SBM > ABM	F	n.s. (p=0.787)
Post Test (KFT)	H4: FBM > ABM	F	p=0.057
	H5: FBM > SBM	F	n.s. (p=0.2)
	H6: SBM > ABM	F	n.s. (p=0.639)
Post Test (OS)	H7: SBM > ABM	F	n.s. (p=0.579)

those in ABM (p=0.051) and SBM (p=0.069). This supported the Hypotheses 1 and 2. However, Hypothesis 3 cannot be supported because the mean difference between ABM and SBM is not significant. For KNT (Week3), the LSD results indicate that (1) subjects in FBM performed better than those in ABM (p=0.083), and (2) subjects in SBM performed better than those in ABM (p=0.029). Hypotheses 4 and 6 are supported. Worthy to be noted is that H4 is upheld but in the reversed direction. This indicates that ABM is a more effective method than FBM at improving knowledge far transfer.

Four out of nine hypotheses in total are supported. Although not all hypothesized relationships are fully supported, the results obtained are interesting. The most intriguing result is that although there is statistically-justified reason for preferring FBM to ABM or SBM or software training, the pattern of results is not persistent in the long run. FBM resulted in better outcomes than ABM and SBM at Week2, and than ABM at Week3 for KNT. Although it never does so at a statistically significant level, subjects in ABM performed better than those in SBM, followed by those in FBM for KNT (Post-test) and KFT (Post-test). One interpretation of this is that either

ABM or SBM training is no worse than FBM training across all dependent variables. The pattern of results for FBM suggests that trainers might choose ABM or SBM, which should to be a less costly alternative to FBM, without making any significant sacrifices in either learning or trainee reaction outcomes.

Another result of interest is that, with respect to the three online asynchronous training methods, the pattern of results suggests that FBM might be the best for KNT in the short term. Of the nine hypotheses concerning relationships between these methods, four are in the expected direction, and significantly so. This indicates that use of ABM or SBM may be a better – and certainly no worse – software training strategy in the long term.

Implications for Research

This article studied the impact of training duration on performance and trainee reactions. Trainees were exposed to the same training methods with different degrees of social presence for different durations. These findings indicated that training duration and social presence have little impacts on learning outcomes. Despite this, the findings here raise additional questions for research.

It may be more important to investigate the impacts of information richness (Fulk, 1993) features of online training media on training outcomes. Future studies might vary the social presence features of training media or their combination with social presence features (e.g., with instructor's feedbacks versus discussion boards, e-mail response or playback features). Information richness may be a more influential factor affecting the performance of training approaches.

It may also be useful to replicate the experimental equivalence of FBM, ABM and SBM methods of software training with different software and subjects. Since in the long term different treatments have similar impacts on learning outcomes, it may be practical to demonstrate the cost-based advantage of ABM over SBM, and SBM over FBM for software training in practical settings.

Another way to improve the reliability of the study is to manipulate some useful blocking variables. A series of comparative studies can be conducted to assess the impact of individualism as a cultural characteristic, computer self-efficacy, task complexity (simple tasks vs. fuzzy tasks), professional backgrounds and the ratio of the training duration to the quantity of information to be processed, among others.

Learning style may be an important factor to consider in the online learning environment. According to social learning theory, learners interact with the learning environment to change their behavior. Learning style is situational and can vary with different learning environments. Therefore, it is possible that the combination of training methods, learning style and social presence information richness (SPIR) attributes may jointly determine learning outcomes. This is not the case for BM approach in F2F environment. The self-paced online learning environment may alter the assertion. Hence, it may be necessary to conduct longitudinal studies of the influence of learning style on learning performance and trainee reaction.

Implications for Practice

The largest implication for practice is that ABM and SBM may provide cost-effective substitutes for FBM without significant reductions in training outcomes in the long term. While it may still be true that FBM is still the most effective approach to improve KNT in the short term, ABM and SBM have similar leverage in KFT in the short term and KNT in the long term. Regardless of training environments, trainees have same satisfaction levels in the near- and long-term. These findings strongly indicate that the cost issue is more important than learning effectiveness. When given the options to decide which BM approach to take in the long term, nonperformance issues (teacher

and facility availability, trainee's preferences, location and convenience issues) have to be first taken into account.

CONCLUSION

The success of an online training strategy depends on its effectiveness in improving learning outcomes. This study, built on well-accepted frameworks for training research (Bostrom, Olfman & Sein, 1990; Simon & Werner 1996), examines the relative effectiveness of the behavior modeling approach in online synchronous, online asynchronous and face-to-face environments. The results from this experiment provide an empirical basis for the development of an online behavior modeling strategy: (1) FBM is more effective than ABM and SBM for knowledge transfer in the short term (KNT), and (2) ABM and SBM are as effective as FBM for knowledge transfer and overall satisfaction in the long term (KFT).

What is learned from this study can be summarized as follows: When conducting software training, it may be almost as effective to use online training (synchronous or asynchronous) as it is to use a more costly face-to-face training in the long term. In the short term face-to-face knowledge transfer model still seems to be the most effective approach to improve knowledge transfer in the short term.

The limitation of this experimental study is that it was conducted with a homogeneous group with Taiwanese cultural and educational backgrounds. Therefore, this study may be constrained with the generalizability of its findings to different cultural contexts.

Hofstede (1997) stated that the domains of education, management and organization have nurtured the values context that differs from one country to another. Cultural influences have been discerned in the study of Internet usage (Lederer, Maupin, Sena & Zhuang, 2000; Moon & Kim, 2001; Straub, 1997) and Web site design (Chu, 1999; Svastisinha, 1999). Users from different cultures have different perceptions about the usefulness and ease of use regarding different information systems (Straub, 1994). E-learning systems may differ based on the cultural backgrounds of the learners to improve their satisfaction levels and cognitive gains. Benefits of the congruence may include the improvement of (1) global e-learning adoption rate and (2) learning outcomes (attitude and cognitive gains). From the perspective of research design (Kerlinger & Lee, 2000), a cross-cultural study to replicate the study with American or European subjects may further validate and extend the generalizability of the findings.

The study has accomplished its major goal; it provides evidence as to the relative effectiveness of the behavior modeling approach in different learning environments for software training. This research somewhat improves the generalizability of theories on the behavior modeling approach in different learning environments.

REFERENCES

Ahrens, J. D., & Sankar, C. S. (1993). Tailoring database training for end users. *MIS Quarterly, 17*(4), 419-439.

Aniebonam, M. C. (2000, October). Effective distance learning methods as a curriculum delivery tool in diverse university environments: The case of traditional vs. historically black colleges and universities. *Communications of the Association for Information Systems, 4*(8), 1-35.

Baldwin, T.T., & Ford, J.K. (1988). Transfer of training: A review and directions for future research. *Personnel Psychology, 41,* 63-105.

Bandura, A. (1977). *Social learning theory.* Morristown, NJ: General Learning Press.

Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer pro-

gramming. *Journal of Educational Psychology, 80*(3), 291-298.

Bielefield, A., & Cheeseman, L. (1997). *Technology and copyright law.* New York: Neal-Schuman Publishers, Inc.

Bostrom, R. P., Olfman, L., & Sein, M. K. (1990). The importance of learning style in end-user training. *MIS Quarterly, 14*(1), 101-109.

Bruner, J. (1996). *Toward a theory of instruction*. New York: Norton.

Chu, G.-L. (1999). *The relationships between cultural differences among American and Chinese university students and the design of personal pages on the World Wide Web.* Unpublished doctoral dissertation, University of Georgia.

Compeau, D. R., & Higgins, C. A. (1995). Application of social cognitive theory to training for computer skills. *Information Systems Research, 6*(2), 118-143.

CyberAtlas. (2003). E-Learning market expanding beyond IT training. Jupiter Research. Retrieve July 28, 2006, from *http://cyberatlas.internet.com/markets/education/article/0,,5951_914901,00.html*

Decker, P. J., & Nathan, B. R. (1985). *Behavior modeling training.* New York: Praeger.

Donati, J. (2003). Least-squares deconvolution of Stellar Spectra. Retrieved July 28, 2006, from http://webast.ast.obs-mip.fr/people/donati/multi.html

Fulk, J. (1993). Social construction of communication technology. *Academy of Management Journal, 36*, 921-950.

Gagne, R. M. (1992). *Principles of instructional design*. New York: Holt, Rinehart and Winston, Inc.

Gist, M. E., Schwoerer, C., & Rosen, B. (1989). Effects of alternative training methods on self-efficacy and performance in computer software training. *Journal of Applied Psychology, 74*, 884-891.

Hofstede, G. (1997) *Cultures and organizations: Software of the mind*. New York: McGraw-Hill.

Horton, W. (2000). *Designing Web-based training: How to teach anyone anything anywhere anytime*. New York: John Wiley & Sons.

IDC. (2002, September 30). While corporate training markets will not live up to earlier forecasts, IDC suggests reasons for optimism, particularly e-learning. Retrieved July 28, 2006, from *http://www.idc.com/getdoc.jhtml?containerId=pr2002_09_17_150550*

Ives, B., Olson, M., & Baroudi, S. (1983). The measurement of user information satisfaction. *Communications of the ACM, 26*, 785-793.

Kerlinger, F. N., & Lee, H. B. (2000). *Foundations of behavioral research.* New York: Harcourt Brace College Publishers.

Kirpatrick, D. L. (Ed.). (1967). *Evaluation of training*: *Training and development handbook*. New York: McGraw-Hill.

Lederer, A. L., Maupin, D. J., Maupin, M. P., Sena, M.P. & Zhuang, Y. (2000). The technology acceptance model and the World Wide Web. *Decision Support Systems, 29*, 269-282.

Leidner, D. E., & Jarvenpaa, S. L. (1995). The use of information technology to enhance management school education: A theoretical view. *MIS Quarterly, 19*, 265-291.

Levene, H. (1960). In I. Olkin et al. (Eds.) *Contributions to probability and statistics: Essays in honor of Harold Hotelling.* (pp. 278-292). Stanford University Press.

Lewin, K. (1951). *Field theory in social science: Selected theoretical papers.* New York: Harper and Row.

McGehee, W., & Tullar, W. (1978). A note on evaluating behavior modification and behavior modeling as industrial training techniques. *Personal Psychology, 31,* 477-484.

Mennecke, B. E., Crossland, M. D., & Killingsworth, B. L. (2000). Is a map more than a picture? The role of SDSS technology, subject characteristics, and problem complexity on map reading and problem solving. *MIS Quarterly, 24*(4), 601-627.

Moon, J., & Kim, Y. (2001). Extending the TAM for a World Wide Web context. *Information & Management, 38,* 217-230.

Morris, D., Shaw, B., & Perney, J. (1990). Helping low readers in grades 2 and 3: An after-school volunteer tutoring program. *The Elementary School Journal, 91,* 133-150.

Paris, S. G., Cross, D. R., & Lipson, M. Y. (1984). Informed strategies for learning: A program to improve children's reading awareness and comprehension. *Journal of Educational Psychology, 7,* 1239-1252.

Ramsden, P. (Ed.). (1988). Context and strategy: Situational influences on learning. In *Learning strategies and learning styles.* New York: Plenum Press.

Salomon, G., & Perkins, D. N. (1988). Teaching for transfer. *Educational Leadership, 46*(1), 22-35.

Santhanam, R., & Sein, M. K. (1994). Improving end-user proficiency: Effects of conceptual training and nature of interaction. *Information Systems Research, 5*(4), 378-399.

Simon, S. J., Grover, V., Teng, J. T. C., & Whitcomb, K. (1996). The relationship of information system training methods and cognitive ability to end-user satisfaction, comprehension, and skill transfer: A longitudinal field study. *Information Systems Research, 7*(4), 466-490.

Simon, S. J., & Werner, J. M. (1996). Computer training through behavior modeling, self-paced, and instructional approaches: A field experiment. *Journal of Applied Psychology, 81*(6), 648-659.

Skinner, B. F. (1938). *The behavior of organisms: An experimental analysis.* New York: Appleton-Century Company, Incorporated.

Straub, D. W. (1994). The effect of culture on IT diffusion: E-mail and FAX in Japan and the U.S. *Information Systems Research, 5*(1), 23-47.

Straub, D., Keil, M., & Brenner, W. (1997). Testing the technology acceptance model across cultures: A three country study. *Information and Management, 33,* 1-11.

Svastisinha, R. W. (1999). *Wahhn: Web-based design. Wind and human comfort for Thailand.* Unpublished doctoral dissertation, University of Southern California.

Thorndike, R. L. (1949). *Personnel selection: Test and measurement techniques.* New York: John Wiley & Sons.

Yi, M. Y., & Davis, F. D. (2001). Improving computer training effectiveness for decision technologies: Behavior modeling and retention enhancement. *Decision Sciences, 32*(3), 521-544.

This work was previously published in International Journal of Distance Education Technologies, Vol. 4, Issue 4, edited by S. Chang, and T. Shih, pp. 88-102, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 4.10
Rapid Insertion of Leading Edge Industrial Strength Software into University Classrooms

Dick B. Simmons
Texas A&M University, USA

William Lively
Texas A&M University, USA

Chris Nelson
IBM Corporation, USA

Joseph E. Urban
Arizona State University, USA

ABSTRACT

Within the United States, the greatest job growth is in software engineering and information management. Open source software (OSS) is a major technology base for enterprise application development. The complexity of technologies used by industry is often an obstacle to their use in the classroom. In this chapter, a major software development paradigm change that occurred in about the year 2000 is explained. CS education programs have been slow to adapt to the paradigm change due to problems such as the tenure system, inexperienced student laboratory assistants, lack of leading-edge software tool support, lack of software team project servers, unavailability of help and mentoring services, and software unavailability. This chapter explains how these problems can be solved by creating an open source-based shared software infrastructure program (SSIP) sponsored by industry, but planned and implemented by SSIP member universities at no cost to member universities.

INTRODUCTION

Today students are saying no to computer science (Frauenheim, 2004). CS faculty members have panicked in what David Patterson (2005) calls Chicken Little rumor mongering. He tells everyone to stop whining about outsourcing. In our opinion, CS faculty should panic and adapt to a new software development paradigm. Pat-

terson makes an invalid implied assumption for his article in that CS in some way is related to information technology (IT) jobs in U.S. industry (or, for that matter, that CS is useful to a software engineer). He is correct to say that U.S. IT jobs are increasing. Also, software engineering (SE) degree programs and jobs are increasing. His domino theory of job migration is not correct. We agree with Patterson that U.S. programmers should worry about both India and China. We do not agree that either India or China will have to worry much about the Czech Republic. Both India and China have such large populations and low wages that major CS job migration will mainly be to these two countries. The middle processes of a software product development software life cycle (DSLC) may completely migrate from the United States.

Every Fortune 1000 company with which we are familiar takes advantage of low labor costs in India and/or China. Unfortunately for CS, approximately 80% of high-paying CS jobs in the past have been with Fortune 1000 companies. Jobs that will remain in the United States will go to students that are familiar with open standards, a wide variety of solutions including open source solutions, software development tools that support open standard visualization design models and open source integrated development environments. In this chapter, open standards will be defined as standards that are publicly available. The Object Management Group (OMG) (2006) is an example of an organization that was created to produce open standards. OMG is an open membership, not-for-profit consortium that produces and maintains computer industry open standards for interoperable enterprise applications. OMG membership includes virtually every large company in the computer industry and hundreds of smaller ones. OMG's most widely used standard is described by the unified modeling language (UML) specification. UML is used worldwide to model application structure, behavior, architecture, business process, and data structure. We use the term open source software (OSS) to refer to software that has Open Source Initiative (OSI) (2006) licenses. Examples of OSS are Linux, Apache, Eclipse, and Derby. We also include open-standard compliant software that is provided free for classroom use to universities. An example is IBM Rational Software Architect (RSA).

The objective of this chapter is to explain how leading-edge industrial-strength software can be introduced into the university classroom by using OSS, open standards, distance learning, and infrastructure shared among cooperating universities. In this chapter, we will describe the evolution of software development during the 20th century, the paradigm change at the beginning of the 21st century, and the problems with existing university information technology education. Then we will describe a shared software infrastructure program (SSIP) to rapidly introduce leading-edge industrial software solutions into university classrooms at no cost to SSIP member universities.

BACKGROUND

Software education emerged during the last 50 years of the 20th century. During the mid-1900s, computers were applied to create firing tables for the military. Scientists programmed these computers using computational algorithms. Computer memories were small and expensive, and successful software depended on efficient algorithms. As computer use grew, universities began to offer programming courses based on algorithm methodology. The application of mathematical science of algorithms to computers led to a new field called computer science. As demand for computer programmers grew, computer science programs at U.S. universities grew in number. U.S. universities had the computers, while universities outside the United States and Europe did not have access to computers. As the size and complexity of computer systems continued to grow, one could not rely on the theory of algorithms to provide

acceptable solutions. At a 1968 NATO conference in Europe (Naur & Randell, 1968), computer professionals realized that the software for major systems would have to be engineered based on engineering science and practice. That is when the term *software engineering* was introduced. In 1984, the U.S. Department of Defense created a Software Engineering Institute at Carnegie Mellon University (2006) to advance the practice of software engineering.

Throughout the 1990s, the cost of computers continued to decline, and the capabilities of computers increased. Computer cost was no longer a barrier to the spread of computer-related education programs to universities throughout the world. This has been aided by the creation and expansion of the World Wide Web (WWW) over the Internet.

Education Programs at the Beginning of the 21st Century

Computer-related educational programs at the beginning of the 21st century fall under the umbrella term information technology, which includes computer science, computer engineering, information management, and software engineering. Overall, the demand for information technology knowledge workers worldwide is increasing. The U.S. information technology education programs hit hardest by use of off-shore contractors are the science-based computer science programs. Enrollment in U.S. computer science programs is on the decline. These programs emphasize the middle or coding process of the DSLC. The coding process is the easiest to outsource from high labor cost regions to off-shore low-labor cost regions. All indications are that the computer science down trend will continue. Demand for computer engineering graduates remains strong. The fastest growing demand is for information management and software engineering graduates. Software engineering and information management programs teach students about all phases of the DSLC. The U.S. Department of Labor Statistics (2005) projects software engineers to be one of the fastest growing occupations through at least 2014. Major universities are beginning to offer software engineering certificates, bachelor degrees, and master degrees.

At the beginning of the new millennium, the outlook for software engineers is strong. Distance learning technology is becoming common. Software development is being practiced with project members distributed around the globe. Open source, open standards, and interoperable software are being demanded by customers. Software knowledge will continue to change and expand at a very rapid rate.

MAIN FOCUS OF THE CHAPTER

University Environments

At present, all students that come to the university are computer literate and have their own computers. Many already know how to program and are connected to the Internet. They are looking for software knowledge that will qualify them to find jobs in which they can create complex software products. They are not finding this software knowledge in most computer science (CS) departments.

We now return to David Patterson's problems. As the president of the Association for Computing Machinery (ACM), he has to be a cheerleader for CS faculty who run around yelling that the sky is falling. For most of the CS degree programs, the sky is falling. Many of the CS degree programs are in small liberal arts colleges and former teacher colleges. These schools do not have the technical background or faculty required to give students the software knowledge that will be provided by the increasing number of IT and SE degree programs. As CS job demands decline, CS programs will continue to shrink with the faculty having to face problems of dying programs unless they

adopt new methodologies to quickly introduce into classrooms the latest software knowledge required by industry

Woodie Flowers (2000), a mechanical engineering professor at MIT, recently asked, "Why should education change, we have been doing it this way for 4,000 years?" He said that over the next decade, educators will have to restructure their curricula in order to accommodate the World Wide Web. Change in the university moves at glacial speed. Software knowledge is continually expanding and growing much faster than the current education process can adapt to in order to meet the needs of industry. Ways must be found to upgrade software knowledge that is taught to IT professionals graduating from universities today.

The problem with education of the IT professional can be traced back to the first programmers. The first IT professionals were scientists and engineers who knew how to build and operate the first computers. Many programmers were mathematicians. When universities began to use computers in the 1950s and 1960s, engineering schools emphasized teaching hardware, science schools emphasized teaching programming languages, and business schools emphasized business applications. The science schools originally placed programming languages courses in mathematics departments. As time progressed, mathematicians teaching programming languages separated from the mathematics departments to create CS departments. Instead of teaching the latest software knowledge, the mathematicians began to teach what they knew best: computational mathematics and theory of algorithms. An algorithm is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation (*Webster's new Collegiate Dictionary,* 1981). It can be shown that for any problems other than toy problems, it is impossible to prove that complex software products terminate in a finite number of steps. Thus, the time spent teaching theory of algorithms is time wasted. Tenured faculty members hired to teach mathematical algorithms will probably continue teaching algorithms until they retire. Since faculty members decide which young faculty members are hired and later tenured, they will probably continue to hire computational mathematics and algorithm specialists.

During the 1960s and 1970s, computers came into common use in industry. People with almost any background could be trained to operate the computer applications in industry. With the advent of the personal computer in the 1980s, the people familiar with computers continued to expand. With the commercialization of the Internet in the 1990s and the introduction of Internet browsers and computer games, virtually everyone below middle age used computers. Essentially every high school graduate that enters a university today has a computer that can be connected to the Internet. They learn how to program in high school and are usually familiar with some form of database management system. They learn how to access Internet servers through the use of browsers. If they would like to become an IT professional, they expect to learn the latest software knowledge that industry demands. CS departments that hire specialists in computational mathematics, theory of algorithms, and computational complexity theory will continue to lose students. Unless they change, CS departments probably will be absorbed eventually back into mathematics departments. By working with industry, university IT programs can teach the latest software knowledge to their students who will then be in high demand when they seek jobs in industry.

Year 2000 Productivity Paradigm Change

During the latter part of the 20th century, the demand for U.S. software developers continued to exceed the supply. During the 1950s, 1960s, 1970s, and 1980s, computer hardware was expensive and many developing countries could not

afford computers. Thus, if you wanted to become an IT professional, you almost had to study in the United States. As a result, virtually every university created CS degree programs that were advertised as the correct degree program for the IT professional. The result is a huge oversupply of PhD and Master's degree graduates who learned mathematics theory but obtained very little software knowledge. During the 1990s, computer costs continued to decline to where students and universities even in the poorest countries could afford computers. The *Wall Street Journal* pointed out that the auto worker salary in Germany was $33 per hour, while an auto worker in China earned $0.98 cents per hour. The salary differential for knowledge workers such as software developers is similar. Leading up to the year 2000 was the conversion of all legacy software in the United States to handle a four-digit year instead of a two-digit year built into existing software products. There were not enough experienced programmers in the United States to handle the demand for COBOL programmers. Companies turned to the software houses in India to help with the conversion. The large Fortune 1000 companies were very pleased with the results, and after 2000, they began to out source computer coding to offshore companies in India. Recently, the Chinese commercial software industry, although lagging behind India's, has been undergoing major structural shifts that could make it the Asian industry leader (Kshetri, 2005). Chinese developers are making a major commitment to OSS. Large U.S. companies are out-sourcing the middle processes of the DSLC, while the upstream requirements elicitation, requirements specification, and software architecture processes and downstream acceptance testing and software product installation will remain in the United States. CS educational programs emphasize the middle DSLC processes, while SE and information management programs emphasize the upstream and downstream DSLC processes. While the demand for IT workers in the United States is increasing, the demand for CS professionals is decreasing.

As mentioned earlier, CS faculty members have begun to panic and grasp for schemes to restore CS popularity. Former ACM President David Patterson (2005) suggests expanding student recruiting in high schools by ACM's new CS Teachers Association. He recognizes that software knowledge continually changes and places emphasis on keeping job skills up to date. Former ACM President Peter Deming along with Andrew McGettrick (Denning & McGettrick, 2005) point out that CS places too much emphasis on coding and not enough emphasis on other DSLC processes, including the use of advanced software tools to support requirements gathering, defect tracking, configuration management, middleware services, advanced software solutions, and software process visualization tools. They recognize that emphasis placed on analysis of algorithms and complexity theory as the heart and soul of computing is a mistake. Their solution is wrong. Recruiting of students is not the solution to declining enrollments. What is needed is emphasis on software knowledge that today's computer professional needs in order to be competitive in the global marketplace. A solution must be found to overcome the current problems with CS programs.

CS Education Program Problems

Major problems that must be corrected in order for CS graduates to be attractive to employers include tenure system, inexperienced laboratory assistants, software tool support, software team project servers, inadequate department support personnel, help and mentoring services, and available software.

The university tenure system is a type of union for university faculty. It is almost impossible to remove a tenured faculty member once tenure has been granted. Tenure empowers faculty but does not make them accountable. Tenured CS faculty

members who are specialists in computational mathematics and algorithms will remain faculty members for approximate 30 years between the time tenure is granted and retirement. In most cases, they will continue to teach computational mathematics and algorithm theory even though there is very little industrial or student interest in these areas. Also, these subject areas are unnecessary for an understanding of the software knowledge required by the IT professional.

In universities with major research programs, many of the funded graduate student teaching assistants that oversee laboratories for software team projects have never used any advanced software tools used in industry to create software products. Even though companies may provide these tools to universities at no cost, the laboratory assistants must understand them and be able to help and mentor student teams in project courses.

Many software products can be used directly out of the box. Software users expect to be able to load a new software system and then begin to immediately start using the system. Heavy-duty software tools are not out-of-the-box. To set up a software tool environment, a software tool administrator must create directories and security as well as initialize parameters in which the tools in an environment work together. The typical CS department does not have enough software support staff or funds to hire additional staff to administer a suite of advanced tools.

Student teams working on a capstone project to create a software product must have access to a server for testing the software product. Often student projects crash servers during testing and interfere with other people trying to use that same server. Software projects need a server as a type of sand box for operating their software product. Normally, CS departments do not have the resources to dedicate servers to student projects.

Many software vendors provide free training and use of software tools to universities. But they provide minimal help facilities to answer specific questions that arise while trying to use the tools. Students must be able to contact knowledgeable people who will answer their questions in a timely manner. When students are learning to use complex software design and testing tools, they would like access to a mentor to guide them. Ideally, help and mentoring services should be available 24 hours a day, seven days a week (24/7).

As part of students gaining software knowledge required by industry, students must have easy access to software products and tools. Many vendors will provide free software and licenses to universities. Acquiring the software and licenses to use the software can be a problem when universities are not paying for the software. There needs to be a service to expedite the process of acquiring software for classroom use at no cost to universities.

By helping universities to quickly introduce best software development practices, improved processes, and advanced software tools, students that graduate from these programs will gain software knowledge that is in high demand by industry. The goal of the SSIP is to set up an infrastructure shared among universities in which universities can easily introduce the latest leading software knowledge into both undergraduate and graduate classrooms without building a costly infrastructure at each university. Member universities will contribute software knowledge infrastructure to the SSIP and will use the SSIP as a resource to support their classes. Operation of the SSIP is supported by industry sponsors at no cost to universities. Eventually, the SSIP would like to provide infrastructure to every interested university in order to teach the latest leading-edge software knowledge to their students.

Shared Software Infrastructure Program (SSIP)

SSIP was created in spring 2005. Current industrial sponsors of the program are AVNET, IBM, and Intel. The program is sponsored by companies who share a vision of integrated information flow

within and among enterprises based on OSS, open standards, and global interoperability. The SSIP will support tools compliant with the OMG computer industry specifications for interoperable enterprise applications. Services provided by the SSIP will be determined by the member universities that use the SSIP Web site. Services and software will be provided to member universities at no cost to the universities. Costs of operating the SSIP and developing the infrastructure will be borne by sponsors.

Initial courses supported by the SSIP were capstone software engineering courses that had a software project in which teams of students develop a software product starting with the customer requirements and finishing with a demonstration of a working product. Students are introduced to a full set of computer-aided software engineering (CASE) tools. CASE tools were introduced across all phases of the DSLC. Each week a new tool with open was introduced. For each tool, the SSIP staff provided an overview, tool use examples, and online tutorials, and suggested assignments and a tool Web site. SSIP 34 Member Universities for fall 2006 include the following:

- Arizona State University, Tempe, Arizona
- Auburn University, Auburn, Alabama
- California State University, Los Angeles, California
- DePaul University, Chicago, Illinois
- Iowa State University, Ames, Iowa
- Louisiana State University, Baton Rouge, Louisiana
- Marquette University, Milwaukee, Wisconsin
- Mississippi State University, Mississippi State, Mississippi
- Neumont University, South Jordan, Utah
- North Carolina State University, Raleigh, North Carolina
- Pace University, New York City, New York
- Purdue University, West Lafayette, Indiana
- Queens University, Kingston, Canada
- Rutgers University, New Brunswick/Piscataway, New Jersey
- San Jose State University, San Jose, California
- Sacramento State University, Sacramento, California
- Southern Methodist University, Dallas, Texas
- Texas A&M International University, Laredo, Texas
- Texas A&M University, College Station, Texas
- Texas A&M-Corpus Christi, Corpus Christi, Texas
- Texas State University, San Marcos, Texas
- Texas Tech University, Lubbock, Texas
- University of Arizona, Tucson, Arizona
- University of Arkansas, Fayetteville, Arkansas
- University of California – San Diego, San Diego, California
- University of Houston -- Clear Lake, Houston, Texas
- University of Kentucky, Lexington, Kentucky
- University of Missouri – Rolla, Rolla, Missouri
- University of North Texas, Denton, Texas
- University of Oklahoma, Norman, Oklahoma
- University of Tennessee at Chattanooga, Chattanooga, Tennessee
- University of Tennessee at Knoxville, Knoxville, Tennessee
- University of Texas at Arlington, Arlington, Texas
- University of Texas at Dallas, Dallas, Texas

The current OSS tools supported by the SSIP for the software engineering capstone courses are the following:

- **Apache:** HTTP server and application server
- **CVS:** Configuration management system
- **Derby:** Database management system
- **Eclipse:** Platform for building an integrated development environment with plug-ins for tools
- **FireFox:** Web browser
- **Gantt Project:** Project planning software
- **Java:** With supporting tools
- **JRequire:** Requirements engineering tools
- **Linux:** Operating system
- **Tomcat:** Application server

At no cost to SSIP member universities, SSIP sponsors are very helpful in closing the information technology gap between software used by industry and software used in classrooms at universities. Avnet has agreed to provide computer server hardware, and Intel has agreed to support software and provide access to the Intel Software College (2006) where students can learn how to optimize and accelerate applications and to enhance software design, anticipate and address potential issues, and improve application performance. They also provide online courses as well as live and recorded Webcasts. IBM and IBM Rational provide computer servers, operational support, and the following software tools:

- **ClearCase:** Configuration management system
- **ClearQuest:** Defect tracking and change management system
- **DB2:** Database management system
- **ProjectConsole:** Visual project monitoring tool
- **PureCoverage:** Code coverage tool
- **Purify:** Automatic error detection tool for finding runtime errors and memory leaks
- **Quantify:** Performance analysis tool
- **RequisitePro:** Requirements tracking tool
- **Robot:** Automated functional regression testing tool
- **RSA:** Rational Software Architect visual modeling tool
- **SoDA:** Report generation tool that supports day-to-day reporting and formal documentation requirements
- **Test Manager:** Test management tool
- **Websphere:** Web server technologies
- **SSIP distributes content using the SSIP Web site located at the following URL:** http://ssi7.cs.tamu.edu/ssi/

For each software tool, the SSIP Web site contains a short tool overview describing the tool in terms easily understood by a student. There are also online tutorials for how to use the tool as part of the student team project. Use-cases are used to describe the relationship of the user to the sample application of the tool. Where available, a WWW link points to the tool Web site. New SSIP member universities are provided WWW linkages to course Web sites for courses that use SSIP content and services. The SSIP provides a user help service to answer questions about any of the tools. Where required, mentors are made available to provide one-on-one tool use help. SSIP servers are available for student project teams to test their project software products. SSIP user forums can be set up for member schools to discuss all aspects of introducing the latest software technology into classrooms.

Many of the CS education program problems are solved by using the SSIP. Students that come to the university today have their own personal client computer that can connect to the Internet. They can connect to computer servers for everything needed in a university curriculum. The SSIP has servers available to SSIP member universities to support member university courses. By using OSS and free software tools provided by industry, the latest software solutions used by industry can be introduced into classrooms at very little cost to the SSIP member university.

Tenured university faculty members who control the courses taught can reduce the time it takes to introduce into the classrooms the latest advanced software solutions used by industry. In the lecture part of a course, the faculty member introduces software development theory, practice, and processes. In the software project laboratory, tools used to support software development are introduced by distance learning through an SSIP Web site. Examples showing the use of each tool are provided. When additional assistance is required, the SSIP operates a help desk and can supply mentors.

The problem of inexperienced laboratory assistance is solved by SSIP supplying services in which students in the laboratory obtain all of the knowledge that is necessary to learn and effectively use software tools. As a result, laboratory assistants spend most of their time managing the student laboratory assignments and activities.

Software tools and support of the tools are provided through the SSIP Web site. The goal is to minimize the support staff that must be provided at the local university. The cost of development tools is minimized by use of OSS and by free software provided by industrial sponsors. Interoperability of applications developed by student software development teams is assured by emphasizing open standards.

Industry today is looking to hire students who know how to be a productive team member. Often universities are reluctant to let student projects use department computer servers shared with other applications for fear that the students will cause the servers to fail. Student project teams need a type of sand-box server on which the student team can build a software product. Sand-box computer servers are provided by the SSIP for use by SSIP member universities.

Help desks and mentoring services are expensive. Industry provides extensive help desk and mentoring services to their customers at great costs. Individual universities cannot afford to provide these infrastructure services to students for the wide variety of software tools needed to support team software development projects in order to create the latest software solutions. By member universities sharing these services, the SSIP can provide services to a large number of universities at a low cost to SSIP sponsors and at no cost to the universities. The SSIP can make these services available 24/7.

Without outside help, universities have difficulty making the latest software solutions available to the students in the classroom. Three barriers to availability are cost, training, and licensing. An SSIP goal is to solve the availability problem by providing open source or free software tools at no cost to SSIP member universities, software tool training classes by distance learning through the SSIP Web site, and free licensing to SSIP member universities.

The SSIP has been well received by member schools. Leading-edge software knowledge is being introduced into university classes at no cost to the university. We are very encouraged with the SSIP success to date. We plan to continue to take advantage of the existing environment in which every student has his or her own Internet-connected client computer on which the student can access the latest software knowledge content from Internet-connected SSIP servers.

CONCLUSION

With the beginning of the new millennium, software development is in a state of change. Low-cost client computers that can be interconnected by the Internet are available worldwide. Software development teams can be globally distributed around the world. OSS tools can be used to create infrastructures to help introduce industrial strength software into university classrooms. The latest software development process and practices along with open standards can help university students learn how to create enterprise-level interoperable software solutions. The SSIP is an example of how

universities working with industry can cooperate to share infrastructure to rapidly close the gap between advanced software technology used by industry and the software knowledge and skills taught in the classroom.

REFERENCES

Denning, P. J., & McGetrick, A. (2005). Recentering computer science. *Communications of the ACM*, *48*(11), 15-19.

Flowers, W. (2000). Why change? Been doin' it this way for 4,000 years! In *Proceedings of the ASME Mechanical Engineering Education Conference*, Fort Lauderdale, Florida. Retrieved July 6, 2006, from http://www.asmenews.org/archives/backissues/may/features/educonf.html

Frauenheim, E. (2004). Students saying no to computer science. *ZDNet News.* Retrieved July 6, 2006, from http://news.com.com/Students+saying+no+to+computer+science/2100-1022_3-5306096.html

Intel Software College. (2006). Retrieved July 6, 2006, from http://or1cedar.cps.intel.com/softwarecollege/HomePage.aspx

Kshetri, N. (2005). Structural shifts in the Chinese software industry, *IEEE Software*, *22*(4), 86-93.

Naur, N., & Randell, B. (1968). *Report on a conference sponsored by the NATO SCIENCE COMMITTEE*, Garmisch, Germany.

Object Management Group. (2006). Retrieved July 6, 2006, from http://www.omg.org

Open Source Initiative. (2006). Retrieved July 6, 2006, from http://www.opensource.org/

Patterson, D. (2005a). Stop whining about outsourcing! *ACM Queue*, *3*(9), 63-64.

Patterson, D. A. (2005b). Restoring the popularity of computer science. *Communications of the ACM*, *48*(9), 25-26.

Software Engineering Institute at Carnegie Mellon University. (2006). Retrieved July 6, 2006, from http://www.sei.cmu.edu/

U.S. Department of Labor Statistics. (2005). *Occupational outlook handbook, 2006-2007 edition.* Retrieved July 6, 2006, from http://www.bls.gov/oco/ococ267.htm

Webster's new collegiate dictionary. (1981). Springfield, MA: C. & C. Merriam Company.

KEY TERMS

Capstone Project: Designed for students to synthesize and integrate knowledge acquired through course work and other learning experiences.

Computer-Aided Software Engineering (CASE) Tools: Software tools used to assist in the development and maintenance of software.

Development Software Life Cycle (DSLC): Includes the multiple phases during which defined information technology work products are created or modified as part of the software development process. The last phase of development occurs when the software product is placed into operation.

Interoperable Software: Software that operates with various kinds of software applications and systems by agreeing on a common method with which to communicate and exchange data with one another.

Open Source: Refers to software that has Open Source Initiative (OSI) (2006) licenses. Examples of open source software are Linux, Apache, Eclipse, Derby, and so forth. Also included is open standard compliant software that is provided free

to universities for classroom use. An example is IBM Rational Software Architect (RSA).

Open Standard: Refers to standards that are publicly available. The Object Management Group (OMG) (2006) is an example of an organization that was created to produce open standards.

Outsource: To send work that would normally be done by employees in a company to workers that are employed by an outside company.

Productivity Paradigm Change: The improvement of productivity by use of the Internet, clients and servers connected to the Internet, improved communication technologies, advanced software tools, and outsourcing to low-cost labor regions.

Shared Software Infrastructure Program (SSIP): The goal of SSIP is to set up an infrastructure shared among universities in which universities can easily introduce the latest leading software knowledge into both undergraduate and graduate classrooms without building costly infrastructure at each university.

Software Tool: A software product that software developers use to create, debug, or maintain software.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant, and B. Still, pp. 670-680, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 4.11
The Migration of Public Administrations Towards Open Source Desktop Software:
Recommendations from Research and Validation through a Case Study

Kris Ven
University of Antwerp, Belgium

Dieter Van Nuffel
University of Antwerp, Belgium

Jan Verelst
University of Antwerp, Belgium

ABSTRACT

Several public administrations (PA) have expressed an increasing interest in open source software in the past few years and are currently migrating to open source software on the desktop. Given the large impact such a migration has on the organization, there is a need for learning from the experiences of previous migrations. In this chapter, we deduct a number of recommendations and lessons learned from previous research conducted on the migration of PAs to open source desktop software. Next, we describe a case study on the migration of the Brussels-Capital Region towards OpenOffice.org, and compare their experiences to these recommendations. In general, our results are quite consistent with previous findings, but also indicate that additional research is still required in order to create a set of best practices—based on empirical research—for the migration towards open source software on the desktop.

INTRODUCTION

In the past few years, open source software has become a viable solution for organizations, and is being increasingly adopted. This increased popularity has been enabled by the fact that open source vendors (e.g., RedHat and SUSE) and traditional software vendors (e.g., IBM and HP) provide reliable support for open source solutions. Studies indicate, however, that organizations are primarily using open source software for server applications (see e.g., Dedrick & West, 2003; Lundell, Lings, & Lindqvist, 2006; Ven & Verelst, 2006; Wichmann, 2002). This can be explained by at least two factors. First, open source software has a strong tradition in developing server-side applications. Given this background, most open source projects are situated in horizontal domains such as Internet applications, developer tools, and technical tools (Fitzgerald, 2005). Thanks to the maturity level of most well-known open source server software (e.g., Apache and Linux), these packages are widely diffused through organizations. Successful open source software for the desktop has surfaced only recently with applications such as OpenOffice.org, Mozilla Firefox, and Mozilla Thunderbird. Second, a migration towards open source software on servers is far less disruptive for members of an organization than a migration at the desktop. In case a Web server running Microsoft IIS is replaced by the Apache Web server, or the operating system for an ERP system is changed from Unix to Linux, end users in the organization will not (or hardly) be affected by this change. A migration from Microsoft Office to OpenOffice.org will, however, affect all end users in an organization, possibly even impacting productivity.

Recently, there has been an increased interest in migrations towards open source software on the desktop. Interestingly, this trend is primarily driven by public administrations (PA). In fact, PAs can be considered pioneers in the adoption of open source desktop software. At first sight, this is actually quite remarkable. In the past, it was frequently assumed that PAs are restricted by their organizational structure, and thus limited in their innovative behavior (Nye, 1999; Moon & Bretschneider, 2002). Other studies have found PAs to be surprisingly innovative with respect to certain innovations (Bretschneider & Wittmer, 1993; Moon & Bretschneider, 2002). The use of information technology is currently considered to be an opportunity for PAs to improve their efficiency, as illustrated by the large number of recent e-government initiatives. There are two important drivers for the adoption of open source desktop software in PAs. First, it has been suggested that PAs should be conscious of their IT expenses, to make efficient use of taxpayers' money (see e.g., Applewhite, 2003; Brink, Roos, Weller, & van Belle, 2006; Fitzgerald & Kenny, 2003; Waring & Maddocks, 2005). Since the license costs for open source software are either absent or at least lower than for proprietary software, open source software has often been touted as a means for reducing overall software expenses. Since the number of desktop licenses is far greater than the number of server licenses, cost savings on the desktop may be considerably larger. Second, some authors argue that PAs should use open standards in their communication with citizens, to avoid that citizens need to buy a commercial product for communicating with the PA (see e.g., Applewhite, 2003; Kovács, Drozdik, Zuliani, & Succi, 2004b; Rossi, Scotto, Sillitti, & Succi, 2005). Other reasons for the adoption of open source software by PAs include supporting the local economy, increased flexibility and avoiding vendor lock-in (see e.g., Drozdik, Kovács, & Kochis, 2005; Kovács et al., 2004b; tOSSad, 2006; Waring & Maddocks, 2005).

Notwithstanding the advantages that open source software can offer, the migration towards open source software on the desktop will be disruptive for most users within a PA. Hence, special attention should be paid to planning the migration in order to minimize discomfort and disruptions

for end users. Although the migration towards open source desktop software is a relatively new phenomenon, a number of academic studies have already described case studies of migrations of PAs towards open source software (mainly OpenOffice.org and Linux). As a result, some lessons can be learned from these migrations. Due to the exploratory nature of this research, results of these migrations are rather fragmented.

The purpose of this chapter is to integrate the recommendations from various studies on the migration of PAs towards open source software on the desktop. Subsequently, we will report on the migration of the Government of the Brussels-Capital Region towards OpenOffice.org and compare their approach to the recommendations made in previous reports. The chapter is structured as follows. First, we will provide a brief background on migrations undertaken by PAs in Europe, and the academic literature on this topic. Next, we will derive a number of recommendations with respect to the migration towards open source software on the desktop. In the fourth section, we will present the migration of the Brussels-Capital Region and compare our results to the recommendations made in previous research. Finally, we will discuss a number of implications for practice and research, and will offer our conclusions.

RELATED INITIATIVES AND STUDIES

Open standards and open source software are increasingly used by PAs in Europe. In fact, it has been suggested that PAs will be one of the driving forces behind open source software in Europe in the next few years (González-Barahona & Robles, 2005). Indeed, several initiatives for studying the adoption of open source software have been taken at different levels in the European administration (i.e., the European Community, national governments, regional governments, and municipalities). In fact, most European countries have initiated programs to study the advantages and drawbacks of open source software in the PA (Canonico, 2005; González-Barahona & Robles, 2005), or have created policies with respect to the use of open source software[1]. The interest of the European Commission (EC) in open source software dates back to 1999, when the European Working Group on Libre Software[2] was founded. Since then, there has been increased commitment from the EC to open source software. Most important in this regard is the IDA (Interchange of Data between Administrations) program, and its successor IDABC[3] (Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens). The aim of the program is to develop cross-European e-government services towards citizens and businesses. As part of the IDA program, the eEurope 2005 Action Plan[4] was written. The document states that to ensure interoperability, open standards will be used in e-government services. Furthermore, the use of open source software will be encouraged. Today, the IDABC Web site offers much information on migrations towards open source software in PAs[5]. Similarly, several research projects on the adoption and use of open source software are funded by the EC (e.g., COSPA[6], FLOSSPOLS[7], Calibre[8] and tOSSad[9]).

In Europe, several PAs have already migrated—or are planning to migrate—to open source software, including on the desktop. Several migrations of PAs are known in Austria, the Czech Republic, Finland, France, Germany, Spain, the Netherlands, and the United Kingdom[10]. Although these national initiatives vary from country to country, many of the PAs seem to at least investigate the use of open source software (González-Barahona & Robles, 2005). An illustrative sample of these initiatives is listed in Table 1. Many other initiatives are however known, especially concerning the use of open source software on servers. An example of a famous successful migration is found in Extremadura in Spain. In order to increase the IT literacy in this region—but faced

Table 1. Migrations of public administrations towards open source desktop software

Country	Projects
Spain	*Extramadura region:* Project to increase connectivity and IT literacy of the region. A custom Linux distribution, gnuLinEx (http://www.linex.org), was created.
France	*Gendarmerie Nationale:* Migration towards OpenOffice.org and Mozilla Firefox on ± 80,000 desktops. *City of Paris:* Migration towards OpenOffice.org and Mozilla Firefox on 17,000 desktops.
The Netherlands	*City of Haarlem:* Migration of 2,000 desktops to OpenOffice.org. *City of Amsterdam:* Currently conducting pilots to investigate the feasibility of migrating towards OpenOffice.org. *City of Groningen:* Decided to migrate 3,650 desktops to OpenOffice.org.
Germany	*City of Munich:* Migration of 14,000 desktops to Linux.
Austria	*City of Vienna:* Migration of 7,500 desktops to OpenOffice.org and Linux. A custom Linux-distribution, Wienux (http://www.wien.gv.at/ma14/wienux.html), was created.

with a limited budget—it was decided to base the project on open source software. This resulted in the creation of a custom Linux distribution, called *gnuLinEx*. Originally, this distribution was intended for classroom use, but is used nowadays in PAs as well (Vaca, 2005). On the other hand, some of these migrations fail—or are delayed by various problems—as illustrated with the migration of the city of Munich[11] in Germany.

Several academic studies have been devoted to investigating the difficulties encountered in migrating towards open source software on the desktop and to obtaining an overview of the use of open source software by PAs. An overview of these studies is shown in Table 2. Most of these studies have used a qualitative, case study-based approach to study the migration of various public administrations. Their aim was to investigate the feasibility of the transition, describe the migration itself, and to highlight any difficulties experienced during the transition, as well as recommend solutions. Some authors have combined the qualitative approach with an experiment to investigate the usage patterns of OpenOffice.org in comparison with Microsoft (MS) Office, to provide quantitative data on the migration (Rossi et al., 2005; Rossi, Russo, & Succi, 2006).

An important study in this field is the recent FLOSSPOLS study (Ghosh & Glott, 2005), that conducted a large-scale survey among 955 PAs in 13 European countries. The study provides insight into the perceived advantages of and barriers to the use of open source software. The study also provides an overview of the extent and type of open source software that is being used by European PAs. Results show that 49% of PAs intentionally use open source software. Interestingly, many PAs seem to be unaware of their use of open source software. In about 29% of the cases, open source software is being used without the respondent being aware of the fact that the software is open source. Results further show that half of the respondents would find an increase in open source software usage useful. The use of open source software is still mainly focused on servers, with only about 20% of the organizations using OpenOffice.org to some degree (Ghosh & Glott, 2005). The use of open source software on the desktop seems, however, to be somewhat higher than three years earlier, as reported by the first FLOSS study (Wichmann, 2002). This study showed that only 6.9% of PAs and businesses used open source software on the desktop[12] (Wichmann, 2002).

Table 2. Studies on the adoption of open source software by public administrations

Study	Research design
Wichmann (2002)	Large-scale survey as part of the FLOSS project to study the use of open source software in organizations and PAs in Germany, UK, and Sweden.
Russo et al. (2003)	Presents the results of a trial migration performed by the Province of Bolzano-Bozen (Italy) in 10 local PAs.
Fitzgerald and Kenny (2003)	Describes the migration of an Irish (public sector) hospital towards open source software.
Zuliani and Succi (2004b)	Reports on some lessons learned on the migration towards OpenOffice.org of 60 PAs in the Province of Bolzano-Bozen (Italy).
Kovács et al. (2004b); Kovács, Drozdik, Zuliani, and Succi (2004a)	Presents an overview of the advantages and challenges in migrating towards open standards and open source software in the PA.
Zuliani and Succi (2004a)	Provides the results of a migration towards OpenOffice.org of 60 PAs in Bolzano-Bozen (Italy).
Ghosh and Glott (2005)	Follow-up survey of the FLOSS project, investigating the use of open source software in PAs in 13 European countries.
Drozdik et al. (2005)	Investigates the risks involved in migrating desktops completely to open source software (i.e., OpenOffice.org and Linux), based on a PA in Törökbálint (Hungary).
Rossi et al. (2005)	Reports on an experiment on the transition from MS Office to OpenOffice.org, studying the use of OpenOffice.org throughout 32 weeks.
Waring and Maddocks (2005)	Reports on advantages and disadvantages of open source software for the UK public sector, including the results of eight case studies.
COSPA (2005)	Reports on the experiences of the migrations in seven European PAs, conducted as part of the COSPA project.
Rossi et al. (2006)	Reports on an experiment in a PA to compare the use of MS Office and OpenOffice.org documents after migrating towards OpenOffice.org.
Brink et al. (2006)	Reports on three case studies in South African organizations who have migrated towards open source software on the desktop. Two of the cases are situated in the public sector.
Jashari and Stojanovski (2006)	Survey of 14 municipalities in Macedonia to highlight the challenges and obstacles to use open source software.

RECOMMENDATIONS FROM PREVIOUS LITERATURE

The studies that are described in the previous section, have provided more insight into the migration process that was followed by a number of PAs. Potential adopters of open source software can draw important lessons from these studies to avoid running into the same problems that previous migrations already handled effectively. Unfortunately, given the exploratory nature of this research, the literature on this topic is rather fragmented. Few attempts have been made to integrate these results, although the COSPA and tOSSad projects are currently working towards this. Hence, our aim is to integrate the results from the currently available empirical literature, to provide a comprehensive overview of the recommendations and "*lessons learned*" from previous migrations.

Although ideally we would like to present a number of guidelines or best practices, we feel that this is currently not yet feasible because of a number of reasons. First, this type of research is

Table 3. Recommendations and lessons learned from previous research

1. Analysis and Preparation

(a) Planning

- Prepare well for the migration by performing proper analysis and planning (Brink et al., 2006).
- Make a detailed business case for the migration, including the expected cost (IDA, 2003).
- Cost is (one of) the most important reasons for migrating to open source software (Zuliani & Succi, 2004a; Brink et al., 2006; COSPA, 2005; Fitzgerald & Kenny, 2003; Waring & Maddocks, 2005).
- The migration will however imply important switching costs (e.g., training and migration), which may become a barrier (Drozdik et al., 2005; Kovács et al., 2004b, 2004a; Waring & Maddocks, 2005; IDA, 2003).
- Consequently, the real cost savings (if present at all) are difficult to quantify (Drozdik et al., 2005; Russo et al., 2003; Wichmann, 2002; COSPA, 2005).
- Other factors such as quality and productivity may also be important (Zuliani & Succi, 2004a).

(b) Pilot Study

- The use of a pilot study is recommended (IDA, 2003; Brink et al., 2006).

2. Migrating towards Open Source Software

(a) Pace of Migration

- A "big bang" approach should be avoided since it increases the risk of the migration (IDA, 2003).
- When migrating to a fully open source desktop on Linux, users should first be migrated to open source desktop software such as OpenOffice.org and Mozilla on MS Windows. In a second phase, MS Windows can be replaced by Linux (COSPA, 2005; Drozdik et al., 2005).
- When replacing MS Office by OpenOffice.org, there should be a transition phase in which users have access to both office suites to lower user resistance (Zuliani & Succi, 2004b; Zuliani & Succi, 2004a; COSPA, 2005). On the other hand, users are then likely to expect the same behavior from OpenOffice.org as from MS Office (Drozdik et al., 2005).

(b) Top Management Support

- Top management support has been found to be critical during deployment (Brink et al., 2006; Fitzgerald & Kenny, 2003).

(c) Attitude of End Users

- It is important to create a positive attitude with end users before the migration, since personnel resistance is the most important issue in the migration (Drozdik et al., 2005; Rossi et al., 2005; Zuliani & Succi, 2004b, 2004a).
- Personnel may perceive the transition negatively if they are satisfied with the current application that they may be using at home as well (Russo et al., 2003; COSPA, 2005).
- It is important to consult and communicate with users in order to minimize discomfort for users (Drozdik et al., 2005; IDA, 2003).
- Users may fear becoming "deskilled" in using a non-industry standard, thereby decreasing their value on the labor market (IDA, 2003; Fitzgerald & Kenny, 2003).

3. Training

- Although users with a good knowledge of MS Office tend to experience few problems with the transition to OpenOffice.org (Kovács et al., 2004b; Kovács et al., 2004a; Russo et al., 2003), it is important that employees receive proper training to improve user acceptance (Rossi et al., 2005; Brink et al., 2006; IDA, 2003; COSPA, 2005; Kovács et al., 2004b, 2004a).
- Training is also important because some functions in OpenOffice.org behave differently than in MS Office or have different names (Jashari & Stojanovski, 2006; COSPA, 2005; Drozdik et al., 2005).
- Training should immediately precede (or follow) the migration, so that users can start practicing their skills in OpenOffice.org (Zuliani & Succi, 2004b; Zuliani & Succi, 2004a; COSPA, 2005).
- Training should focus on generic capabilities in office productivity, i.e., functionalities that are used each day (Russo et al., 2003; COSPA, 2005).
- The training approach can vary, e.g., face-to-face training, interactive tutorials over the Intranet and seminars (Russo et al., 2003; Brink et al., 2006).

continued on the following page

Table 3. continued

4. Support

- A lack of external support for assisting in the migration can be a barrier, if the required knowledge is not available in-house (Kovács et al., 2004b; Kovács et al., 2004a; Jashari & Stojanovski, 2006).
- It is important that users have access to several kinds of support, e.g., knowledge bases containing FAQs, guides, and manuals; access to telephone and e-mail support; and access to a contact person (e.g., a product champion) in case of questions (Zuliani & Succi, 2004b; Zuliani & Succi, 2004a; Brink et al., 2006; IDA, 2003).

5. Document Conversion

- The conversion of OpenOffice.org documents from and to MS Office format does not cause many issues in most cases, although some incompatibilities may occur (e.g., wrong image size or margin settings) (COSPA, 2005; Drozdik et al., 2005; Russo et al., 2003).
- However, when using complex documents with precise formatting or MS Office macros, conversion may become a very labor-intensive task, and conversion incompatibilities may arise, especially if these files are frequently converted from MS Office to OpenOffice.org format and vice versa (COSPA, 2005; Zuliani & Succi, 2004b, 2004a; Drozdik et al., 2005).

6. Functionality

- Perceptions towards the functionality offered by OpenOffice.org tend to vary. Some users report that OpenOffice.org offers the same functionality as MS Office (Rossi et al., 2005; COSPA, 2005), while other users complain about missing features (Rossi et al., 2005).
- In general, the functionalities of OpenOffice.org seem more than adequate for daily use, and a migration is possible with no or few problems (Zuliani & Succi, 2004b; Rossi et al., 2005; COSPA, 2005).
- It should be noted, however, that depending on the use of OpenOffice.org, specific issues may arise (e.g., mail merge feature, and differences in hard and soft line breaks) (Zuliani & Succi, 2004a; COSPA, 2005).
- Interoperability and compatibility with existing systems (e.g., databases and desktop applications) can be a problem in certain situations (Kovács et al., 2004b; Kovács et al., 2004a; Zuliani & Succi, 2004a).

still exploratory, and there are still many variations in the adoption processes followed and in the context in which the migration takes place, leading to a more or less—depending on the situation—successful migration. Second, and more importantly, the studies that are currently available are not likely to be representative for all PAs. Studies in this domain tend not to report how the cases in their sample were chosen. Hence, the possibility cannot be excluded that the cases were selected out of practical considerations (e.g., having access to the site), rather than based on their theoretical relevance. Consequently, the lessons learned that are presented here should be considered preliminary, and the results of future studies should be contrasted with this set of recommendations.

Based upon previous studies, we derived a set of recommendations and lessons learned in a number of different areas. This set was derived as follows: first, the literature was searched for studies on the adoption of open source desktop software by PAs. We restricted our search to empirical studies in academic literature, discussing the experiences and results from migrations in PAs. We also chose to include the *IDA Open Source Migration Guidelines* (IDA, 2003) in the literature study. Although this is not an academic study, it was one of the first studies to provide recommendations on how PAs should migrate towards open source software. It has been reported, however, that these guidelines are not very often used in practice (Zuliani & Succi, 2004a). The studies included in this literature study are listed in Table 2. Next, the studies were analyzed by one of the authors and any lessons learned or recommendations were coded in the text. The initial coding categories were based on several aspects related to a migration process that are commonly known to be relevant (e.g., training, analysis, and

document conversion). Coding was flexible and opportunistic, adding new categories when they were encountered. This initial list of lessons learned was subsequently reviewed, and data were further integrated as necessary. The revised list was then reviewed by a second author and some additions were applied as required. The result of this analysis is shown in Table 3. In total, six different areas regarding the transition were identified, namely *analysis and preparation*, *migration*, *training*, *support*, *document conversion*, and *functionality*. Each area contains a number of recommendations and lessons learned from previous research.

CASE STUDY IN THE BRUSSELS PA

In this section, we will present the results of a case study on the migration of the Government of the Brussels-Capital Region in Belgium. The Brussels-Capital Region consists of the 19 municipalities of Brussels. The Government of the Brussels-Capital Region consists of eight Ministries, each having its own cabinet. The *Brussels Regional Informatics Center (BRIC)*, responsible for the promotion and assistance of information technology (IT) within the Government of the Brussels-Capital Region, was responsible for organizing the transition towards OpenOffice.org. Our aim is to compare the approach taken by BRIC to the list of lessons learned that was compiled in the previous section. Due to space limitations, we will focus exclusively on the areas present in Table 3.

Methodology

A descriptive case study approach was used to study the transition towards OpenOffice.org at the Government of the Brussels-Capital Region. The case study approach allowed us to study the phenomenon in its real-life context (Benbasat, Goldstein, & Mead, 1987; Yin, 2003). An embedded case study design was used in order to investigate the migration towards OpenOffice.org at BRIC as well as at the ministerial cabinets of the Brussels-Capital Region. Using the key informant method, we selected two informants within BRIC, since the use of a single informant may lead to unreliable results (Benbasat et al., 1987; Phillips, 1981). Our informants were the director of the IT department and the project leader who was assigned to the OpenOffice.org project. Both informants were highly involved in the migration towards OpenOffice.org, and were responsible for planning and coordinating the migration, developing documentation, designing the training sessions, and conducting user evaluations.

The primary mode of data collection consisted of two face-to-face interviews which were conducted by a two-person team. During the first interview, important background information on the transition was gathered. Based on this information, the case study protocol was completed, leading to the generation of a detailed set of questions. During the second interview, detailed information on the migration was gathered from our informants. This interview was digitally recorded for future reference. One researcher of the team was primarily responsible for posing the interview questions, while the other was responsible for taking notes and supplementing the interview with additional questions. Using different roles for each researcher also allowed us to view the case from two perspectives and to compare the impressions of both researchers afterwards (Eisenhardt, 1989; Yin, 2003). Additional sources of evidence were internal documents of BRIC, legislative texts, and secondary information such as press releases. Extensive follow-up questions on the interview and recent developments took place via e-mail. A draft copy of the case study report, as well as a draft of this chapter, was reviewed by our informants to increase the validity of our findings.

Findings

Analysis and Preparation

Planning

The migration to OpenOffice.org in the Brussels PA was driven by two political decisions. First, a resolution was voted in which the use of open standards and open source software was encouraged by the Brussels-Capital Region in order to facilitate communication with its citizens. Consequently, BRIC was required to consider at least one open source alternative in each new project. Second, following this resolution, the coalition agreement of the Brussels-Capital Region in 2004 also stated that the use of open standards and open source software would be encouraged within the Brussels-Capital Region. Based upon this coalition agreement, the Government of the Brussels-Capital Region decided that open source office software would be used by the ministerial cabinets of the Brussels-Capital Region. OpenOffice.org was, however, not mentioned by name at that time. The migration involved installing OpenOffice.org 1.1 on 400 workstations running on MS Windows XP. In addition, four out of eight servers of the ministerial cabinets were migrated from MS Windows to Linux.

Preceding the migration, no formal TCO analysis was carried out. Although the main argument for migrating towards OpenOffice.org was to facilitate communication with citizens, cost savings realized by the migration were also stressed, especially in public announcements. License costs were cut back by 185,000 EUR during the first year and 15,000 EUR in the subsequent years (when a limited number of remaining workstations will be migrated). On the other hand, our informants confirmed that significant hidden costs of training and support occurred. Unfortunately, it was not possible to accurately quantify these hidden costs.

Pilot Study

To verify the feasibility of a transition from MS Office to OpenOffice.org, BRIC performed a pilot project in March 2004. This pilot consisted of migrating the workstations of BRIC personnel to OpenOffice.org. The outcome of the pilot project confirmed the feasibility of migrating the ministerial cabinets. As soon as OpenOffice.org 2.0 was available, BRIC again executed an internal pilot study. In contrast to the prior pilot, the aim was not only to prepare the upgrade towards OpenOffice.org 2.0, but also to enable BRIC to market the new version towards end users. They therefore invited the key users of each cabinet (i.e., the cabinet clerk and the local IT responsible) to participate in the pilot. Based on these experiences and significant improvements of OpenOffice.org 2.0 in comparison with version 1.1, it was decided in December 2005 to upgrade the ministerial cabinets to OpenOffice.org 2.0.

Migrating Towards Open Source Software

Pace of Migration

As a new Government of the Brussels-Capital Region is elected every five years, the computer equipment of the ministerial cabinets is updated simultaneously. Because of efficiency reasons, and to minimize discomfort for end users, BRIC decided to have the migration towards OpenOffice.org 1.1 coincide with the replacement of the PCs. When the user's workstation was replaced, only OpenOffice.org was installed and MS Office was no longer available to the user, except in a limited number of cases in which the user required advanced functionality of MS Excel or MS Access (e.g., in the finance department). Concurrently, the default data format for internal communication changed from MS Office to OpenOffice.org format. This change did not pose any insur-

mountable problems thanks to the import/export filters of OpenOffice.org. These filters enabled opening and saving previously existing MS Office documents, or MS Office documents that are sent from and to external users.

The decision to use a "big bang" approach was motivated by a number of reasons. First, BRIC judged that it would be more manageable to instantaneously switch to OpenOffice.org without temporarily installing MS Office. Furthermore, it was expected that users with an initial negative attitude towards OpenOffice.org would continue using MS Office as long as it was still available on their workstation. Finally, by changing the default document format to OpenOffice.org, BRIC wanted to encourage staff members to use OpenOffice.org.

Top Management Support

Top management support has frequently been shown to be positively related to the acquisition and successful implementation of new technologies (see e.g., DeLone, 1988; Rai & Patnayakuni, 1996). As already mentioned, the migration to OpenOffice.org was mandatory since it was a political decision. Therefore, top management (i.e., the Government of the Brussels-Capital Region) was formally supporting the migration. Given that the migration was mandated by law, BRIC was endowed with a powerful mandate while performing the migration. This helped in countering possible user resistance. It should be noted that some cabinet clerks were not in favor of the migration towards OpenOffice.org. It is likely that they have had an impact on the attitudes of end users. Unfortunately, it was not possible to empirically confirm this hypothesis.

Attitude of End Users

Within the ministerial cabinets, it could be observed that users who showed an initial negative attitude towards OpenOffice.org, remained rather skeptical. This fact confirms the importance of training and providing users with adequate information before migration takes place. The fear of becoming "deskilled" (Fitzgerald & Kenny, 2003) did not seem to have occurred. According to our informants, end users articulated no concerns about moving away from the industry standard, possibly reducing their value in the job market.

Training

According to the recommendations from previous literature, users should be able to start practicing with OpenOffice.org immediately after training. However, this was not always possible because of the narrow time frame in which the migration took place. Nevertheless, all users obtained their training within one week before or after their workstation was migrated.

The training consisted of a voluntary training course in the offices of BRIC and a CD-ROM. During the course, the basic functionality of OpenOffice.org Writer and Calc was explained. The first sessions of this course were intended for the key users of the different cabinets to enable them to provide first-line support to their users. Representatives of two out of eight ministerial cabinets found the training too basic, and did not encourage members of their cabinet to attend the training sessions. Afterwards, a survey among end users indicated that users who did not attend the course reported more problems in using OpenOffice.org.

It was observed that people tried to work in the same manner as they were used to in MS Office. This is however not always possible because some functions in OpenOffice.org are fundamentally different from their MS Office counterpart. Therefore, short personal demonstrations were organized during which BRIC personnel illustrated the procedure to be used in OpenOffice.org.

Although the general training session proved to be useful, BRIC noticed that additional, focused

training sessions would be required following the upgrade to OpenOffice.org 2.0, due to functional differences with OpenOffice.org 1.1. These sessions will be organized as short workshops: each session will focus on a particular functionality that different groups of end users require (e.g., using the Mail Merge feature or working with document templates). The workshops will also be more practically oriented than the initial training sessions.

Support

Three important sources of support are available to end users in the ministerial cabinets. First-line support for common problems can be provided by the key users in each cabinet. Second, BRIC itself provides end user support for OpenOffice.org. Third, during the pilot study and training, BRIC employees have built an extensive knowledge base on OpenOffice.org. This knowledge base—which is regularly updated by new questions formulated by end users—is sufficient to solve most problems.

Part of the knowledge base consists of a CD-ROM containing a manual, a FAQ list and the OpenOffice.org installation files. This CD-ROM was handed out to the participants of the training session. The manual and FAQ list were based on the documentation provided by the OpenOffice.org communities. However, since the Brussels-Capital Region is bilingual (with Dutch and French being the official languages), documentation had to be sourced from two different OpenOffice.org localization communities. It then became apparent that significant differences in quality between these online communities exist. While the French community is very vivid and provides much information, the Dutch counterpart does not produce the same quantity of documentation. This did lead to some difficulties, since BRIC is required to provide (equivalent) training material in both languages. Another indication of the difference between the French and Dutch community are delays in the release of upgrades. For both OpenOffice.org 1.1 and 2.0, the French localization was available much earlier than the Dutch version. This has been criticized by some end users as a disadvantage of working with open source software.

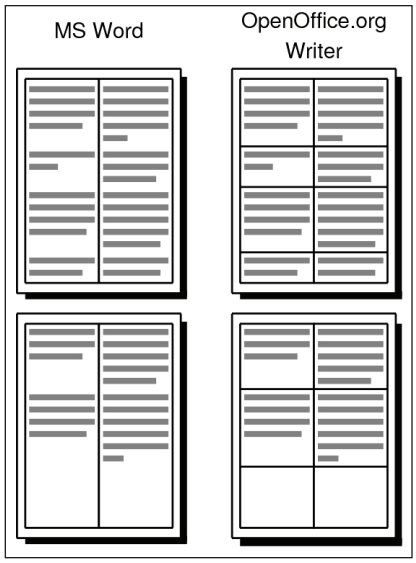
Document Conversion

In general, few problems concerning document conversion between MS Office and OpenOffice.org were reported. However, in three specific cases, some issues did occur. First, difficulties arose when documents with extensive and complex layout were exchanged with external parties. When these documents were converted several times back and forth between OpenOffice.org and MS Office format, incompatibilities in layout became unmanageable. As a result, it was decided to exchange documents without formatting as long as they needed revision. Once the final version of the document had been approved, the formatting was done either in OpenOffice.org or in MS Office. The use of the PDF-format was also promoted for documents requiring no further revisions.

Secondly, the conversion of MS Office document templates to OpenOffice.org posed some problems, mainly because a number of incompatibilities exist between the two products, for instance, with respect to margin settings. Since the Brussels PA has a very rigorous style guide, these templates had to be migrated very accurately. As a result, BRIC only recently finalized the OpenOffice.org templates.

A third issue, concerning the editing of legislative texts, was again caused by very specific format prescriptions. Since the Brussels-Capital Region is bilingual, these texts have to be published in two columns, one for each language. Moreover, each paragraph has to start on the same height as the corresponding paragraph in the other language. Given that French paragraphs are somewhat lengthier than the Dutch equivalents,

Figure 1. Formatting of legislative texts in MS Office vs. OpenOffice.org



some adjustments in vertical spacing between paragraphs must be made (see Figure 1). In MS Word this layout was realized by using a table with two columns and one row. OpenOffice.org 1.1, however, did not support multi-page table cells. Hence, the layout had to be changed by putting each paragraph in a different cell. When a cell does not fit on a single page, it is moved automatically to the next page. In OpenOffice.org 2.0 this workaround has become obsolete, since it is possible to use multi-page rows in the new version[13].

Functionality

Regarding functionality, several remarks can be made. First, a number of end users did not consider OpenOffice.org a fully fledged alternative to MS Office. This was mainly caused by the absence of certain features which are present in MS Office. The fact that MS Excel and MS Access are still used by a limited number of users, confirms that OpenOffice.org does not yet support all advanced features of MS Office. On the other hand, most users do not experience any problems in their daily tasks.

Furthermore, users reported problems concerning the Mail Merge feature. When using this feature to create a mailing based on an address list in a database, OpenOffice.org creates a new document for each addressee. To solve this issue, a script was developed to merge the separate files into one document. In OpenOffice.org 2.0, however, this problem does not occur anymore because the user can choose whether to generate one large file or to generate separate documents. It is therefore hoped that the improved functionality of OpenOffice.org 2.0 will enhance the perceived quality of OpenOffice.org by end users. Nevertheless, a number of deficiencies are still reported by users working with OpenOffice.org 2.0. Although workarounds are possible for most of these problems, these defects still have a negative impact on the general perception of OpenOffice.org.

The use of data sources by OpenOffice.org is considered to be a mixed advantage and disadvantage. In contrast to MS Office, OpenOffice.org allows a document to access more than one data source. The implementation of these data sources is, however, significantly different from MS Office: while in MS Office the data source is included in the document itself, OpenOffice.org stores it in the user profile at the user's workstation. Thus, when exchanging the document between users, the data source is lost. To solve this problem, BRIC wrote a script to ensure that each user has access to the commonly used data sources.

The perceived usability of OpenOffice.org 1.1 is also reported to be inferior compared to MS Office. The most often heard critique is that the look and feel of OpenOffice.org 1.1 feels outdated. Again, OpenOffice.org 2.0 could provide a solution because its look and feel has been updated considerably to resemble that of MS Office.

Discussion

In general, the findings in this case study are consistent with the experiences and recommendations from previous literature. The case study illustrated that a migration towards OpenOffice.org was possible within the Government of the Brussels-Capital Region. Nevertheless, the case study also confirms that there are a number of important issues that should be paid attention to when planning and performing the migration.

Following the recommendations from previous literature, it can be observed that the documentation available and training given to end users are very important. With respect to training, the case study showed that in a first phase, the training can focus on general office suite capabilities applied to OpenOffice.org. Therefore, it serves as a revision course of skills that are also applicable to MS Office. This is consistent with recommendations made in previous literature (see, e.g., COSPA, 2005; Russo, Zuliani, & Succi, 2003). In a second phase, however, it seemed useful to offer short, focused training sessions on particular tasks in working with OpenOffice.org. Both training and documentation can positively influence the attitude of end users. While the resistance of end users is frequently a major problem (Rossi, et al., 2005; Zuliani & Succi, 2004b), we feel that in this case the acceptance was facilitated not only by strong top management support, but also by the fact that the adoption of OpenOffice.org was mandated by law, giving end users no choice but to conform with the transition. In terms of interoperability, the document conversion facilities of OpenOffice.org seemed to be sufficient in most cases (COSPA, 2005; Russo et al., 2003). However, advanced use of OpenOffice.org, or specific applications (e.g., bilingual legislative texts) may lead to certain issues (COSPA, 2005; Zuliani & Succi, 2004a). Conducting a pilot test may help in identifying potential issues that arise during implementation within the specific context in which the migration will take place. Although the decision to start using OpenOffice.org was primarily politically motivated, there was an emphasis on the cost savings realized by the transition. It must be noted, however, that considerable effort was invested in developing appropriate training, documentation, and document templates.

One important difference with previous studies is that the "big bang" approach taken by BRIC did seem successful. Previous studies have recommended that a transition phase is planned in which MS Office and OpenOffice.org are installed concurrently on the users' workstation (COSPA, 2005; Zuliani & Succi, 2004a, 2004b). Based on the present study, we feel that there are situations in which MS Office can be replaced immediately with OpenOffice.org. While the transition period enables users to continue using MS Office in case they experience problems with OpenOffice.org, the immediate migration towards OpenOffice.org forces users to start using (and learning) OpenOffice.org. By having to rely solely on OpenOffice.org to complete office productivity tasks, users might become familiar more quickly with the functionality of OpenOffice.org, shortening the learning period. It also encourages users to start adopting the preferred OpenOffice.org methods, instead of trying to work with OpenOffice.org in the same way as with MS Office (Drozdik, et al., 2005). Therefore, we feel a "big bang" migration from MS Office to OpenOffice.org is feasible in certain situations. On the other hand, if a PA would decide to replace the MS Windows desktop with Linux desktops, it seems better to use a phased approach, as recommended in previous studies (COSPA, 2005; Drozdik, et al., 2005) (see also Table 3).

IMPLICATIONS

This study has a number of implications for both practice as well as research.

Implications for Practice

Several PAs have already decided to migrate towards open source software based solutions or have already completed their migration. It is likely that, thanks to the encouragement of open standards and open source software by the European Commission, many other PAs will also decide to migrate to open source software. The results of currently ongoing research projects such as COSPA, FLOSSPOLS, and tOSSad will probably further help PAs to decide on the possible risks and advantages. The lessons learned identified in this chapter should serve as a first step towards developing research-based best practices for the adoption of open source software on the desktop. The presented overview can be of help to PAs contemplating a migration. It will assist in identifying possible barriers or issues that should be anticipated in the project to avoid difficulties during implementation. As a result, PAs can learn from previous efforts of early adopters. We believe that by promoting the sharing of experiences of such migrations, future migrations can be handled more effectively (both in time and financially).

Based on available research, we believe that a migration towards open source software on the desktop is possible, but is contingent on the specific environment in which the migration will take place. If the PA is using advanced features of MS Office (e.g., the extensive use of macros), or when interoperability with existing systems and applications is important (e.g., database servers, or proprietary tools written for MS Office), it will be more difficult to migrate. Although workarounds can be devised, this will imply additional costs, and productivity may be negatively affected. We believe, however, that as more organizations and PAs decide to start using open standards and open source software, software vendors will be more likely to provide interoperability with open source solutions[14].

When the use of MS Office is mainly restricted to basic functionality—which seems to be the case in most PAs currently studied in literature—the functionality of OpenOffice.org should suffice in most cases. Nevertheless, PAs should consider the issues raised in this chapter, and sufficient attention should be paid to areas such as planning, training, and support. Moreover, PAs should be aware that specific requirements in their environments may lead to a number of issues that were not previously encountered. For example, due to the bilingualism in the Brussels PA, the implementation of tables in OpenOffice.org resulted in changing the way documents were formatted. This implied altering the daily work habits of end users, which may prove to be very difficult. Hence, having top management support—and possibly a product champion—may be required to ensure that new working practices are adopted. Furthermore, the use of a pilot study may help in encountering issues in the daily work habits of employees that need to be resolved before migrating.

Our results may also be relevant to the open source community. Results from previous studies, as well as the current study, indicate that one of the main drivers for a migration towards open source software is cost reduction. This is consistent with literature on the organizational adoption of open source software, which has shown that the lower or non-existing license costs of open source software is a major reason influencing the adoption decision (see e.g., Dedrick & West, 2003; Ven & Verelst, 2006). This is in contrast with the attitude of the open source community that tries to downplay the cost advantages, and tends to emphasize other advantages such as having access to the source code of the application, and being allowed to modify it. Studies in this field may offer the open source community a better insight into the perceptions of organizations and PAs, since it has been noted that the open source

community in general has limited access to the opinion of its customers (Fitzgerald, 2005). A more profound insight into the main motivations of adopters of open source software may lead the open source community (and especially the open source vendors) to emphasize different advantages of open source software. Similarly, open source communities (especially projects such as OpenOffice.org and Mozilla) may also try to actively solicit feedback from PAs in order to include missing features and further improve functionality.

Implications for Research

This study has a number of limitations. First, given the fact that the migration of PAs towards open source software on the desktop is a new phenomenon, relatively little research has been devoted to studying the migration process. Consequently, our lessons learned are based on the limited amount of studies that were available at the time of writing. Furthermore, it is difficult to assess the representativeness of these studies. Second, we compared the lessons learned with a single case study in the Brussels PA. Hence, the external validity of this study cannot be ascertained, and we therefore cannot generalize our results to all PAs. Future research can therefore study additional migrations that can be compared to the issues raised in this chapter. By incorporating additional findings from different contexts, a set of research-based best practices for the adoption of open source software on the desktop may be deducted. These may further help clarifying the factors that influence a successful migration. This study should be considered a first step in this direction.

Another interesting avenue for further research is to investigate to which degree the migration process in PAs is different from the process in enterprises. For example, PAs may be better suited to motivate—or force—their users to make the migration succeed. In the case of the Brussels PA, the migration towards OpenOffice.org was mandated by law, making it very difficult to resist this change. It is likely that enterprises are not capable of supporting the migration in such a strong way. Second, it is possible that enterprises generally make more advanced use of MS Office, or use it in a more complex environment, making the migration more difficult.

CONCLUSION

There has been an increased interest in open source software by PAs in the past few years. Some authors expect that this trend will continue in the following years, making PAs a driving force behind open source software in Europe (González-Barahona & Robles, 2005). Lately, much attention has been given to migrations towards open source software on the desktop. Given the apparent success of several attempts in this direction, we expect that PAs will continue to consider these migrations in the future. However, given the fact that a migration towards open source software on the desktop is quite disruptive for end users, it is important that guidelines are available. The recommendations deducted from previous research that are listed in this chapter can be considered a first step in this direction. By comparing the lessons learned from previous migrations to a case in the Brussels PA, it was noted that, in general, there was a good match between the literature and the current case. Due to the specific context of the case, we were able to highlight a few issues that may be of interest for future adopters. Furthermore, we feel that it is important to conduct additional case studies on the migration towards open source software and compare the results to the recommendations described in this chapter. This way, the recommendations will be based on a substantial body of knowledge, leading to a set of best practices for the migration towards open source software on the desktop.

TRADEMARK USE

Microsoft Office and Microsoft Windows are registered trademarks of Microsoft Corporation. OpenOffice.org is a registered trademark of Team OpenOffice.org e.V.

REFERENCES

Applewhite, A. (2003). Should governments go open source? *IEEE Software*, *20*(4), 88-91.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly*, *11*(3), 368-386.

Bretschneider, S., & Wittmer, D. (1993). Organizational adoption of microcomputer technology: The role of sector. *Information Systems Research*, *4*(1), 88-108.

Brink, D., Roos, L., Weller, J., & van Belle, J.-P. (2006). Critical success factors for migrating to OSS-on-the-desktop: Common themes across three South African case studies. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, & G. Succi (Eds.), *IFIP International Federation for Information Processing* (Vol. 203, pp. 287-293, open source system). Boston: Springer.

Canonico, P. (2005). Deploying open source applications within the public sector domain: Preliminary findings on potential organisational benefits and drawbacks. In G. Mangia & R. Mohr (Eds.), *Proceedings of the German-Italian Workshop on Information Systems (GIWIS 2005)* (pp. 85-92).

COSPA Project. (2005). *Work package 4, deliverable 4.3—Experience report on the implementation of OS applications in the partner PAs.* Retrieved July 4, 2006, from http://www.cospa-project.org/ download_access.php?file=D4.3-ExperienceReportOnTheImplementationOfOS.pdf

Dedrick, J., & West, J. (2003). Why firms adopt open source platforms: A grounded theory of innovation and standards adoption. In J. L. King & K. Lyytinen (Eds.), *Proceedings of the Workshop on Standard Making: A Critical Research Frontier for Information Systems* (pp. 236-257).

DeLone, W. H. (1988). Determinants of success for computer usage in small business. *MIS Quarterly*, *12*(1), 50-61.

Drozdik, S., Kovács, G. L., & Kochis, P. Z. (2005). Risk assessment of an open source migration project. In M. Scotto & G. Succi (Eds.), *Proceedings of the First International Conference on Open Source Systems* (pp. 246-249).

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532-550.

Fitzgerald, B. (2005). Has open source software a future? In J. Feller, B. Fitzgerald, S. Hissam, & K. Lakhani (Eds.), *Perspectives on free and open source software* (pp. 93-106). Cambridge, MA: MIT Press.

Fitzgerald, B., & Kenny, T. (2003). Open source software in the trenches: Lessons from a large scale implementation. In S. T. March, A. Massey, & J. I. DeGross (Eds.), *Proceedings of 24th International Conference on Information Systems (ICIS)* (pp. 316-326).

Ghosh, R., & Glott, R. (2005). *Free/libre and open source software: Policy support (FLOSSPOLS)—deliverable D3: Results and policy paper from survey of government authorities.* Retrieved August 5, 2005, from http://flosspols.org/deliverables/ FLOSSPOLS-D03 (MERIT, University of Maastricht)

González-Barahona, J. M., & Robles, G. (2005). Libre software in Europe. In C. Dibona, D. Cooper & M. Stone (Eds.), *Open sources 2.0* (pp. 161-188). Sebastopol, California: O'Reilly.

IDA—Interchange of Data between Administrations. (2003). *The IDA open source migration guidelines.* Retrieved October 23, 2003, from http://europa.eu.int/idabc/servlets/Doc?id=1983

Jashari, B., & Stojanovski, F. (2006). Challenges and obstacles: Usage of free and open source software in local government in Macedonia. In B. Özel, C. B. Çilingir, & K. Erkan (Eds.), *Proceedings of tOSSad OSS2006 Workshop: Towards Open Source Software Adoption: Educational, Public, Legal, and Usability Practices* (pp. 49-55). Kocaeli, Turkey: Tübitak.

Kovács, G. L., Drozdik, S., Zuliani, P., & Succi, G. (2004a). Open source software and open data standards in public administration. In *Proceedings of the IEEE International Conf. on Computational Cybernetics (ICCC2004)* (pp. 421-428).

Kovács, G. L., Drozdik, S., Zuliani, P., & Succi, G. (2004b). Open source software for the public administration. In *Proceedings of the 6th Computer Science and Information Technologies (CSIT)* (pp. 1-8).

Lee, J.-A. (2006). Government policy toward open source software: The puzzles of neutrality and competition. *Knowledge, Technology, & Policy*, *18*(4), 113-141.

Lundell, B., Lings, B., & Lindqvist, E. (2006). Perceptions and uptake of open source in Swedish organisations. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto & G. Succi (Eds.) *IFIP International Federation for Information Processing* (Vol. 203, pp. 155-163, open source systems). Boston: Springer.

Moon, M. J., & Bretschneider, S. (2002). Does the perception of red tape constrain IT innovativeness in organizations? Unexpected results from a simultaneous equation model and implications. *Journal of Public Administration Research & Theory*, *12*(2), 273-291.

Nye, J. S. (1999). Information technology and democratic governance. In E.C. Kamarck & J. S. Nye (Eds.), *Democracy.com? Governance in a networked world*. Hollis, NH: Hollis Publishing (pp. 1-18).

Phillips, L. W. (1981). Assessing measurement error in key informant reports: A methodological note on organizational analysis in marketing. *Journal of Marketing Research*, *18*(4), 395-415.

Rai, A., & Patnayakuni, R. (1996). A structural model for CASE adoption behavior. *Journal of Management Information Systems*, *13*(2), 205-234.

Rossi, B., Russo, B., & Succi, G. (2006). A study on the introduction of open source software in the public administration. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, & G. Succi (Eds.). *IFIP International Federation for Information Processing* (Vol. 203, pp. 165-171, open source systems). Boston, MA: Springer.

Rossi, B., Scotto, M., Sillitti, A., & Succi, G. (2005). Criteria for the non invasive transition to openoffice. In M. Scotto & G. Succi (Eds.), *Proceedings of the First International Conference on Open Source Systems* (pp. 250-253).

Russo, B., Zuliani, P., & Succi, G. (2003). Toward an empirical assessment of the benefits of open source software. In J. Feller, B. Fitzgerald, S. A. Hissam & K. Lakhani (Eds.), *Taking stock of the bazaar: Proceedings of the Third ICSE Workshop on Open Source Software Engineering* (pp. 117-120).

tOSSad. (2006). *F/OSS National Programme Start-Up Roadmap Report.* Retrieved June 28, 2006, from http://tossad.org/tossad/publications/ f_oss_national_programme_start_up_report__1

Vaca, A. (2005). Extremadura and the revolution of free software. In M. Wynants & J. Cornelis (Eds.), *How open is the future? Economic, social & cultural scenarios inspired by free and open source software* (pp. 167-197). Brussels, Belgium: VUB Brussels University Press.

Ven, K., & Verelst, J. (2006). The organizational adoption of open source server software by Belgian organizations. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, & G. Succi (Eds.), *IFIP International Federation for Information Processing* (pp. 111-122). Boston, MA: Springer.

Waring, T., & Maddocks, P. (2005). Open source software implementation in the UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management, 25*(5), 411-428.

Wichmann, T. (2002). *FLOSS final report—Part 1: Use of open source software in firms and public institutions—Evidence from Germany, Sweden and UK.* Retrieved September 8, 2003, from http://www.infonomics.nl/FLOSS/report/reportPart1_use_oss_in_firms_and_public_institutions.htm

Yin, R. K. (2003). *Case study research: Design and methods* (3rd ed.). Newbury Park, California: Sage Publications.

Zuliani, P., & Succi, G. (2004a). An experience of transition to open source software in local authorities. In *Proceedings of E-Challenges on Software Engineering.*

Zuliani, P., & Succi, G. (2004b). Migrating public administrations to open source software. In P. Isaías, P. Kommers, & M. McPherson (Eds.). *Proceedings of E-Society 2004 IADIS International Conference* (pp. 829-832). IADIS Press.

ENDNOTES

1. For a thorough discussion of the concerns involving the use of open source software by PAs, see Lee (2006).
2. Although this group is currently inactive, some information on it can still be retrieved at http://eu.conecta.it
3. http://europa.eu.int/idabc
4. See http://europa.eu.int/information_society/eeurope/2005/all_about/'action_plan/index_en.htm
5. http://europa.eu.int/idabc/en/chapter/452
6. http://www.cospa-project.org
7. http://www.flosspols.org
8. http://www.calibre.ie
9. http://www.tossad.org
10. See IDABC "open source case studies" (http://ec.europa.eu/idabc/en/chapter/470) and news (http://ec.europa.eu/idabc/en/chapter/491).
11. http://www.muenchen.de/Rathaus/referate/dir/limux/89256/
12. It must be noted that the results of these two studies cannot be directly compared. The first FLOSS study included commercial businesses as well as PAs, and included subjects from only 3 European countries.
13. The workaround is however still used, since it is a better approach for formatting these documents.
14. Actually, at the time of writing, Microsoft has announced its support to an independent open source project, developing an ODF plug-in (Open Document Format) for MS Word 2007 (http://odf-converter.sourceforge.net). Coincidently, this announcement was issued a number of days after the Belgian government announced that it would start using the ODF-format exclusively from September 2008. Other governments (e.g., the state of Massachusetts) have announced similar initiatives, or are studying them.

This work was previously published in Emerging Free and Open Source Software Practices, edited by S. Sowe, I. Stamelos and I. Samoladas, pp. 191-214, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 4.12
Issues and Aspects of Open Source Software Usage and Adoption in the Public Sector

Gabor Laszlo
Budapest Tech, Hungary

ABSTRACT

This chapter introduces L-PEST model as the proposed tool for better understanding the fields are influenced by motivations and adaptation policy on FLOSS of public authorities and governments. Software usage in the public sector is a highly complex topic. In the confines of this chapter the selected case studies will show consideration to the vastly different needs and capacities and the different approaches and motivations towards the utilization of FLOSS by governments and/or local authorities. The primary objective of this chapter is to identify and describe the actors associated to the usage of FLOSS within and by the public sector. This chapter has made an attempt to fill this research gap and place the different actors into one complex model. It is hoped the proposed model assists better clarifying the intricate relationship between relevant factors. Nevertheless, much more research work is needed in the years to come. According to Michel Sapin, French Minister in charge of Public Administration and e-Government (2001), "The next generation e-government has two requirements: interoperability and transparency. These are the two strengths of open source software. Therefore, I am taking little risk when I predict that open source software will take a crucial part in the development of e-Government in the years to come."

INTRODUCTION

The digital economy transforms governments and governments took on new roles in those areas of the economy most affected by technological changes. Governments play important roles in creating the proper environment for ICT development, and also have a significant leading role as users of these technologies by creating new modes of public's behavior. Governmental functions and

operations can be managed only by the extensive use of ICTs and by using software applications (Lanvin, 2003).

The world's largest consumers of computer software are usually governments and they thus can have considerable influence on the software market. Governmental usage of software can impact on virtually all aspects of civil life: the inclusion and participation of citizens in public life, the transparency and openness of decision making, and the elimination of the digital divide, digital persistence, and digital literacy. The question of which software is utilized by public administrations is, therefore, of fundamental importance. Free Software advocate Eben Moglen has said, "Who controls the software, controls life."[1]

In the early days of computing the common software model was based on the open source model. Software and hardware were often combined in a single package. The software was usually traded in the form of source code and computer users have shared their computer code. Many important early programs, also with government funding, were widely shared (Bessen, 2002).

Then, the late 1970s and early 1980s with the appearance the consumer computing saw the beginning of the commercialization of software products based on the proprietary model. The software that operates the hardware has become as important as the hardware itself.

A significant difference between open source and proprietary software is that the open source (as it is called) software source code is freely available to the user. In contrast, the proprietary software vendors release their product only in binary form and it is illegal for end users to decompile the binary machine code to usable source code.

Free/libre open source software (or FLOSS as it is commonly referred to) has gained enormous momentum all over the world. While this movement has been closely followed with attention by many advocates and practitioners, academic research on the subject has only started emerging. These research projects have focused mainly on individual motivations, knowledge sharing and the user communities themselves.

The primary objective of this chapter is to identify and describe the factors related to the usage of open source software within and by the public sector.

To achieve this objective, background is given on the discussion about government roles and policies towards open source software, as in the selected case studies.

One of the strengths of this chapter is that it presents a theoretical framework, a general model of software usage at large within the public sector and the identified factors assigned to global perspectives.

BACKGROUND

ICTs have the capacity to play a valuable role in improving the quality of life, particularly in health, education, agriculture, and the environment. To take one example, in the healthcare sector ICTs enable the implementation of tele-health programs in remote areas, allowing some health care to be provided remotely, independent of person-to-person contact. Further, improvements in medical equipment are also a result of advances in ICTs. In education, remote access to the knowledge bases, e-libraries and even e-learning systems and universities can deliver knowledge to rural areas, where such opportunities for learning would be unavailable without ICTs. Agriculture and environmental issues can be better managed by, for example, geographic information system (GIS) and weather forecasts.

However, at the same time, there exists the so-called digital divide, an umbrella term that is commonly understood to mean the gap between ICT *haves* and *have-nots*. Generally, the approach to the question of the capacity of ICT to increase standards of living and to that of the digital divide has focused on two main issues.[2] One focuses mainly on actual connectivity—infrastructure

and access. Another approach beyond connectivity is to consider the level of the ICT literacy and skills of a particular population and as well, consequently take into consideration political and social cohesion aspects.

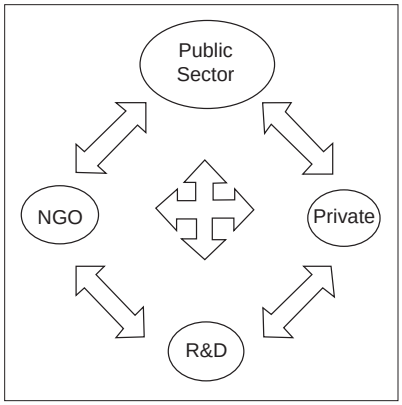
An improved economy can not alone eliminate the gap, so governmental "intervention" is a prerequisite for overcoming the digital gap. Today, governments, businesses, international groups, and nongovernmental organizations (NGOs) have undertaken numerous initiatives aimed at eliminating this digital divide (http://www.bridges.org/digital_divide). These initiatives have targeted not just the consequences of economic differences between countries and peoples and the relevant differences in access to technologies, but also the cultural capacity and political will necessary to apply these technologies for effective development. A nation's intellectual capital and capacity for innovation are based on its human capital, which is why it is so important for governments to make steps to strengthen the equality.

Wilson pointed to a four-sided social formation—a Quad—that has emerged at the heart of the still-inchoate knowledge society. "Conceptually 'quad' refers to persistent four-sided networked interactions of small groups of elites across four sectors of the political economy—government, private sector, research centers, and NGOs" (Wilson, 2003, p. 6).

The Quad theory predicts causal relationships between the architecture of the Quad and the subsequent performance of the ICT sector. The more robust the architecture of the Quad, the better performance of the ICT sector as a whole. The architecture and dynamics of the Quad relationships are different in every country and change time to time.

As a member and part of the Quad, the government has a special obligation to protect the integrity, confidentiality and accessibility of public information, to protect the privacy of its citizens, to educate the "next generation", to create jobs, and to preserve and make available the national heritage (also in electronic format) for the public and for the next generation. Other important roles for the governments are to make the country competitive in the globalized marketplace, and to carefully manage the budget (Stanco, 2003).

Figure 1. The Quad (Source: Used with permission by E. J. Wilson)



METHODOLOGY

This chapter provides an inductive general conceptual model—based on known and publicly available strategy documentation of various public sector and government initiatives for promoting or using FLOSS. The selection of key factors is grounded in available research literature on FLOSS and the above mentioned documentation and case studies.

OPEN GOVERNMENT

The average citizen has limited access to important government records, and what is available is often incomprehensible. An open government must be transparent and accountable and information related to the decisions an open

government makes must be open to the public and freely available. Access to government and public information is regulated by law in many countries. Perrit states:

Freedom of information issues are centrally important in countries around the world, and the Internet's World Wide Web offers the potential to provide freedom of information at low cost. Achieving a sound information policy to promote open government requires constant vigilance by those who care about the goal. (Perrit, 1997, p. 397)

In the aftermath of 9/11 the relationship between IT, governments and their citizens has dramatically and radically changed. Security has become the most important factor. Yet, in the face of increased demands for security, for many within societies, the demands of privacy and trust remain paramount, thus giving rise to conflict between governments and their citizenry. Governments make greater efforts based on anti-terrorism legislation[3] to monitor their citizens' activities, while simultaneously citizens demand a greater ability to monitor the activities of his or her government.

INITIATIVES FOR FLOSS IN GOVERNMENT WORK

E-government work and what is commonly understood as general government work are now too closely intertwined to be realistically separated. At the same time, public administrations have special functions and operations which cannot be adequately handled with proprietary software applications on the market that are developed for multiple purposes (Stanco, 2003). The moderate opinions which stress that there is no need to make a choice between FLOSS and proprietary software vendors gather ground but the feasible solution is mixing these software.

It seems likely that all governments use FLOSS applications on some level, with or without open source label—though perhaps without deliberate policy. Whereas many governments have policies or consideration towards FLOSS usage, the motivations may vary from cost reduction to security or dependency issues and within the broader context of policies to support such issues as equity or education. However, FLOSS policies and legislation as developed by national, regional or local governments around the World (USA, Canada, Australia, many countries in Africa or in Europe) are more often than not inadequate to support the viable realization of such policy goals.

The Center for Strategic & International Studies (2004) maintains—Government Open Source Policies—a list of such initiatives that were approved or proposed. This section highlights different approaches of adaptation and policy considerations for the implementation of FLOSS.

European Union

In the recent years many open source-related programs have been launched by the European Union. Fields of development of FLOSS within the EU include security, interoperability and e-participation. The software usage and the interaction between different systems is a complex approach. Interoperability is one of the key factors. One early Commission Working paper stressed the need for interoperability of program for public administration across the EU. It states that the proposed interoperability framework "will be based on open standards and encourage the use of open source software." "Interoperability, therefore, for both the public and enterprise sectors, is at the heart of the eEurope 2005 Action Plan and the achievement of the Lisbon goals" (Linking up Europe, 2003, p. 5).

In the European Union, the public sector were advised to avoid proprietary document formats, known as lock-in. Using the open standards would assure the desired interoperability and open stan-

dards would more greatly be supported by open source software. Using interoperable systems would guarantee equality among the citizens using different kind software applications (Promotion of Open Document Exchange Format, 2003).

On the other hand, notwithstanding the above-mentioned initiatives, the relationship between governments and open source is not unambiguous.

Extremadura

Extremadura was the poorest region of Spain, lagging behind the rest of the country in both the economic and technological field. Though short on financial resources, the region set very high goals for itself in its Regional Strategy on Information Society in 1997. The policy lay "in the application of technological innovation for the promotion of freedom and equal opportunities, taking advantage of and putting at the disposal of everyone, what is nobody's property: the knowledge gathered by Humanity all through History." Two formal strategic objectives were put forth: "Accessibility for all—the Internet as a public service" and "The stimulation of technological literacy."

Given the combination of Extremadura's strategic goals and the limited financial resources available, the use of FLOSS was a logical choice. The LinEx project, a combination of "Linux" and "Extremadura," was born of these strategic initiatives. The objective of the Linex project was to create a fully functional platform, based on FLOSS, providing universal access of IS tools to all citizens. While doing so, its aim was to provide adaptability, economic benefits, and security to as great a degree as possible, without losing sight of actual feasibility. LinEx is specifically designed for use in regional administration and schools. Early on in the project, it was decided that LinEx would not innovate the software itself, but rather concentrate on localization of the software and take care of the distribution. To avoid technical problems during the initial phase of the project, a Spanish company was hired. The region's government ships the resulting software for free to all of its citizens.

Extremadura was also simultaneously funding a development center whose task was to create accounting software, hospital applications and agricultural applications (IDABC, 2003).

Munich

Coming after the switch to Linux in the servers of the Bundestag in 2002, Germany's interior minister signed an agreement with IBM to offer the German government offices deep discounts on computer systems based on Linux (IBM signs Linux deal with Germany, 2002). Soon afterward, Germany's third largest city government, Munich, commissioned Client Study for the State Capital Munich (UNILOG Integrata, 2003) comparing the alternatives and assigning 6,218 (out of 10,000) points to a Linux/OpenOffice migration, versus 5,293 to an upgrade of Microsoft Windows. Based on this study the Munich municipal government made a decision to adopt for their computer systems open source software. The Council of Munich voted on May 2003 in favour of the adoption for its desktop and notebook computers an open source operating system and office applications. This move, unprecedented in scale in the European public sector, has been widely commented upon and discussed since then.

Following a test phase conducted in cooperation with SuSE Linux and IBM, the Council formally adopted on June 16, 2004 detailed plans to manage the migration process, which is expected to last until 2009. According to the plan the migration was to be gradual, starting in 2004 with office desktop applications (OpenOffice.org office suite and Mozilla Web browser running on the existing Windows NT desktops), and then moving to operating systems and more specialized applications over a period of five years. The municipal government of Munich released a statement in September 2005 that the completion

of migration phase one, scheduled to be completed in 2005, had been pushed back to at least 2006. The reasons were that Novell Inc. announced in late 2003 the acquisition of SuSE and meanwhile legal problems regarding a proposed EU patent law. The chosen new Linux distribution was the Debian (Grassmuck, 2005).

USA

The most famous report concerning FLOSS usage within the Department of Defense (DOD) was released in 2003. "The goals of the study are to develop as complete a listing as possible of FOSS applications used in the DoD, and to collect representative examples of how those applications are being used." Over a two-week period the survey identified a total of 115 FLOSS (in the report named as FOSS) applications and 251 examples of their use. "The main conclusion of the analysis was that FOSS software plays a more critical role in the DoD than has generally been recognized. FOSS applications are most important in four broad areas: Infrastructure Support, Software Development, Security, and Research" (The MITRE Corporation, 2003, p. 2).

The Commonwealth of Massachusetts launched a new policy regarding the planning, development, and implementation of IT systems. "The goal of the Commonwealth's open initiatives is to ensure that investments in information technology result in systems that are sufficiently interoperable to meet the business requirements of its agencies and to effectively serve its constituencies" (Open Initiatives of Massachusetts, n.d.). The Massachusetts case illustrates the technology based considerations concerning software usage.

Brazil

The Government National Institute of Information Technology is charged with implementing open source software in Brazil. They released the first strategy planning document in 2003.

On the surface, the decision of the Brazilian government was a simple cost cutting measure. According to the National Information Technology Institute, Brazilians spend $1.1 billion every year on software licensing fees, and the federal government was the nation's biggest customer. The government is paying around $500 to Microsoft for license fees for every workstation. The government accounted for 6% of Microsoft's 2003 Brazilian revenues of $318 million. Switching to FLOSS would save millions of dollars (Kim, 2005). The decision to migrate to open source software on a national scale was not simply a matter of choosing one product over another. Although the Brazilian government identified economic reasons to migrate to open source software, it was a political decision that validated open source software as a movement. Through numerous open source projects, the government has tried to bridge the technology divide within the Brazilian population. While in the European Union the research experts recommend free software licenses for software deriving from public funds, Brazil has become the first country to require any company or research institute that receives government financing for the development of software to license it as open-source, meaning the underlying software code must be free to all (Benson, 2005).

Peru

Peru passed a law encouraging the procurement of free software by the government in September 2005. The bill was originally introduced in 2002. A Peruvian congressman stated in his letter to Microsoft: "The basic principles which inspire the bill are linked to the basic guarantees of a state of law, such as: the free access to public information by the citizen; the permanence of public data; the security of the state and citizens" (Greene, 2002). This bill has as its aim to establish measures and policies which will permit the acquisition of software licenses by the public administration under

conditions of technology neutrality, and the free concurrence and equal treatment of suppliers. The technical evaluation of the software and hardware required by the public administration will be according regulations dictated by the National Informatic System governing body. The bill offers an excellent summary of the idea of neutral software usage: "The entity will ensure that the procurement answers to the principles of effectiveness and technological neutrality, transparency, efficiency, within the boundaries of austerity and economizing of public resources" (Peruvian bill translation, 2005). One essential item included in the bill also stress the need for the education of the employees and users of computer and IT technology.

South Africa

One of the best-known case studies concerns the South African government's official strategy for FLOSS. This was one of the first strategies, that officially recognized the legitimacy of the adoption of FLOSS within the public sector. The South African strategy highlights that "the government will implement OSS, where analysis shows it to be the appropriate option. The primary criteria for selecting software solutions will remain the improvement of efficiency, effectiveness and economy of service delivered by the government to its citizens" (Using Open Source Software in the South African Government, 2003, p. 24). One of the main strengthens of this strategy is the appreciation of the social benefits that could include, but are not limited to, better education, greater governmental transparency, more effective e-government services, and wider access to governmental information.

China

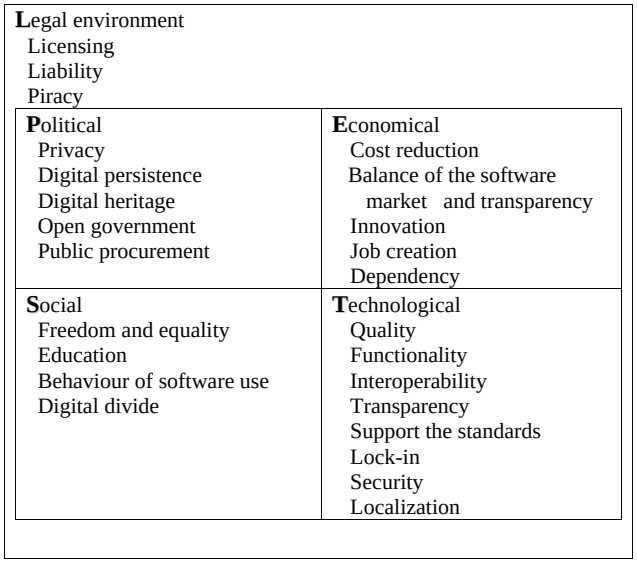
China has been very aggressively promoting Linux. The military has been one of the earliest adopters of Linux. The Red Flag Linux was designed for use in government offices, schools and on home computers. Red Flag Linux, a Beijing-based provider of Linux software and services, is connected to the Chinese Academy of Sciences, the central government's top research institute. The main reasons for the adoption of Linux were political and the desire for independency from Microsoft (Einhorn, 2003). Membership in the World Trade Organization (WTO) and access to its benefits are strongly affected by the level of protection given to intellectual property rights in a country (Wong, 2004). According the Piracy Study (BSA, 2005), in the country there is a high frequency of pirated software. Since China became a full member of the World Trade Organization, the government has been trying to reduce software piracy within its country. This is another strong reason why government agencies and business are currently adopting the Linux operating system on their desktop workplaces.

L-PEST MODEL

As is shown in the above selected case studies there are many different approaches around the world to using FLOSS within the public sector. In this section a general model is introduced. The L-PEST model is a theoretical creation. The idea of the model reaches back to IDABC "The Many Aspects of Open Source" (n.d.) material. The original text summarized some of the various reasons for choosing different organizations of FLOSS. This idea was then extended, modified, and put into a model based on the research by the author. The author's proposed L-PEST model can give a broader picture as to the aspects of software usage in the public sector. With further and ongoing work, it can be applied to all kinds of software as a comprehensive tool.

The key factors were derived from motivations of governments within their environments, which were revealed in the case studies. The five key actors of the model—as shown on Figure 2—are:

Figure 2. L-PEST model



political, economic, social, and technical and all around these fields the legal environment can be founded. This structure maps real life.

Every actor has its own attributes, however in some cases there are attributes with different meanings. In this case, when an attribute could be assigned to more than one field it was put into the most characteristic actor (e.g., lock-in which is based on the technical elements may well also influence the economical aspect, or transparency has quite a different meaning in economics than in technology).

Legal Environment

The legal environment surrounds the model because it has an effect on the other four factors. It has own attributes as well. The constitution determines the framework activity of the country; the law determines the operation of society, while the economy is influenced by the law of economics. Acts regulate and ensure competition, and technology is also regulated by industrial law (including, patents, trademark, and copyright law).

Copyright is the most usual method of protection for software products. The copyright automatically and implicitly protects all intellectual creation, including computer software.[4] "The copyright laws, by default, do not allow for redistribution (nor even use) of software. The only way that redistribution can be done is by granting specific permission in a license (Working group on Libre Software, 2000, pp. 20-21). A license is a contract between the user and the licensor. The licensing model of about FLOSS differs from the proprietary software, but is based on same idea. In fact, open source licenses are also enforceable because they use, in one form or another, copyright law. Most open source licenses were designed according to the United States law. Open source (OSS) licenses are more permissive than free software (FS) licenses.[5]

One of the main threats for open source may be software patents—which are not currently

common outside of the USA—but efforts to introduce them are in progress worldwide, usually lobbied for by large multinational corporations. The issue of software patents[6] divides even the governments of countries and the parliaments within them, as can be seen in many cases in Europe. On the other hand, many companies which have huge software patent portfolios in the USA, such as IBM and Novell, open numerous of their owned patents and put them at the OSS developing community's disposal. At the same time, many companies adopt and encourage OSS policy and business model.

Liability means that the software producers are responsible for their own products, warranties, and indemnifications. In reality almost all kinds of software, even the proprietary kind, are shipped on an "AS IS" basis, which means that the producer wriggles out of any kind of responsibility. In many countries, legislation does not allow the exclusion or limitation of this kind of liability.

Software piracy is a problem all around the globe and it can hurt a country in many ways. A country with poor protection for intellectual property rights is not as attractive to foreign investors. This is the reason why China, since joining the World Trade Organization (which strongly defends and pursues intellectual property rights protection), has made enormous efforts to reduce the prevalence of piracy within its borders. In a developing country, piracy is much more prevalent than in the industrialized nations; however, the greater dollar losses are incurred in the latter situation (BSA, 2005).

Political Aspects

The political aspect is related to government's function and roles. They may be distinguished between such roles as promoting social justice and functions such as tax collection (Lanvin, 2003). The government's role ensures the viable environment for ICT development and also the ICTs for Development. This can be summarized as a National Information Strategy that was well defined by the Library and Information Association of New Zealand:

A National Information Strategy addresses strategic issues to ensure that all citizens have the opportunity to access and utilize a nation's knowledge wealth in a way that will enhance the social, political and economic well-being of that country. It states the government position on the creation, management and use of information, and sets direction for government action in support of the strategic goals. (LIANZA, 2002, p. 7)

A national information strategy can be defined in terms of political planning or political action planning for development.

Privacy is a key factor in the interaction between governments and citizens. Whatever software is utilized by governments to control, manage and transmit the citizenry's personal data must be transparent in order to protect the citizen's right to privacy (Stanco, 2003). For example, an e-voting system without transparency leaves organizations and governments at the mercy of software providers.

The preservation of digital heritage and digital content has become a major challenge for society.

Digital persistence means continued accessibility to the stored content, even as the technology is changed—in this case for the governments' and public administrations' documentation. (It also preserves the original documents, in the case of national heritage.) It is in close relation with lock-in and dependency that it will be introduced. The secretary of administration and finance of the Commonwealth of Massachusetts, stated:

Our public policy focus is to insure that public records remain independent of underlying systems and applications, insuring their accessibility over very long periods of time. In the IT business a long period of time is about 18 months. In government

it's over 300 years, so we have a slightly different perspective. (Kriss, 2005)

Economic Aspects

Within the scope of this chapter only several issues can be highlighted. Governments sometimes need to undertake intervention into the market on behalf of common good. A high degree of market transparency can result in disintermediation due to the buyer's increased knowledge of supply pricing. Transparency is important since it is one of the theoretical conditions, which reaches back to Adam Smith's invisible hands theory, required for a free market to be efficient. Consequently, it may well be true, that the government should not intervene in the free market except to assure neutrality and a level playing field for all types of software. The governments should to be assuring the neutral decision on software and public procurement and choices on software products should be made objectively, flexibly, and with a focus on a range of factors.

One of the primary economic concerns is the cost of software usage. Total cost of ownership (TCO) shows the real cost of software utilization. The purchased software will usually remain the property of the supplier; the consumer pays for the right to use the software. Total costs need be divided into two main categories, direct and indirect costs. The measurement is difficult because the indirect costs are extremely difficult to assess and measure (Wheeler, 2005). Another approach to the issue of value and cost can be to focus on the examination of return on investments (ROI). Both methods are extremely sensitive to the set of assumptions made by the individual or group taking the measure.

Research and development (R&D) and other innovation are more important than ever before. In their role as a member of the Quad (see Figure 1), governments should undertake to stimulate innovation. The economic benefits of such stimulation, as in the case of job creation, for example, are well known. There are numerous arguments that R&D that is financed through public funding should be released under FLOSS license. This kind of license supports the sharing of scientific results and dissemination of created information and value—and "there is not need to reinvent the wheel." Many FLOSS licenses are business friendly (Wong, 2004).

One of the major arguments in favor of FLOSS is concern over the issue of dependency; that is, the public becomes reliant on software suppliers. In many instances, there are painfully few options as to software vendors. Beyond the issue of economic costs incurred from near monopoly, the question of dependency also speaks to the issues of security and privacy protection.

Social Aspects

The ICTs have a huge potential to make life better, despite the consequences of the so-called digital divide. The dual societal pursuits of freedom and equality are furthered via the ability of citizens to access the information and services of national and municipal governments. The goal of open, transparent government is dependent upon ever-greater access that ICTs offer. Meanwhile, the choices governments make as regards open or proprietary software, and the value they place on either, act as an example to the public, as well as reflecting the governments' position vis-à-vis issues such as privacy and security.

Education, of course, also greatly impacts on the economic development and potential of a country. Governments, of course, play a major role in creating a proper environment for education. Digital literacy and elimination the digital divide are close correlation with education. In educational systems there are two major expenses related to software: in using proprietary software, schools must buy licenses for every single computer that uses the software, while at the same time, the school has to ensure the possibility that students do not abuse its use after the class.

Many NGOs are not able to afford commercial, non-pirated software. This is a compelling reason to seriously consider FLOSS as a viable option for NGOs. An excellent example of the benefits of open source is the "Human Rights Tool" open source software named Martus (http://www.martus.org/). The developer uses the new model of social entrepreneurship, which combines market forces with philanthropic capital and entrepreneurial drive. Social entrepreneurship as a focus of academic research has a relatively brief history and as yet no research has been made on the connection with FLOSS communities and businesses.

Technical Aspects

The measurable technical parameters are, among others, the reliability, performance and scalability of the systems. These parameters can be compared using the same technical analyses (Wheeler, 2005). The quality of a software product is a controversial field. Functionality of software means the software functions fit the users demand and requirements.

In technical context interoperability is used to describe the ability of different software and hardware form different vendors to exchange data and utilize the same protocols and to operate effectively together. If the competitors' products are not interoperable, the result may be monopoly. To avoid the vendor lock-in, it may be prudent for governments to take steps to encourage interoperability in various situations.

Transparency refers to the fact that, when software is developed, the original source code is available (or not) to public (or user) review. The government is responsible for storing a large amount of data in name of the public. *Lock-in* means that if the data is stored in closed format using proprietary software, the information will only with difficulty be available and retrievable for many decades to come. Since FLOSS and open standards make available the source code, the way in which information is stored is publicly known, or at least traceable. Lock-in can be only avoided by using open standards. Moreover, lock-in may also refer to education where the brand-specific trainings confine the students and users.

Security is one of the main issues when software is used by governments and public administrations. Computing is crucial to the infrastructure of countries. Nowadays the information environment is extraordinary complex and fragile. Modern society is increasingly vulnerable in its technological and economical infrastructure, in its telecommunications, its energy sources, and its transportation. The infrastructure and information systems can be attacked, destroyed, disrupted, and corrupted by small groups or even single individuals. It is not necessary to destroy the infrastructure in its entirety, nor to attack it physically via traditional means: it can be crippled electronically, and virtually anonymously (Steele-Vivas, 1996). This vulnerability is a reason why the choice of software used is relevant and important. This refers also to the political actions.

Countries where English is not commonly spoken face a serious disadvantage when it comes to the uptake and dissemination of ICTs. However the translation is one of the major parts of localization, moreover, localization involves the task for adapting and customizing the products for local users' specific cultural and/or technical needs.

FUTURE TRENDS

The consideration and utilization of FLOSS by national and municipal governments will continue to grow in the coming years. One of the main fields where FLOSS can best be utilized is in the e-government services increasingly in demand.

A related area where FLOSS can be adopted is within Public Authorities, which are quite different in the each country and which therefore require the flexibility of localization, which

FLOSS affords. Another main issue where FLOSS is already utilized with success regardless of cost consideration is in healthcare, which is one of the costliest segments within governmental services around the world. FLOSS can also improve the performance of healthcare services, whilst ensuring both interoperability and patient privacy.

Around the world, governments are developing e-voting systems, but the resistance to these systems by citizens link back to a lack of the trustworthiness of closed systems, which can be avoid by using freely available source code.

Countries in the developing world can gain the possibility to use high-quality free software as opposed to scaled-down versions of more costly proprietary software.

CONCLUSION

Information and communication technologies have drastically changed societies, influencing the everyday activities of both individuals and governments. The information society has become a reality and acted as a call to action by governments. Although much research has been done on the use and consequence of FLOSS in the public sector, not enough knowledge exists on public sector and government policy options and behaviour as regards the adoption of this software. In addition, there are numerous negative perceptions and misunderstanding about FLOSS.

This chapter has made an attempt to give a comprehensive overview of the different fields and aspects relevant to governments and peoples that are influenced by the choice of software that governments make. Another aim has been to delineate the relationship between these related issues and factors. FLOSS touches upon multiple areas as it was introduced in the paper, using the L-PEST model. Beyond a well-known cost consideration, the case studies and the proposed model showed the FLOSS and open standards could afford a workable social-economic-technological solution.

In using the model, it has become clear that the utilization of and decisions regarding software adoption, there are numerous factors that could, and should have impact on software decisions. Within networked societies, interconnectivity was the first step, and nowadays interoperability has gained emphasis. There are numerous arguments that software application that is financed through public funding should be released under FLOSS license. It is not enough that this software is freely available at no cost. With the freely available source code there is the opportunity for improved quality, while simultaneously avoiding lock-in and the development for only one platform. This can bolster the elimination of the digital divide and help foster participation and inclusion programs.

As it was introduced in the Quad theory, the relationships between the Quad's elements determine the performance of the information society and development as a whole. The members of the Quad are involved in the different categories of the L-PEST model.

Much empirical and theoretical work is still needed in this field and in reference to the presented model as well as a better graphical representation of the model. Future research will focus on a detailed examination of motivations and a more precisely defined analysis of every factor involved. In reference to Wilson's Quad model, it might be interesting to investigate stakeholder analysis in contrast of the Quad and L-PEST model.

Protagoras, Greek philosopher (c 485- c 410 BC) said: "There are two sides to every question." And this case there do exist disadvantages in the utilization of FLOSS. It should be noted that the advantages and disadvantages can be measured and evaluated in relation to those incurred by using proprietary software. This model has considered general recommendations focusing on FLOSS but also makes possible comparison between proprietary and FLOSS software.

ACKNOWLEDGMENT

The author wishes to thank Nora for her encouragement and patience throughout the duration of research and writing process, Matthew Strauss for his valuable feedback and improvement of wording of the manuscript, and the anonymous reviewers for their helpful comments and all those who provided advice and suggestions on earlier versions.

REFERENCES

Benson, T. (2005, March 29). Free software's biggest and best friend. *The New York Times*, p. C1.

Bessen, J. (2003). What good is free software? In W. R. Hahn (Ed.), *Government policy toward open source software* (pp. 12-33). Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

BSA. (2005). *Piracy study.* Retrieved January 8, 2006, from http://www.bsa.org/globalstudy/upload/2005-Global-Study-English.pdf

Einhorn, B. (2003). Why Gates opened windows in China. *Business Week Online*. Retrieved December 12, 2005, from http://www.businessweek.com/technology/content/mar2003/tc2003033_6406_tc058.htm?tc

Grassmuck, V. (2005). LiMux: Free software for Munich. In J. Karaganis & R. Latham (Eds.), *The politics of open source adoption (POSA) Version 1.0* (pp. 14-36). Social Science Research Council [Electronic document]. Retrieved June 3, 2005, from http://www.ssrc.org/wiki/POSA/index.php?title=Main_Page

Greene, T. C. (2002). MS in Peruvian open-source nightmare. *The Register*. Retrieved November 17, 2005, from http://www.theregister.co.uk/2002/05/19/ms_in_peruvian_opensource_nightmare/

IBM signs Linux deal with Germany. (2002). *BBC News.* Retrieved December 2, 2005, from http://news.bbc.co.uk/1/hi/business/2023127.stm

IDABC. (2003). *FLOSS deployment in Extremadura, Spain.* Retrieved December 12, 2005, from http://europa.eu.int/idabc/en/document/1637

Kim, E. (2005). F/OSS adoption in Brazil: The growth of a national strategy. In J. Karaganis & R. Latham (Eds.), *The politics of open source adoption (POSA) version 1.0* (pp. 53-59). Social Science Research Council [Electronic document]. Retrieved June 3, 2005, from http://www.ssrc.org/wiki/POSA/index.php?title=Main_Page

Kriss, E. (2005). Informal comments on open formats. *Mass.gov.* Retrieved December 21, 2005, from http://www.mass.gov/eoaf/open_formats_comments.html

Lanvin, B. (2003). Leaders and facilitators: The new roles of governments in digital economies. In S. Dutta, B. Lanvin, & F. Paua (Eds.), *The global information technology report 2002-2003—Readiness for the networked world* (pp. 74-83.). Oxford: Oxford University Press.

LIANZA (Library and Information Association of New Zealand). (2002). *Towards a national information strategy.* Retrieved October 10, 2004, from http://www.lianza.org.nz/text_files/nis_7nov02.rtf

Linking up Europe: the Importance of Interoperability for eGovernment Services. (2003). *Commission Staff Working Paper, Commission of the European Communities.* Retrieved October 22, 2005, from http://europa.eu.int/idabc/en/document/2036/5583

Open Initiatives of Massachusetts. (n.d.) *Mass.gov.* Retrieved December 6, 2005, from http://www.mass.gov/open_initiatives

Perritt, H. H., Jr. (1997). Open government. *Government Information Quarterly, 14*, 397-406.

Peruvian bill translation. (2005). Retrieved January 8, 2006, from http://www.apesol.org/news/199

Promotion of Open Document Exchange Format. (2003). *IDABC European eGovernment Services.* Retrieved November 28, 2005, from http://europa.eu.int/idabc/en/document/3428/5890

Stanco, T. (2003). *US: Testimony.* Retrieved October 1, 2005, from http://www.egovos.org/Resources/Testimony

Steele-Vivas, R. D. (1996). Creating a smart nation: Strategy, policy, intelligence, and information. *Government Information Quarterly, 13*, 159-173.

The Center for Strategic & International Studies. (2004). *Government open source policies.* Retrieved September 14, 2005, from http://www.csis.org/media/csis/pubs/040801_ospolicies.pdf

The Many Aspects of Open Source. (n.d.). *IDABC European eGovernment Services*. Retrieved January 19, 2005, from http://europa.eu.int/idabc/en/document/1744

The MITRE Corporation. (2003). *Use of free and open source software (FOSS) in the U.S. Department of Defense*. Retrieved October 22, 2005, from http://www.terrybollinger.com/dodfoss/dodfoss_pdf.pdf

UNILOG Integrata .(2003). *Client study for the state capital Munich*. Retrieved January 5, 2006, from http://hdl.handle.net/2038/490

Using open source software in the South African government. (2003). Retrieved September 19, 2005, from http://www.oss.gov.za/docs/OSS_Strategy_v3.pdf

Wheeler, D. A. (2005, November 14). *Why open source software / free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers!* Retrieved December 12, 2005, from http://www.dwheeler.com/oss_fs_why.html

Wilson, E. J., III. (2003). *Forms and dynamics of leadership for a knowledge society: The Quad*. Retrieved June 14, 2005, from http://www.cidcm.umd.edu/wilson/leadership/quad2.pdf

Wong, K. (2004). Free/open source software. Government policy. *International Open Source Network, Elsevier.* Retrieved October 12, 2005, from http://www.iosn.net/government/foss-government-primer/foss-govt-policy.pdf

Working group on Libre Software. (2000). *Free software/open source: Information society opportunities for Europe?* Retrieved November 5, 2005, from http://eu.conecta.it/paper.pdf

KEY TERMS

Dependency: In this context, dependency means that the users are dependent on the software vendor for products and services so that he or she cannot move to another vendor without substantial cost.

Interoperability: Means the ability of systems to operate effectively together independently of different software or hardware vendors.

Localization: Means more than simply the translation of software; it refers to the customization of the software for local needs and demands.

Piracy/Copyright Infringement: The software piracy refers to the duplication, distribution, or use of software without the permission of the copyright holder.

Return on Investment (ROI): Generally, a ratio of the benefit or profit received from a given investment to the cost of the investment itself. This approach also focuses on the benefits and the measurement of the value of making an investment, not only the cost savings.

Total Cost of Ownership (TCO): A financial estimate for such things as (but not limited to) computer software or hardware. TCO is commonly used to support acquisition and planning decisions for a wide range of assets that bring significant maintenance or operating costs across a usable life of several years or more. TCO analysis is not a complete cost-benefit analysis. It pays no attention to business benefits other than cost savings.

Transparency: Transparency involves openness, communication, and accountability. In this context it refers to the fact that, when software is developed, the original source code is available (or not) to public (or user) review.

Lock-In, **Vendor Lock-In:** In technical terms it means that, if the data is stored in closed format using proprietary software, the information will only be available and retrievable with difficulty. The term also refers to dependency of different types of lock-in, such as when the users are 'locked-in' when trained for a specific technology or the dependency of the specific vendor.

ENDNOTES

1 "Who controls the software, controls life. Well, it had better us. That's the real political meaning of the free software movement, " said Eben Moglen, professor of law, General Counsel, Free Software Foundation at Open Source Conference, May 2004, Toronto.

2 The digital divide has a number of definitions and approaches. Examples can be found at: Bridge the Digital Divide (http://www.bridgethedigitaldivide.com/digital_divide.htm) Digital Divide Network (http://www.digitaldivide.net/)

3 Many governments passed anti-terrorism laws, aimed at enhancing security and facilitated the capture of terrorists. Global Policy Forum Web page (http://www.globalpolicy.org/empire/terrorwar/liberties/libertindex.htm) looks at cases where the "War on Terrorism" threatens civil liberties. The European Union ratified controversial data retention legislation (Directive 2006/24/EC) on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/EC). A week later on, EU and US representatives met for an informal high level meeting on freedom, security, and justice where the US expressed interest in the future storage of information.

4 Creative Commons has built upon the traditional copyright law based on the all-rights-reserved concept to offer a voluntary some-rights-reserved approach. The Creative Commons licenses provide a flexible range of protections and freedoms for authors, artists, and educators. http://www.creativecommons.org

5 FLOSS licensing approach based on differences between FS and OSS movement. The free software licenses do not allow closing"the source code while the permissive (OSS) licenses permit the creation of proprietary development. Philosophy on: "Why Free Software" is better than "Open Source" http://www.gnu.org/philosophy/free-software-for-freedom.html; Free Software licenses: http://www.fsf.org/licensing/; Open Source licenses: http://www.opensource.org/licenses/

6 More detailed reading on software patents in the European Union and other involved issues can be found at: Software Patents in the EU (http://www.oreillynet.com/pub/a/network/2005/03/08/softwarepatents.html) and Software Patents vs. Parliamentary Democracy (http://swpat.ffii.org/).

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 445-459, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 4.13
An Empirical Study on the Migration to OpenOffice.org in a Public Administration

Bruno Rossi
Free University of Bolzano-Bozen, Italy

M. Scotto
Free University of Bolzano-Bozen, Italy

A. Sillitti
Free University of Bolzano-Bozen, Italy

Giancarlo Succi
Free University of Bolzano-Bozen, Italy

ABSTRACT

The aim of the article is to report the results of a migration to Open Source Software (OSS) in one public administration. The migration focuses on the office automation field and, in particular, on the OpenOffice.org suite. We have analysed the transition to OSS considering qualitative and quantitative data collected with the aid of different tools. All the data have been always considered from the point of view of the different stakeholders involved, IT managers, IT technicians, and users. The results of the project have been largely satisfactory. However the results cannot be generalised due to some constraints, like the environment considered and the parallel use of the old solution. Nevertheless, we think that the data collected can be of valuable aid to managers wishing to evaluate a possible transition to OSS.

INTRODUCTION

Open Source Software (OSS) and Open Data Standards (ODS) emerged in recent years as a viable alternative to proprietary solutions. There are many cases in which the adoption of OSS has

proven advantageous for companies deciding to adopt it in replacement or in conjunction with closed solutions. The limitation of these migrations for our point of view is that they were very often server-side oriented and not supported by empirical evidence of the benefits of the new solution. In this sense, there are very few case studies that report successful transitions on the desktop side (ZDNet, 2005) and some are still underway (Landeshauptstadt München, 2003; Stadt Wien, 2004). It is our opinion that the reason of the apparent different results in the two fields is due to the nature of OSS development (Feller & Fitzgerald, 2001) that leads to repercussions on the resulting usability (Nichols & Twidale, 2003).

When comparing OSS and proprietary software and when comparing software solutions in general, it is impossible to get a global index referring to quality in order to compare two solutions (Fenton & Pfleeger, 1997).

If we consider the most important aspects under which it is significant to analyse software, as:

- Reliability
- Performance
- Price
- Security
- Interoperability
- Usability
- Extendibility
- Functionalities
- Privacy protection

The categories have to be balanced with the requirements of the environment and users in which the solution is deployed. Where the aspects of security, reliability, and extendibility are of key importance, OSS has proven a valid solution, if not superior to proprietary solutions. Where functionalities, usability, and in general user interaction acquires importance as on the client side, OSS has yet to prove as a valid alternative. Price is a controversial issue as there is the need not only to evaluate the license price but also the software maintenance and other costs inherited from the migration. These considerations originated the study we propose.

The purpose of the study is to evaluate in a rigorous way the introduction of OSS in a working environment, following the criteria of a controlled experiment from the selection of the sample to the evaluation of the results. We selected a sample of 22 users from different offices in the public administration target of the experiment. We divided the sample in two groups, one to be migrated, the other to be used as a control group. The results obtained seem to report that the initial reduction of productivity is not as consistent as we thought, also taking into account that half of the users considered the introduced solution as offering less functionality than the proprietary one.

STATE OF THE ART

There are many studies available evaluating the Total Cost of Ownership (TCO) of OSS. The original model derived from the work of the Gartner Group in 1987 and has since then been inserted in different models. The TCO model helps managers by considering not only the cost of purchase but also further costs as maintenance or training.

All the studies are not unanimous as the savings that can be reached with the adoption of OSS (Robert Frances Group, 2002; The Yankee Group, 2005). One of the reasons is probably the different weight given to costs and benefits that are difficult to measure. Two of such measures are, for example, the risks of lock-ins and the development of local economies. The risks of entering a mechanism of lock-in, for example, by relying only on a single software supplier or storing massive amounts of data by means of closed data standards are real and must be considered in a TCO model evaluating a transition (Shapiro & Varian, 1999). On the other side, the adoption of OSS can be of benefit to local software companies

that can exploit the possibility given by the available source code and open data standards. Also in this case, the amount of this kind of externality is difficult to quantify.

Considering OSS, there are many projects worth mentioning, we will name here two of the most famous and see how they perform on the market against proprietary solutions:

- the Apache Web server[1]
- the Mozilla Firefox Web browser[2]

Table 1 shows that the Apache Web server detains almost 70% of the whole market share (Netcraft Survey, 2005). As virus attacks of the last years have proven (CERT, 2001), one of the reason of such wide adoption is the security proposed by the Apache architecture.

Table 2 shows the market share of the Mozilla Firefox browser between January and April 2005. As it can be seen also in this case the software is gaining constant market shares in the last months. The software is still behind in market shares but it represents an important competitor for the market dominator, Microsoft Internet Explorer.

These are surely two of the most popular OSS that emerged during the last few years; there are many more that can compete with proprietary solutions. By looking at these and other examples, we can conclude that OSS already could represent an important alternative to proprietary software.

Another important consideration on OSS is represented by the cases in which a large migration has been performed or is in the process of being performed. Having a look at the different case studies available for the migration to OSS, we summarise the most famous during these years in Table 3; three are European, while one is U.S.-based.

Table 1. Web servers in September and October 2005 (Netcraft Survey, 2005)

Developer	September 2005	Percent	October 2005	Percent	Change
Apache	49598424	69.15	52005811	69.89	0.74
Microsoft	14601553	20.36	15293030	20.55	0.19
Sun	1868891	2.61	1889989	2.54	-0.07
Zeus	584598	0.82	585972	0.79	-0.03

Table 2. Browsers market share ("Browser Market Share Study," itproductivity.org, 2005)

Browser	January 2005 (%)	April 2005 (%)
Internet Explorer	84.85	83.07
Firefox	4.23	10.28
Mozilla	4.48	3.81
Netscape	3.03	0.92
AOL	2.20	0.85
MSN	0.58	0.67
Opera	0.34	0.41
Total	99.71	100.01

Table 3. Large scale migrations to OSS of public administrations

Region	Clients to migrate	Side	Distribution
Extremadura	80,000	Desktop/Servers	gnuLinex
Munich	14,000	Desktop	Debian
Vienna	7,500	Desktop	Wienux (Debian/KDE)
Largo, FL	900	Desktop/Servers	Linux KDE 2.1.1

One of the most remarkable deployments of OSS on the desktop side is surely the one of the Extremadura region in Spain, recently installing 80,000 Linux systems, 66,000 for the educational system and 14,000 for administrative workstations. The local administration created their Linux distribution called gnuLinex.[3] According to their IT department, the savings have been of the order of €18M (ZDNet, 2005). Another case of success is the one of the city of Largo, FL where the migration has involved 900 clients; the savings have been estimated at $300,000-$400,000 (Newsforge, 2002). The migration of the city of Munich and the one of the city of Vienna are currently underway (Landeshauptstadt München, 2003; Stadt Wien, 2004). As the delay of the Munich migration seems to demonstrate, a transition to OSS is not a process to underestimate. There are also cases where the proprietary solution has been considered as more convenient, like the city of Nürnberg, where according to their own migration study the transition from Windows 2000/Office 2000 to Windows XP/Office XP was considered €4.5M cheaper than the transition to Linux/OpenOffice.org (Stadt Nürnberg, 2004).

A final consideration on studies performed on OSS usability. Of certain interest for our study, albeit a little dated, is the experimentation conducted by the Berkeley University in November-December 2001 (Everitt & Lederer, 2001), comparing two different solutions in the office automation field, namely Sun StarOffice Writer 5.2 and Microsoft Word 2000. Authors report about an experiment on 12 users, regarding the user interface integration. As a result of the study, the two products were comparable, although the Microsoft solution proved to be more satisfactory and easier to use.

THE STUDY

Our study has been inserted into this framework; the intention is to contribute to the field with a solid and sound analysis of a real transition to OSS on the client side, specifically the analysis of a migration in the office automation field in one public administration.

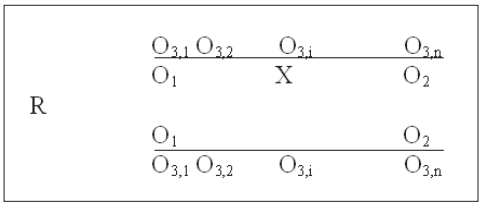
In particular the study related the introduction of the OpenOffice.org[4] suite. The suite offers comparable functions as the one offered by Microsoft Office.[5] It is composed of several applications, a text-processor, a spreadsheet, software for presentations, for drawing operations and for the creation of formulae. The only functionality missing in the version installed was the possibility to create small local databases. In the organisation where we performed the study, this was a feature rarely employed by users and, in general, deprecated by IT managers. We focused our analysis mainly on word-processors and spreadsheets.

The experiment was performed on 22 users of a public administration (PA) during the transition to OSS. In the following sections we expose the methodology adopted, the tools employed, and the main results obtained from the qualitative and quantitative data collected. The limitations and possible future additional work is listed at the end of the article. The overall sample of 22 users has been selected from three departments of the PA under exam. Some constraints had to be followed, for example, the fact that the head

Table 4. The selected sample and distribution among groups

Group	Women	Men	Total	Departments	Notes
Group 1	6	3	9	3	Only using MS Office
Group 2	9	4	13	3	Using MS Office and OpenOffice.org
Total	15	7	22	3	-

Figure 1. Experimental design adopted



of the different offices posed a limitation on the number of the available workers per office. Table 4 represents the different groups, with two office directors per each group as part of the sample. The average age of participants to both groups was uniform between groups and has not influenced the results; users were selected from such departments in a random way with the limitations described previously. Regarding the protocol, the selection of the experimental groups has been done in a way to enable that the participants were, when possible, in some way in relation with each other, physically near and if possible coming from the same organisational units, to take advantage of possible network externalities that arise in terms of document exchange and reciprocal help (Shapiro & Varian, 1999).

One group experimented with the introduction of OpenOffice.org (our treatment X, in Figure 1), while the other group was used as a control group. The experimental design followed an experimental pretest-posttest control group design (Campbell & Stanley, 1990).

A questionnaire has been submitted to both groups before (O1) and after (O2) the introduction of OpenOffice.org to evaluate the effects of the experimentation on the attitude towards OSS. The activities of both groups have been constantly monitored by an automatic system for data collection (Sillitti, Janes, Succi, & Vernazza, 2003) that permitted the gathering of a series of objective process data (the series of observations O3,i).

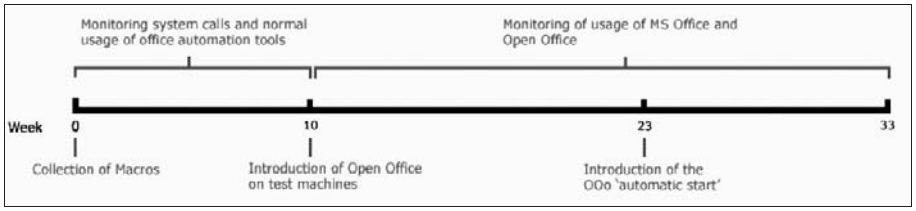
In some other cases where there has not been the possibility to have a control group and a proper randomisation of the sample, a "quasiexperimental" and a "one-shot" design have been employed (Campbell & Stanley, 1990). We are aware that in this way the results obtained are less extendible to the general case and more subject to exogenous effects.

TIME EVOLUTION OF THE EXPERIMENT

The experiment lasted for 32 weeks; during the first 10 only Microsoft Office was monitored and the different system dependencies were collected. OpenOffice.org was introduced in group 2 after week 10 and during the 23rd week of experimentation OpenOffice.org has been associated with Microsoft Office formats .doc and .xls. The results of the choice will be exposed in the subsequent sections. Figure 2 shows a graphical representation of the evolution of the experimentation.

In detail, the steps performed during the experimentation were the following:

Figure 2. Evolution of the experiment expressed in weeks



1. Selection of the participants to the experiment;
2. Submission of the questionnaires on the attitude towards OSS;
3. Motivational seminar on the reasons of the experimentation;
4. Identification of the OpenOffice.org experimental group and control group;
5. Analysis of the most used documents and the possible software dependencies;
6. Installation of OpenOffice.org and translation in the OpenOffice.org format of the most used documents and on a per request basis;
7. Installation of the monitoring and data collection system to define the situation before the transition;
8. Training, performed on a single day, trying to focus on the different approach proposed by the new software; users were instructed on how to perform the usual office automation tasks;
9. Start of the OpenOffice.org data collection;
10. Support given to users through an online forum and a hotline;
11. Periodic verification meetings with users of OpenOffice.org to identify possible problems;
12. Automatic start of OpenOffice.org with files with Microsoft Office extension starting from week 23;
13. Submission of the final questionnaires.

Two types of questionnaires have been submitted to users. The first one identical before and after the experimentation, to understand the attitude towards OSS and the effects of the experimentation on such attitude; the second one has been submitted only at the end, where all the final results of the project have been collected and more information for the replication of the experiment have been determined. The Goal Question Metrics (GQM) paradigm (Basili, 1995) was employed in every phase of the project, from the overall design to the creation of the questionnaires. The GQM is a methodology that was developed at the University of Maryland in the mid-1980s that relates the goals of an organisation to a set of questions. Questions are further associated to a set of metrics. In this way it is always possible to evaluate whether a goal has been reached and what are the informational needs of a certain goal are.

SOFTWARE EMPLOYED

The tools used during the experimentation were useful to assess the evolution of the experiment and in particular to gather quantitative and objective data about the migration process. In particular, two applications were employed for the ex-ante analysis and one was continuously employed during the transition to monitor the usage of the proposed solutions.

- PROM (PRO Metrics), a noninvasive monitoring tool was used to evaluate the usage of OpenOffice.org and Microsoft Office during all the transition process (Sillitti et al., 2003). Metrics of interest were the number of documents handled and time spent per single document. The software has been running during all weeks of the experimentation, permitting us to acquire objective data on the experimentation.
- DepA (Dependency Analyser) has been employed to evaluate at the beginning of the project the existing dependencies of Microsoft Office in terms of called and calling programs (Rossi & Succi, 2004). The program is a simple agent running on workstations to determine the calls from different applications, collecting in this way information on the different interrelations

between applications. The program has been running on client desktops for the first 10 weeks.

- FLEA (FiLe Extension Analyser) has been used to perform a scan of the data standards available on the users' drives and analyse the eventual presence of macros. The software permits us to collect information on the type of extension, date of creation, date of last access, size of the file, and for particular extensions also information about the macros contained. The scan was performed at the beginning of the experimentation.

All tools deployed are not noninvasive in order not to bias the results. From the final questionnaires emerged that users did not notice the presence of any external software during the experimentation.

DATA ANALYSIS

In this section we report the results of the data collection activities. In particular we can distinguish the data collection across a temporal boundary (*ex-ante, during, and ex-post*) and between *qualitative* and *quantitative* data.

The biggest effort during the project has been to monitor constantly the users during the experimentation. To gather objective data on the migration, we used the PROM software. Data collected included the time spent on documents and the number of documents opened using the selected office automation suite. A more fine-grained analysis on the function utilised has not been performed. During every phase of the project, the quantitative data collected has been backed with qualitative data coming from interviews and questionnaires. As a side effect, we noticed that the periodic meetings performed with users caused a small increase in the usage of the open source solution during the immediate subsequent days.

We will briefly review all the data collected, starting from the analysis of the existing situation, performed at the beginning of the experimentation.

EX-ANTE ANALYSIS

The aspects we analysed for an overview of the existing situation were concerned with the presence of interoperability issues in the users' environment and the presence of possible dependencies in the form of macros inside office automation documents. Macros are a series of commands inserted in the form of code inside documents, to perform a series of repetitive actions. They are generally very common in office automation documents, to permit the automation of repetitive tasks. As the usage in OpenOffice.org of macros written for Microsoft Office is not possible – at least at the time the experimentation was carried out – this is an interoperability issue. Macros need, in this way, to be completely rewritten.

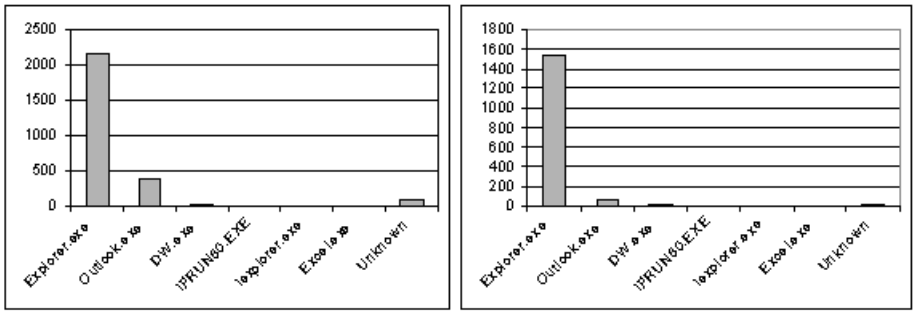
Table 5. Type of data collected during the experimentation

	Ex-ante	During	Ex-post
Qualitative	Interviews/questionnaires	Periodic meetings for feedback	Interviews/Questionnaires
Quantitative	Collection of data standards (FLEA), Collection of dependencies (DepA)	Monitoring of SW usage (PROM)	Number of OOo files created during the project.

Table 6. Number of documents and macros during the pretransition phase; LOC = Lines of Code, CNO = Could Not Open

Location	Microsoft Word				Microsoft Excel			
	Files	Macros	LOC	CNO	Files	Macros	LOC	CNO
Users' driver	19144	2	12	-	1367	8	1070	14
Network driver	4484	-	-	-	816	43	21482	331
Network drives (templates)	2182	-	-	-	9	3	10197	-

Figure 3. Number of incoming dependencies of Microsoft Word (left) and Microsoft Excel (right)



The number of templates in the preexisting format represents another interoperability issue. Our software for data collection granted us the possibility to evaluate the number of this type of documents, but not the complexity, another factor to take into account when there is the need to migrate a document. The collection of such data has to be crossed with interviews with the IT personnel to evaluate the real relevance of the macros discovered and the real necessity of the conversion of templates.

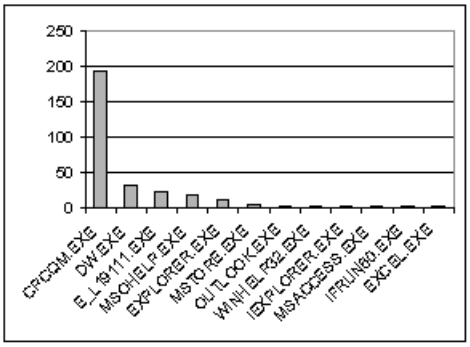
The first step for performing the initial analysis of the experimentation environment was the one related to the presence of macros inside documents and the distribution of the documents. Another important issue was to find the number of templates available. This analysis has been performed statically at the beginning of the project. In the evaluation of macros impact, Microsoft Word and Microsoft Excel documents of the participants to the project were considered. Two different locations were considered:

- Users' drives
- Network drives

In the last location, both normal documents and templates were considered. Table 6 reports a summary of the results. We found 25.810 Word documents and 2.192 Excel documents. Among these, only 2 Word documents (0.01%) and 49 Excel documents (2.24%) contained macros. Moreover, our tool has identified a high number of Excel documents protected by password (near 16% of the total). In this case, it is not possible to determine whether they contain macros.

The second step has been the identification of the software dependencies that existed in the office automation environment. A dependency is either a call to an external application (outgoing

Figure 4. Number of outgoing dependencies for Microsoft Word/Excel



call) or a call from an external application to the office automation suite (incoming call). As the dynamic evaluation of the calls, two different typologies have been considered:

- Applications that call Microsoft Word or Excel (Figure 3)

In this category we discovered that 80% of the times, Microsoft Word was called from explorer.exe which means a normal start from its icon or a file on the file manager. Furthermore, 12% of the times it was called from the e-mail client Outlook. Microsoft Excel was called 95% of the times by explorer.exe and 4% of the times from Outlook (for a total of 99% of the calls).

- Applications called from Microsoft Word or Excel (Figure 4)

74% of the global calls have been towards printer drivers. Almost 11% of the calls of Microsoft Word and Excel have been towards the program to report problems in Microsoft applications, and about 6% for the help guide.

Further interviews with IT managers confirmed the situation outlined by data collection tools. Globally, the system environment of the experimentation was less turbulent than we had thought initially. Templates, macros, and dependencies collected were not so critical to increase significantly the migration costs.

ONGOING EXPERIMENT ANALYSIS

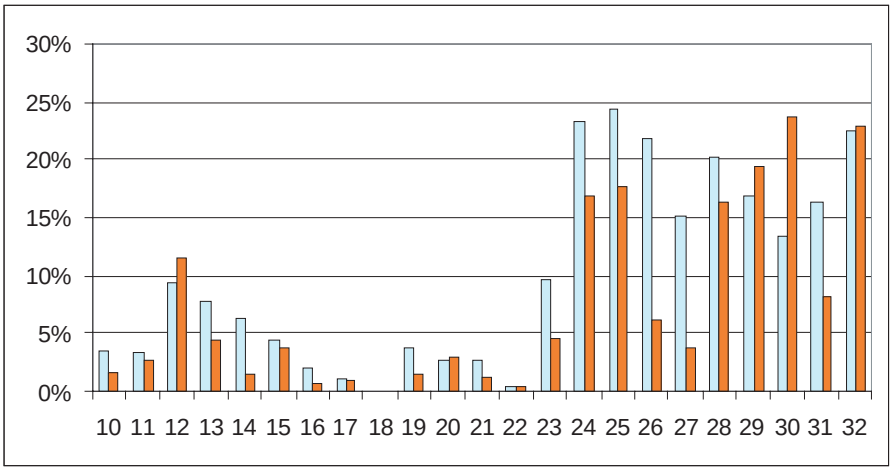
The experimentation has been monitored constantly by PROM software. Figure 5 shows two different measures of productivity, the percentage of documents opened by using OpenOffice.org and the percentage of time spent within the OpenOffice.org suite. As an example, a percentage of 5% means that users spent in that week 95% of the time using Microsoft Office.

From the data collected, we can notice, in particular, two effects:

- As expected, the level of adoption of the new software has been increased by the decision to associate the Microsoft Office file formats with OpenOffice.org. We expected to have complaints and reports of incompatibilities deriving from this decision. What happened was instead that users learned when it was convenient for compatibility reasons to adopt one solution or the other.
- Even after 32 weeks of experimentation, the time spent with OpenOffice.org was below 25% of the total time dedicated to the two suites for office automation. Also to note is the fact that the time and documents of Figure 5 are inclusive of the Microsoft Office documents opened with OpenOffice.org.

It is noteworthy that, during the experimentation, we did not benefit fully from the network effect that can be raised by the growing number of documents in one format and the subsequent exchange between users (Shapiro & Varian, 1999). In a broader migration such effects can also have an impact increasing the usage of the new platform proposed. In our experimentation users were in some way constrained in the adoption of

Figure 5. Increase in OpenOffice.org usage. On the X-axis there is the week of the project; OpenOffice.org has been inserted during week 10; and the automatic association has been activated during week 23. In blue the percentage of opened files using OpenOffice.org is represented and in orange the percentage of average time devoted to OpenOffice.org among users that effectively used OpenOffice.org.



the new format, as they could not exchange the documents with the users not participating in the experimentation.

We also posed two questions to evaluate the impact on productivity, according to the GQM methodology.

a) Did the usage of OpenOffice.org caused a reduction in the number of documents used per day? We studied the correlation between the number of documents used each day and number of documents opened with OpenOffice.org. A negative effect on the usage of OpenOffice.org had to produce a negative impact on the usage of the office automation documents and a significant negative correlation between these two variables; that is, the more documents are handled with OpenOffice.org, the less they are globally handled. The correlation in question has been of -0.08, therefore we exclude that the usage of OpenOffice.org has reduced the number of documents handled daily.

b) Did the usage of OpenOffice.org caused an increase in the time devoted to each document? We studied the correlation between the time spent managing all the documents and the OpenOffice.org ones. A negative effect on the usage of OpenOffice.org has to create a significant positive correlation; that is, it should be evident that the more time spent with OpenOffice.org, the more time is spent globally managing documents, as OpenOffice.org required more time to accomplish the same tasks. This correlation has been determined in -0.04, therefore it has to be excluded that the usage of OpenOffice.org has increased the global effort to handle documents. The comparison with the control group confirmed furthermore that the evolution of the usage of documents among the test group and the control group has been consistent, excluding the presence of exogenous factors.

EX-POST ANALYSIS

At the end of the experimentation, the evaluation of the attitude towards OSS and in general the

evaluation of the project has been performed. We submitted one questionnaire to users in order to evaluate their attitudes towards OSS and the knowledge that was acquired after the transition. We submitted also the same questionnaire to the control group to ensure that no exogenous effects biased the results. The questionnaire was designated to answer to the following two questions:

- What is the user perception of OSS at the end of the experiment?
- Has the user modified his/her perception of OSS at the end of the experiment?

All figures of this section represent the ex-ante situation on the left and the ex-post situation on the right. In this way it should be easier to evaluate the change of users' attitudes.

The *first question* that has been submitted was whether the Open Source concept has become more familiar after the experimentation. The result in this case is quite obvious; at the end the users had a clearer idea of the concept of OSS. It may be surprising the initial number of users claiming to know OSS, but this is due to the preproject meetings with IT managers explaining the reasons of the experimentation.

The *second question* enters the heart of the matter, questioning about the perception of OSS. In this case an interesting phenomenon has been discovered. At the beginning one group had no opinion on OSS (almost half of the interviewed). At the end of the experiment, almost all users have an opinion: those that were positive maintained the same opinion and the uncertain were divided into three groups almost of the same size, those with a positive opinion, those with a negative one, and those that had no opinion. Nevertheless, at the end of the experiment only a small part of the participants had a negative opinion on OSS.

The *third question* focuses on the importance of the diffusion of the software in use. In this

Figure 6. Question 1 – How familiar are you with the expression Open Source Software? Possible answers: (a) very well, (b) well, (c) neither nor, (d) not well, (e) not at all.

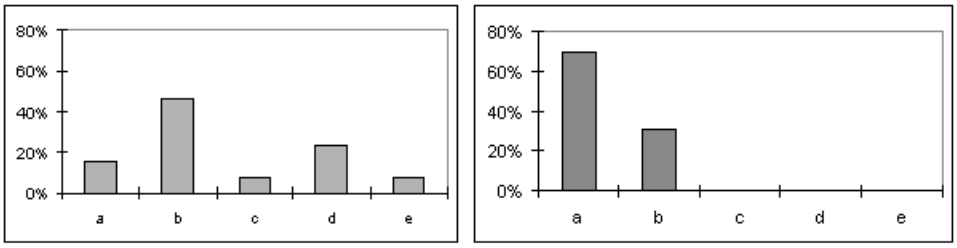


Figure 7. Question 2 – How do you perceive the expression Open Source Software? Possible answers: (a) as something negative, (b) as something positive, (c) neither positive nor negative.

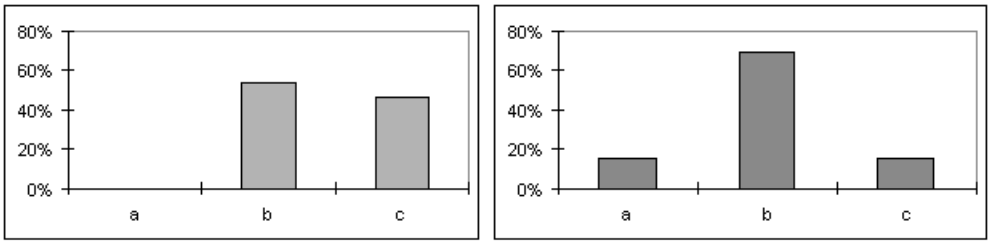


Figure 8. Question 3 - How important is it that the application you use is established and widely used? Possible answers: (a) very important, (b) important, (c) of moderate importance, (d) of little importance, (e) not important.

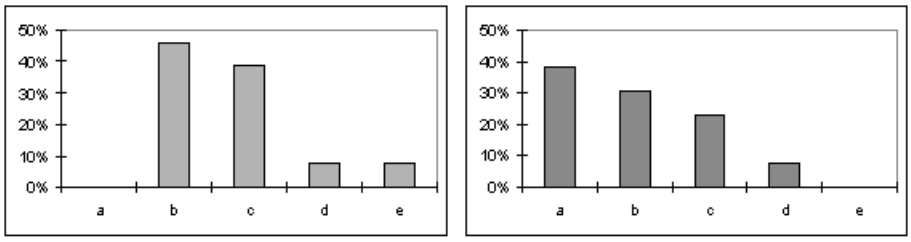
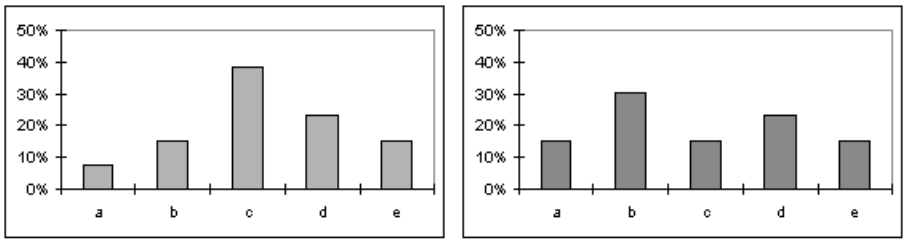


Figure 9. Question 4 – You are reluctant to give up the use of application software that you are using to in favour of an OSS alternative! Possible answers: (a) totally agree, (b) agree, (c) neither, (d) disagree, (e) totally disagree.



answer we have a strong movement towards a bigger knowledge; for example, the category "very important" has gone from 0 to almost 40%. The experimentation in the usage of OpenOffice.org has strongly increased the consciousness of the participants on the importance of the usage of well established software.

The *fourth question* poses the problem of the direct substitution and asks the user how much the user is reluctant to abandon the application in use in favour of an OS solution. In this case the user in favour maintained the same opinion, while reluctant people became more reluctant; also in this case users had a clearer idea of OSS after the experimentation. More than half of the interviewed users were still positive towards a possible transition.

The *fifth* and *sixth questions* present a slightly different nature. The aim is to find whether the experimentation changed the perception on the requirements of the software to be adopted and used. Question 5 focuses on the factors to consider when a new product is adopted and question 6 discusses the more important aspects of an efficient usage. In both cases the role of the training and support is evidenced as important, while other aspects as security, privacy, and availability of source code are considered less important. We can conclude that the effect of the OpenOffice.org introduction has increased the perception of the importance of the training. From one side, this can be obvious: the introduction of a new instrument requires always a training period, especially if it partially substitutes an old one. We must also point out that all the interviewed people had already performed their training with the office automation tools some time before, therefore such perception should already be present. It can be concluded that OSS has additionally stimulated the curiosity of participants, to the point to ask

Figure 10. Question 5 – How important do you find the following factors when you use a new IT-platform? Factors considered are: (a) support and training, (b) easiness of use, (c) interoperability, (d) source code available, (e) functionalities, (f) security, (g) privacy.

Position	*Before the experiment*	*After the experiment*
1	Easiness of use	Easiness of use
2	Functionalities	Support and training
3	Support and training, interoperability, security	Functionalities
4		Interoperability
5		Security
6	Privacy	Privacy
7	Source code available	Source code available

Figure 11. Question 6 – The biggest advantages you perceive with OSS are: Factors considered: (a) better support and training, (b) it is easier to use, (c) stability, (d) better functionalities, (e) better security.

Position	*Before the experiment*	*After the experiment*
1	Easiness of use	Easiness of use
2	Better functionalities, more stable	Better support and training
3		Better functionalities, more stable
4	Better support and training	
5	Better security	Better security

more questions about the tools used; such approach, if confirmed, can go in favour of people who claim that the adoption of OSS brings more "shared" knowledge.

The *last question* proposed before and after the experimentation is a sort of summary and deals with the motivations that the user would have to use OSS. It is interesting to note that near OSS supporters a new group emerged also absorbing neutral users towards a more negative opinion.

Some questions about the overall evolution of the migration have been submitted to users at the end of the project. Two questions are interesting for our evaluation of the migration, both related to the functionalities of OpenOffice.org and the possible full migration to the new solution proposed. We must point out that the experimentation has been performed with version 1.1.3 of OpenOffice.org; the latest release would, probably, obtain better results.

Question 8 faces the problem of the choice between the two proposed solutions in a general way. It asks whether the functionalities offered by the two suites are equivalent. Some users answered that Microsoft Office offers more functionalities than OpenOffice.org 1.1.3. In some way, the surprise may be that half of the users considered the set of functionalities offered equivalent.

Question 9 contextualises the problem. It tries to evaluate the impact of a possible substitution of Microsoft Office with OpenOffice.org. In this case almost all the participants in the test considered the migration as possible, even though the majority sustained later that this operation requires some effort and is not only a simple substitution.

A final evaluation of the experimentation has been performed on the number of files generated by the users adopting the new data standard supplied by OpenOffice.org. Table 7 contains a summary of all the different files created during

Figure 12. Question 7 – Which motivations do you have to use Open Source Software? Possible answers: (a) You believe it is right to support OSS initiatives, (b) You find that today's market dominance of a single software vendor is wrong, (c) other: specify, (d) I have no motivation to use Open Source Software.

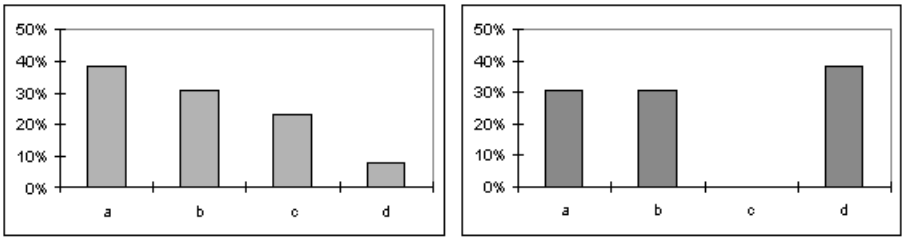


Figure 13. Question 8 – How do you evaluate the functionalities of OpenOffice.org with respect to the Microsoft Office ones? Possible answers: (a) widely superior, (b) superior, (c) equal, (d) inferior, (e) widely inferior.

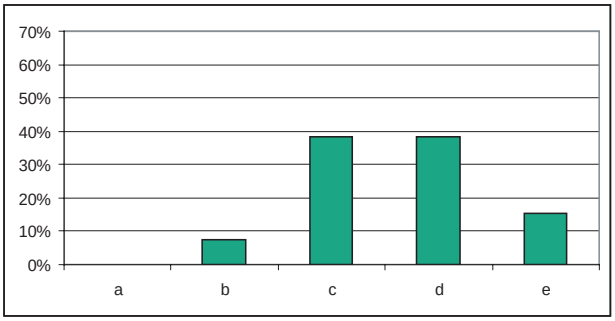


Figure 14. Question 9 – In this moment, if Microsoft Office is removed, are you still able to perform the same tasks? Possible answers: (a) yes, (b) yes but with some problems, (c) no.

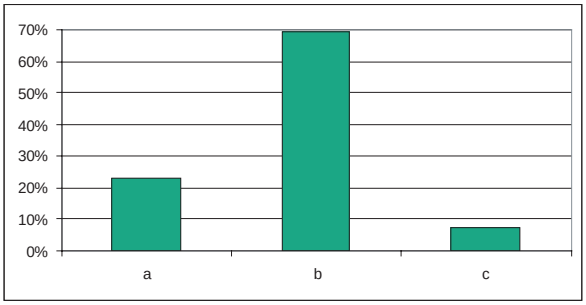


Table 7. Number of documents created during the experimentation

	Writer documents	**Calc documents**
Department 1	27	2
Department 2	223	34
Department 3	164	12
Total	414	48

the experimentation in the two different formats proposed by the OpenOffice.org suite for text processors and spreadsheets. Obviously this kind of static analysis represents only a small subset of the usage of the open solution, as users had also the freedom to open Microsoft Office proprietary formats using OpenOffice.org. For technical reasons FLEA could not be employed in this type of scan to give us more fine-grained data.

LIMITATIONS

This study represents the results of a single experience and such results cannot be systematically generalised, as the essential comparative aspect is missing. As already mentioned, the PA under exam has imposed some constraints on the selection of the sample. The office automation field, in particular, may not be fully comparable to other desktop environments where open source solutions are not as strong as OpenOffice.org. Furthermore, this study does not focus on a complete substitution of the old solution, rather on the evaluation of the coexistence of both solutions. A further step might be the evaluation of the effects deriving from a complete migration.

CONCLUSION

The migration to OSS described in this article has to be taken with care before generalising the results to other similar cases. In particular the migration has been restricted to the OpenOffice.org platform and to the PA field. The migration approach has been the more gradual as possible, maintaining the proprietary solution in parallel with the new one.

The results obtained from the experimentation have been encouraging for the introduction of OSS on the desktop-side. Data collected during the experimentation shows that the usage of the new platform increased during the whole period, reaching at the end 25% of the total office automation tasks. Proprietary software remained the preferred solution for users. The impact on productivity has been minimal also due to the similarities of the software considered. Users acquired a better understanding of OSS after the experimentation and tended to have, in general, a positive vision of the whole movement. Software used has been considered adequate for the transition, although a sort of lack of functionalities emerged from the opinions of the users. More recent releases of OpenOffice.org should solve these problems.

ACKNOWLEDGMENT

We acknowledge Dr. Hellmuth Ladurner for his precious help and support. Acknowledgments also go to all the users involved, participants in the experiment, technical personnel, and supervisors; without their constant effort and their availability, this study would not have been possible.

REFERENCES

Basili, V. (1995). Applying the goal/question/metric paradigm. Experience factory. In *Software quality assurance and measurement: A worldwide perspective* (pp. 21-44). International Thomson Publishing Company.

Campbell, D.T., & Stanley, T.D. (1990). *Experimental and quasi-experimental design*. Houghton Mifflin Company.

CERT. (2001). Advisory CA-2001-19: Code Red Worm. Retrieved June 15, 2006, from http://www.cert.org/advisories/CA-2001-19.html

Everitt, K., & Lederer, S. (2001). *A usability comparison of Sun StarOffice Writer 5.2 vs. Microsoft Word 2000*. Retrieved June 15, 2006, from http://www.sims.berkeley.edu/courses/is271/f01/projects/WordStar/

Feller, J., & Fitzgerald, B. (2001). *Understanding Open Source Software development.* Addison-Wesley.

Fenton, N.E., & Pfleeger, S.L. (1997). *Software metrics: A rigorous and practical approach* (2nd ed.). PWS Publishing Company.

Gartner Inc. (2003). *Distributed computing chart of accounts.* Retrieved June 15, 2006, from http://www.gartner.com/4_decision_tools/modeling_tools/costcat.pdf

Landeshauptstadt München. (2003). *Clientstudie der Landeshauptstadt München*. Retrieved June 15, 2006, from http://www.muenchen.de/aktuell/clientstudie_kurz.pdf

Netcraft Survey. (2005). Retrieved June 15, 2006, from http://news.netcraft.com/archives/web_server_survey.html

Newsforge. (2002). *Largo loves Linux more than ever.* Retrieved June 15, 2006, from http://www.newsforge.com/print.pl?sid=02/12/04/2346215

Nichols, D.M., & Twidale, M.B. (2003, January). The usability of Open Source software. *First Monday, 8*(1). Retrieved June 15, 2006 from http://www.firstmonday.org/issues/issue8_1/nichols/

Robert Frances Group. (2002). *Total cost of ownership for Linux Web servers in the enterprise*. Retrieved June 15, 2006, from http://www.rfgonline.com/subsforum/LinuxTCO.pdf

Rossi, B., & Succi, G. (2004). *Analysis of dependencies among personal productivity tools: A case study.* Undergraduate Thesis, Free University of Bolzano-Bozen.

Shapiro, C., & Varian, H.R. (1999). *Information rules: A strategic guide to the network economy.* Harvard Business School Press.

Sillitti, A., Janes, A., Succi, G., & Vernazza, T. (2003, September 1-6). Collecting, integrating and analyzing software metrics and personal software process data. In *Proceedings of EUROMICRO 2003*, Belek-Antalya.

Stadt Nürnberg. (2004). *Strategische Ausrichtung im Hinblick auf Systemunabhängigkeit und Open Source software*. Retrieved June 15, 2006, from http://online-service.nuernberg.de/eris/agendaItem.do?id=49681

Stadt Wien. (2004). *Open Source software am Arbeitsplatz im Magistrat Wien.* Retrieved June 15, 2006, from http://www.wien.gv.at/ma14/pdf/oss-studie-deutsch-langfassung.pdf

The Yankee Group. (2005). 2005 *North American Linux TCO survey.* Retrieved June 15, 2006, from http://www.yankeegroup.com

ZDNet. (2005). *Extremadura Linux Migration case study.* Retrieved June 15, 2006, from http://insight.zdnet.co.uk/software/linuxunix/0,39020472,39197928,00.htm

ENDNOTES

1. Apache Software Foundation, http://www.apache.org
2. The Mozilla Firefox project, http://www.mozilla.com/
3. gnuLinex, http://www.linex.org/
4. OpenOffice.org, http://www.openoffice.org
5. Microsoft Office, http://www.microsoft.com/office/editions/prodinfo/default.mspx

This work was previously published in International Journal of Information Technology and Web Engineering, Vol. 1, Issue 3, edited by E. Damiani and G. Succi, pp. 64-80, copyright 2006 by Idea Group Publishing (an imprint of IGI Global).

Chapter 4.14
An Empirical Investigation into the Adoption of Open Source Software in Hospitals

Gilberto Munoz-Cornejo
University of Maryland Baltimore County, USA

Carolyn B. Seaman
University of Maryland Baltimore County, USA

A. Güneş Koru
University of Maryland Baltimore County, USA

ABSTRACT

Open source software (OSS) has gained considerable attention recently in healthcare. Yet, how and why OSS is being adopted within hospitals in particular remains a poorly understood issue. This research attempts to further this understanding. A mixed-method research approach was used to explore the extent of OSS adoption in hospitals as well as the factors facilitating and inhibiting adoption. The findings suggest a very limited adoption of OSS in hospitals. Hospitals tend to adopt general-purpose instead of domain-specific OSS. We found that software vendors are the critical factor facilitating the adoption of OSS in hospitals. Conversely, lack of in-house development as well as a perceived lack of security, quality, and accountability of OSS products were factors inhibiting adoption. An empirical model is presented to illustrate the factors facilitating and inhibiting the adoption of OSS in hospitals.

INTRODUCTION

The open source software (OSS) phenomenon has become an important area of interest in information systems research due in part to the large and fast-growing number of OSS users and software products in a large variety of domains. OSS is already being adopted and used as a software platform in a number of fields other than

healthcare (Dedrick & West, 2003; 2004; Norris, 2004; Waring & Maddocks, 2005), and it has the potential to be equally promising for the hospital industry (Fitzgerald & Kenny, 2004). Studying OSS adoption in any domain can help reveal patterns and phenomena that are applicable to adoption in general, in addition to revealing insights into the domain being studied. In particular, the adoption and use of OSS in a hospital context remains a poorly understood phenomenon; only a handful of researchers have addressed the factors inhibiting or facilitating such adoption. Such an understanding is important in helping hospitals make better decisions about whether and how adoption of OSS could benefit them.

The first step in developing a better understanding is to explore the current state of OSS adoption, and the factors inhibiting and influencing it in hospitals. Such an exploration is the goal of this study. Once this current state is well described, it will be possible to seek answers to higher-level questions about the pros and cons, the costs and benefits, the advantages and disadvantages of OSS adoption in this domain, which is the second goal.

Therefore, the present study is of considerable interest for both practitioners and researchers. It will provide hospitals and healthcare organizations that are considering the adoption of OSS technologies with an understanding of how technological, environmental and organizational factors affect the adoption process. This way hospital IT practitioners, or others attempting to introduce OSS technology into hospitals, can prepare against the expected barriers and can utilize the facilitators for successful adoption. This research also provides scholars with an empirical model for better understanding facilitating and inhibiting factors, as well as providing the foundations for further research that may validate and expand on the empirical model in other healthcare organizations and other domains.

The main objective of this investigation was to explore and analyze the extent of OSS adoption in hospitals, along with the factors influencing or inhibiting this adoption process. Hospital IT managers were chosen to represent the hospitals' perspective on this topic. The following three questions guided this investigation:

1. What are the types and names of OSS products that hospitals choose to adopt?
2. What is the extent of OSS adoption for these products in hospitals?
3. What are the factors facilitating and inhibiting the adoption of OSS in hospitals?

To research these questions, a survey and interviews were used to acquire both breadth and depth of understanding. The purpose of the survey was to answer the first two questions—to explore and characterize the types of OSS products adopted in hospitals and to discover the extent to which these products have been adopted. The interviews were used to answer question three to attain a deeper understanding of the factors that are facilitating and inhibiting the adoption of OSS in hospitals.

In the following sections of this article, we first present the related work in this area. Then, we introduce the methodology for our survey and interview studies. After that, we present our data analysis and results. Then, we introduce our empirical model of the adoption of OSS in hospitals. Finally, we present our conclusions and the implications of our work.

LITERATURE REVIEW

Open Source Software Adoption in Healthcare

Over the past few years, a small number of researchers have focused on the study of the potential advantages and risks of adopting and implementing OSS in the healthcare domain. Prior research encouraged the adoption and use of OSS in healthcare organizations because of OSS's

potential to both enhance healthcare delivery and lower software acquisition costs (Carnall, 2000; Kantor, Wilson, & Midgley, 2003; McDonald et al., 2003; Valdes, Kibbe, Tolleson, Kunik, & Petersen, 2004).

OSS could potentially be more reliable and secure than proprietary software because its source code can be inspected and reviewed (Carnall, 2000). Past research introduced and extended the idea of OSS as a software development model that could definitively improve clinical and research software in the field of medical informatics (Yackel, 2001). A paper by Kantor, Wilson, and Midgley (2003) also presents the potential benefits that OSS could provide in the area of primary care. Kantor et al., also proposed that the adoption of OSS would reduce the excessive costs, the frequent turnover of vendors, and the lack of common data standards that are afflicting electronic medical records (EMR) systems in primary care.

More recently, McDonald, Schadow, Barnes, et al. (2003) also investigated the potential role that the OSS model of software development may have in the medical informatics area. They also described a number of OSS products that have been used in the medical informatics domain over the years, including: OpenEMed, a patient record system; OSCAR, a family practice office management and medical record system; as well as the internationally well-known VistA system, a computer-base patient records system (CBPR) developed in MUMPS (Massachusetts General hospital Utility Multi-Programming System) by the U.S. Department of Veteran's Affairs (Brown, Lincoln, Groen, & Kolodner, 2003; Longman, 2007). A more recent study by Valdes et al. (2004) also pointed out that OSS could be an effective solution for the problems that distress the healthcare industry such as high costs, business failures and barriers of standardization (Valdes et al., 2004). Other papers by Erickson, Langer, and Nagy (2005), Scarsbrook (2007) and Nagy (2007) supported the growth and adoption of OSS in radiology because OSS may significantly lower the entry cost for standards-compliant practices in the healthcare industry. They also proposed that OSS might allow rapid scientific advancement due to the sharing of information and software (Erickson et al., 2005; Scarsbrook, 2007). Other authors such as DeLano (2005) presented some reasons for the potential success of OSS predicting that the pharmaceutical research and development process may benefit from the OSS development model.

Open Source Software Adoption in Hospitals

A case study of OSS adoption was conducted at the Beaumont hospital in Ireland, where the IT department, under limited financial resources, made the decision to adopt OSS. Several OSS products were adopted and implemented successfully. The authors reported that there were important initial start-up and future operational costs when OSS products were preferred in the hospital (Fitzgerald & Kenny, 2004).

Another study by Glynn, Fitzgerald and Exton (2005) investigated the commercial adoption of OSS using an innovation adoption theory framework based on Tornatzky and Fleischer's (1990) model. They derived a framework that was then used to investigate the adoption process of OSS in the case of the Beaumont hospital (Fitzgerald & Kenny, 2004).

The OSS products and processes were also seen as promising in terms of enabling rapid evolution and proliferation of applications in the medical domain through their use of open standards and higher degrees of interoperability (Raghupathi & Gao, 2007). The authors argued that the development processes in the Eclipse project (http://eclipse.org) could improve scalability, prevent vendor lock-ins, and reduce costs in the medical information systems including electronic health record and clinical decision support systems.

Table 1. Main facilitators/inhibitors of OSS adoption

	Author(s)	**Major Factor Findings**	
		Facilitators	**Inhibitors**
Adoption of Open Source Software in Hospitals	**Fitzgerald and Kenny (2004)**	• Limited financial resources • Top management support • Software functionality • User's past experience	• Lack of support from vendors • Perception that OSS would threaten local proprietary software companies • Fear by users to become de-skilled
	Gynn, Fitzgerald and Exton (2005)	• Perception that the benefits of OSS outweigh its disadvantages • OSS-literate IT personnel • Top management support • Personal support for OSS ideology • Network externalities • The OSS champion example	• Perception of work under-valued if using OSS products • Having to change operating model to OSS • Fear by users to be de-skilled • Lack of OSS champion example • Lack of tolerance to technical problems with OSS • Favorable arrangements with proprietary vendors
	Holck, Larsen and Pedersen (2005)	• Limited financial resources • Pressure to upgrade IT systems • Top management support • User's past experience • Government support	• Lack of reliable procurement models ○ Legal (licenses) ○ Technical (functionality, security, usability) ○ Corporate and business policy (vendor, customer support, and software alliances)
Adoption of Open Source Software in Healthcare	Tomas Yakel (2001)	• Access to real-world systems • Reduction of bugs in medical systems • Reduction of software ownership and development cost	• Lack of a mature OSS beyond prototype phase • High level of technical expertise required for OSS • Proprietary mindset of the medical community • Technology complexity of the medical domain • Lack of OSS-IT personnel support, specifically for medical software applications
	MacDonald et al. (2003)	• Public policy encouraging that all software developed by the government must be released under an OSS license • Information mechanisms to disseminate to the community about OSS developments and benefits	• Medical software currently in use is proprietary software • Leadership and top management in healthcare is risk adverse • Elimination of in-house personnel due to outsourcing • Technology complexity of the medical domain
	Hogarth and Turner (2005)	• Reduction of software ownership and development cost • Disappearance of vendor lock-in • OSS adherence to standards for compatibility and data interchange	• Lack of OSS-IT personnel support, specifically for medical software applications • Technology complexity in the medical domain • Success of mainstream applications might not translate to clinical software
	Kantor et al. (2003)	• OSS can reduce EMR ownership and development cost • Disappearance of the vendor lock-in • OSS adherence to standards for compatibility and data interchange	

There are some recent studies focusing only on the managerial and technical barriers to the adoption of OSS (Holck, Larsen, & Pedersen, 2005). Past research on OSS and healthcare also proposed that OSS would reduce the number of bugs and failures in medical systems, as well as reduce their overall cost (Yackel, 2001). A study by Hogarth and Turner (2005) focused on creating a catalogue of existing OSS clinical projects and on determining metrics for their viability. The authors mentioned that many of the factors that are required to make a "successful and vibrant" OSS community within the mainstream software applications systems (e.g., Linux, Apache, etc.) may not necessarily be applicable to the clinical software applications systems.

Another study by Kantor, Wilson and Midgley (2003) presented a set of potential advantages that the adoption of OSS may provide with regards to lowering the resistance of hospitals to the adoption of electronic medical records (EMR). These included: 1) the potential of OSS to reduce EMR ownership and software development costs, 2) the removal of vendor lock-in, and 3) the adherence of OSS to standards for the compatibility and data interchange among systems.

In another study by Valdes, Kibbe, Tolleson, et al. (2004) dealing with the barriers to the proliferation of electronic health records/electronic medical records (EHR/EMR), the authors concluded that OSS is a viable solution to the barriers of high cost, business failure and standardization that the healthcare industry is facing when adopting EHR/EMR. The authors mentioned that, for example, interconnectivity problems are more easily solved when using OSS, since no technical information can be hidden. They also added that OSS can help alleviate the high costs associated with the adoption and implementation of EHR/EMR (Valdes et al., 2004). Although this article presents a good case for the adoption of OSS solving the barriers that EHR/EMR is facing, the authors do not support their case with empirical data.

In summary, even though we have witnessed a widespread, significant OSS research and industry adoption of OSS, there are still few studies on OSS adoption and use, especially in the hospital industry. Only a handful of researchers have addressed the factors inhibiting or facilitating OSS adoption in hospitals (Carnall, 2000; Kantor et al., 2003; Valdes et al., 2004; Glynn, Fitzgerald & Exton, 2005). Each of the aforementioned studies in this section found that top management support, limited financial resources, past experiences using OSS-like systems, and the flexibility to modify, combine, and tailor OSS are the most important facilitating factors for the adoption of OSS within a hospital scenario. The factors inhibiting adoption range from the fear of IT personnel becoming de-skilled by not using mainstream commercial applications, the lack of OSS-literate IT personnel, the lack of other successful OSS examples in the industry, to the lack of reliable procurement models for the adoption of OSS. Finally, many of the papers and studies reported are cases from European countries, with healthcare systems that are very different from that in the U.S. Table 1 presents only a summary of the facilitators and inhibitors shown to influence the adoption of OSS as found in the literature.

METHODOLOGY

A mixed methods design was used in this research to explore the extent of OSS adoption in hospitals as well as to investigate the influencing and inhibiting factors. The exploratory approach of this study is warranted by the fact that, as of yet, the adoption and use of OSS in U.S. hospitals has not been accompanied by any theoretical grounding or by empirical analysis that explains how or why OSS products are being adopted and used. That is, thus far, there are few existing conceptual frameworks to guide a research effort in this area. Similarly, there are no theoretical guidelines that have been

empirically evaluated to support a rigorous understanding of the complex factors that inhibit the adoption and successful implementation of OSS technologies in hospitals. For these reasons, a mixed methods approach using a grounded theory perspective was selected over a confirmatory or causal research design approach. Grounded theory is a systematic, qualitative research procedure used to develop an inductively grounded theory that explains a process, an action, or interaction about a phenomenon (Glaser & Strauss, 1967; Glaser, 1978; 1999; Creswell, 1994; 2005; Strauss & Corbin, 1998; Charmaz, 2006).

STUDY DESIGN

The data collection methods used in this research are a survey and interviews, allowing both breadth and depth of information concerning the adoption of OSS in hospitals. We focused on Baltimore, Washington and Northern Virginia (BWNV) area hospitals instead of a nationwide area. This allowed us to spend more time cultivating each contact from the target population through initial phone calls, and to obtain richer data in the form of personal face-to-face and telephone exchanges.

First, a survey was used to gather data from a wide variety of hospitals dispersed across a geographic area. This was done in order to explore and characterize the extent and the types of OSS products adopted by hospitals. Following the survey, semi-structured interviews were conducted in-person and by telephone with IT managers in order to attain deeper understanding of the factors that facilitate or inhibit OSS adoption in hospitals. Interviews are the quintessential qualitative method for data collection and one of the most widely used techniques for acquiring qualitative data in order to collect impressions and opinions about the particular research issue (Tashakkori & Teddlie, 1998; Patton, 2002).

The target population for the study consists of hospital executives, directors and managers that are involved in IT within BWNV area hospitals. Although we selected the BWNV area largely because of our own location, it is an appropriate choice because it is one of the most diverse areas in the U.S. socioeconomically, politically, and culturally. The survey sample was selected from the Healthcare Information and Management Systems Society (HIMSS) from their electronic mailing list database of chief information officers (CIO), chief technology officers (CTO), vice presidents (VP) (of information technology (IT), information systems (IS) and management information systems (MIS)), and directors and managers of other IT departments within hospitals. HIMSS was selected because it is a leading non-profit organization dedicated to improving healthcare through the application of information technology (HIMSS, 2006). This research takes a key informant approach that allowed the responses of the IT managers to represent those of the hospital being surveyed. The use of managers as key informants has been successfully applied in many IT studies that involve organizations (Huff & Munro, 1985; Gatignon & Robertson, 1989; Chau & Tam, 1997; Eyler et al., 1999; Goode, 2005).

SURVEY ADMINISTRATION

Prior to sending the survey invitation e-mail out, an attempt was made to contact each of the IT managers in the target population by telephone in an effort to encourage participation and receive a verbal commitment from them to complete the survey. After the initial telephone contact, an e-mail invitation letter was sent to the potential respondents. The survey link was appended to the bottom of the e-mail cover letter and upon clicking the survey link, the participant was directed to the online survey (Appendix A).

Descriptive statistics, such as frequency distributions, percentages, standard deviations, confidence intervals, Chi-square and Fisher's tests were computed in order to analyze the survey

results. Moreover, to ensure better reporting and complete description of our Web-based survey results, we applied a checklist of recommendations from the Checklist for Reporting Results of Internet e-Surveys (CHERRIES) (Eysenbach, 2004) (Appendix B).

INTERVIEWS

The interview population consisted of the subset of survey respondents who responded positively to a survey item that specifically asked if they were willing to share their thoughts and experiences in an interview. A total of 11 survey respondents initially agreed to be interviewed. All such respondents were sent an e-mail letter introducing the objectives of the interview and asking to schedule a meeting. By the end of this process, only five IT hospital managers ultimately agreed to be interviewed. The other six managers, for reasons unknown, chose not to respond to the many invitations by e-mail and telephone to participate and were unreachable to be interviewed. Each interview lasted 30-60 minutes and was conducted between January and May 2007. The interviews focused on the organizational, technological, and environmental factors that facilitated or inhibited the adoption of OSS at their hospitals (Appendix C). Before conducting each interview, the participant was briefed on the nature and purpose of the study. All the participants were asked for their authorization to be recorded during the interview and were asked to sign an informed consent.

The interviews were coded and analyzed employing grounded theory consistent with the systematic procedures recommended by Strauss and Corbin (1998), namely open coding, axial coding and selective coding. Coding is the process that dissects, differentiates, combines, and discovers concepts and relevant features from the data (Seaman, 1999). We developed concepts and categories emerging from the data using the line-by-line analysis as described by Straus and Corbin (1998) and Glaser and Straus (1967). The concepts and categories were generated by our analysis of the data and validated applying the constant comparative method. Each interview was treated as an individual case. NVivo® was used to assist the qualitative analysis process, to manage data, to store the interview transcripts, and to help in coding text (Bazeley & Richards, 2000).

RESULTS

Survey Results

This research finds that 23% (n=7) of the hospitals within the survey sample have adopted OSS. Conversely, 76% (n=23) of the hospitals indicated that they have not adopted any type of OSS. All of the hospital adopters of OSS reported having general-purpose products. Among them, only 57% (n=4) reported having adopted domain-specific products. Table 2 presents descriptive statistics profiling the hospitals in our survey sample.

Key findings from this research indicate that hospitals are adopters of both general-purpose and domain-specific products, but they have adopted general-purpose products to a greater extent than domain-specific products. General-purpose OSS adoption in hospitals clusters mainly in databases, desktop software, programming languages, and operating systems, as well as Web development tools and server products. Well-known OSS products such as MySQL, Linux, Apache, Firefox, PHP and Perl were the leading software products that hospitals selected to adopt. The scale used in the survey to indicate extent of adoption was adapted following Fichman and Kemerer (1997) and it ranges from unawareness (no knowledge of OSS), to awareness, interest (actively learning), evaluation/trial (acquisition and initiation of an evaluation or trial version), commitment (use for one or more deployment projects), limited deployment (regular, but still limited, deployment and use), and general deployment stages (a

stable and regular part of the IT infrastructure). The survey results show that the vast majority of general-purpose products are positioned from the evaluation/trial stages to the limited deployment stages. Well-known OSS products, for example, MySQL, Linux, Apache, and Perl, are in the limited deployment stages, whereas OSS desktop software applications, such as Firefox and Mozilla, are in the evaluation/trial stages. The extent of adoption of domain-specific products is lower than that of general-purpose products. The predominant adoption stages for all the domain-specific OSS products are awareness to interest. Domain-specific adoption occurs mainly in the telemedicine, electronic medical records, radiology, laboratory and pharmacy information systems products.

Furthermore, the results of the survey provide information about relevant contextual and structural characteristics of the hospitals that tend to adopt OSS. These characteristics may have a determinant effect on the adoption of OSS. First, the majority of the adopting hospitals are very large hospitals, with 500 beds or more. Second, these hospitals tend to have high annual revenue, more than $500 million. Third, hospital adopters

Table 2. Descriptive statistics of surveyed hospitals

	Frequency (n =30)	Percent %
Hospital type		
Healthcare system hospital	12	40.0
Hospital as a part of a multi-system network	11	36.7
Stand-alone hospital	5	16.7
Ambulatory care facility	1	3.3
Other	1	3.3
Number of beds in the hospital		
<50 beds	1	3.3
101-200 Beds	2	6.7
201-300 Beds	6	20.0
301-400 Beds	4	13.3
401-500 Beds	2	6.7
>501	11	36.7
Not classified by beds	4	13.3
Hospital's annual gross revenue		
< $5M	1	3.3
$5M-$25M	3	10.0
$26M-$50M	3	10.0
$51M-$200M	2	6.7
$201M- $350M	8	26.7
$351M-$500M	2	6.7
> $501M	11	36.7

continued on following page

Table 2. Descriptive statistics of surveyed hospitals

Annual IT operating budget		
<2%	5	16.7
2.1-3.0%	15	50.0
3.1-4.0%	1	3.3
4.1-5.0%	5	16.7
5.1-6.0%	1	3.3
>8%	3	10.0
Type of IT personnel		
In-house	27	90.0
Outsourced	3	10.0
Number of in-house IT staff employed full time		
≤10	5	16.7
10-30	10	33.3
31-60	7	23.3
≥91	8	26.7
Years of experience of in-house IT staff		
≤2 years	1	3.3
2-5 years	6	20.0
5-10 years	15	50.0
≥10 years	8	26.7

of OSS have a propensity to have a large number of IT support staff. Finally, hospitals that have adopted OSS also tend to have IT budgets that are less than 3% of the hospital's total budget.

Interview Results

This study also identifies, through the interview data, key categories that facilitate and inhibit the adoption of OSS in the hospitals within the sample. The interview data reveal that hospital software vendors are the most critical factor influencing the adoption of OSS in hospitals. Further, hospitals rely heavily on software vendors for all of their IT solutions. The results also show that hospital software vendors enlarge their product lines and the services they provide to hospitals to include general-purpose and domain-specific OSS products. In addition, IT managers have a positive satisfaction level, in general, with the software vendor services and products, and, overall, have a good relationship with them. Table 3 presents a concise summary of the results of the interviews.

The majority of the hospital IT managers reported that lack of in-house development, and a perceived lack of security, quality, and accountability of OSS products were the most significant factors that inhibit the adoption of OSS in hospitals. IT managers also identified the lack of medical informaticians, patient-privacy protection and privacy legislation as major inhibitors to adopt OSS, particularly domain-specific products.

Based upon our findings (from both the survey and interviews), the following section presents an empirical model describing the factors facilitating and inhibiting the adoption of OSS in hospitals and the relationships between them.

ADOPTION OF OSS IN HOSPITALS: AN EMPIRICAL MODEL

We have used Strauss and Corbin's (1998) paradigm to develop an empirical model describing the adoption of OSS in hospitals, based on our data. This empirical model helps us to develop and propose connections between the factors that emerged from our findings. Figure 1 presents the empirical model that lays out the analysis of the factors that emerged from our results and the relationships between them.

The empirical model identifies temporal and inferential, rather than causal, relationships between the factors relevant to the adoption of OSS in hospitals. For example, the mix of causal conditions in a particular hospital at a particular point in time (as defined by the level of in-house development, the number of IT personnel, etc.) sets the stage and shapes what happens when an event occurs related to the core category (e.g., when a software vendor offers an open source solution to the hospital). This core category then directly influences the strategic actions (i.e., adoption or non-adoption of OSS) that lead to the consequences. The contextual factors and intervening conditions moderate and mediate the strategic actions that are employed to bring about certain consequences (Strauss & Corbin, 1998; Creswell, 2005). So, in terms of the symbology in Figure 1, an arrow from one construct to another cannot be interpreted to mean that the first construct in any way causes the second, but that the mix of factors and actions described by the first construct influence the mix of factors and actions described by the second construct in any particular instance. The constructs of the model are described in more detail below.

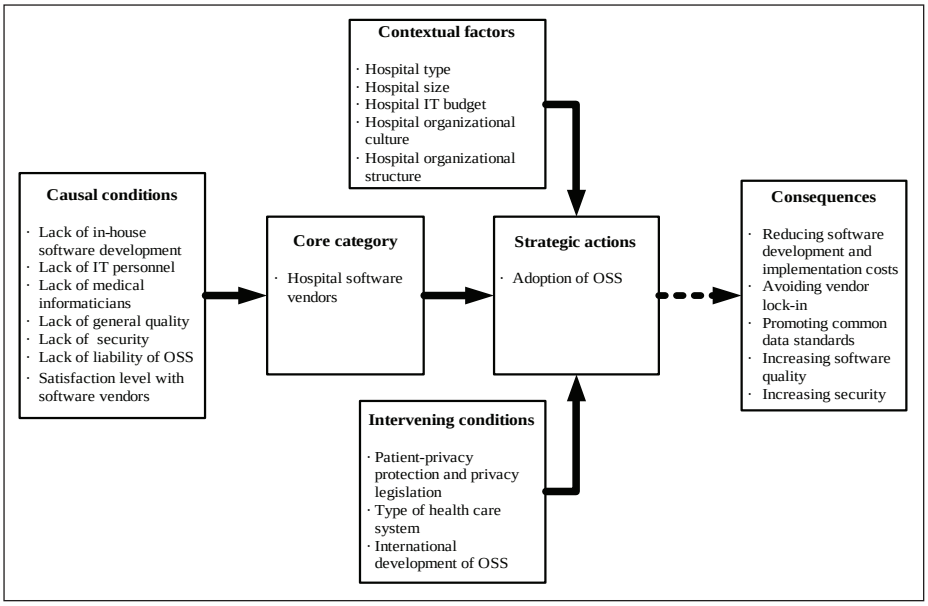
Causal Conditions

Causal conditions, as the term is being used in our empirical model, are factors that are identified as influencing the core category. There is evidence from our findings that all of these causal conditions have an influence on whether or not a hospital is open to an offer of OSS by a software vendor.

Table 3. Emerging code categories and subcategories of the adoption of OSS in hospitals

Core Categories	Subcategories
1. Hospital IT human resources	• In-house software development • IT personnel • Medical informaticians
2. Hospital regulatory landscape	• Patient-privacy protection and privacy legislation • Lack of liability/accountability provided by OSS
3. Hospital software vendors	• Software vendor providers of OSS • Satisfaction level with software vendors • New software business models
4. Hospital organizational factors	• Hospital organizational culture • Hospital organizational structure
5. Hospital technological factors	• Perceived lack of quality • Perceived lack of security
6. International development of OSS	• Labor cost and qualified programmers • Type of healthcare systems

Figure 1. Empirical model for the adoption of OSS in hospitals



The subject of technical personnel in hospitals came up often in our interview data (see "Interview Results" and Table 3). Hospital IT managers report that the lack of in-house development is the rule rather than the exception; hospitals do not develop their own software systems, and thus they depend on software vendors for all their IT operations and software needs. Managers also mentioned that much of their IT staff personnel are exclusively devoted to the on-site support of IT systems provided by vendors. The degree to which a hospital lacks in-house development activity, and IT personnel with technical development skills influences how dependent they are on their software vendors, and thus influences how they would react to the offerings of their vendors. Such a dependence would make a hospital more likely to accept a technology solution from a vendor that included OSS. A related causal condition is the lack of personnel who possess an amalgamation of medicine and information systems expertise and thus who would be able to develop and maintain software systems tailored to hospitals and healthcare organizations.

The perceived lack of general quality and perceived lack of security of OSS products are persistent themes that emerged from our data analysis (these are described under the core category "Hospital technological factors" in Table 3). As one manager commented "OSS is not going to have the same level of quality and not nearly the same level of documentation and rigor you can get from a corporate environment." Another IT manager opined that "the majority of the OSS are probably of inferior quality because they are just gifts that any research lab puts together and hands out from a couple graduate students." Managers also perceive OSS as a high-risk product when it concerns security. As one manager commented, "It is not the fact that the OSS won't be able to provide the functionality that we need in the hospital. The major concern is going to be how secure OSS is." Managers perceive OSS to be highly vulnerable to attacks from hackers or other parties, which may inhibit them from adopting OSS, even from a vendor. These quality and security factors will color a hospital's openness to a vendor's offer of an open source solution.

The lack of accountability of OSS providers is also a concern for the hospital IT managers we interviewed. Having a vendor that can be held liable or accountable if there is inadequate or insufficient quality or security of the software product strongly influences the decision to adopt products from software vendors. As one IT manager expressed: "the factor that caused us not to adopt OSS is the support and accountability that comes with writing a check to a commercial software vendor." The negative perceptions of quality, security and lack of liability reinforce the hospitals' dependence on software vendors.

Finally, our findings report that IT managers have a positive satisfaction level, in general, towards the products, support and services that software vendors provide in their hospitals, as noted in the interview results. This further reinforces the hospitals' dependence on software vendors.

In summary, these causal conditions all shape and impact the core category, that is, they influence what happens when and if a vendor offers a hospital a solution that includes OSS.

Core Category

The mix of causal conditions in a particular hospital setting sets the stage for the "core category," that is, the hospital software vendors. While our survey did not address the issue of software vendors, there was unanimous consensus amongst all the hospital IT managers interviewed that hospital software vendors play a pivotal role in the adoption process of OSS in hospitals, as discussed in the interview results. IT managers identify hospital software vendors who supply OSS products and services as the key facilitators for the adoption of both general-purpose and domain-specific OSS products. In terms of the empirical model presented in Figure 1, the actions of the software vendors is the trigger, or the gateway, that creates the situation where a hospital must decide to adopt or not adopt OSS. Such a decision does not even arise except through the actions of a software vendor, according to the findings of this study. As one manager commented, "hospitals are so dependent on vendors of hospital IT products that we are not in the position to kind of 'buck the rules' and go it alone for the adoption of OSS."

However, sometimes this decision is not even explicit. As one IT manager adopter of OSS expressed, "we don't have a conscious decision to adopt OSS because our hospital outsources a lot of our technical knowledge to vendors, so the adoption of OSS is coming throughout the vendor's decisions for the most part."

The hospitals' decision to adopt OSS from software vendors is linked to their belief that the OSS offered this way has "a professional level of quality control" that is greater than the OSS available from other sources, such as the Internet. As one IT manager who adopted vendor-supported OSS stated, "I am very happy using OSS because, for me, the best of two worlds is when vendors support an OSS solution. I am willing to pay for OSS, because I feel I have professional quality and control over the software."

Contextual Factors

Contextual factors are the "specific set of conditions (patterns of conditions) that intersect dimensionally at this time and place to create a set of circumstances or problems to which persons respond through actions/interactions" (Strauss & Corbin, 1998, p. 132). Our data, especially the survey data presented in "Survey Results", reveal that several contextual factors are expected to moderate the adoption of OSS in hospitals. The combined qualitative and quantitative results of this study provide evidence that the following contextual factors may facilitate or inhibit the adoption of OSS in hospitals: 1) hospital type, 2) hospital size, 3) hospital IT budget, 4) hospital organizational culture, and finally 5) hospital organizational structure. These factors are different from the causal conditions listed earlier, in that they are more general, static factors that

apply to the hospital as a whole and do not specifically form the hospital's attitude towards OSS, or towards the software vendor.

Depending on the hospital type (such as a stand-alone hospital versus a multi-hospital network, or a university hospital versus a private hospital, and so on), the importance of IT adoption within the hospital may differ. Different types of hospitals seem to have different requirements to adopt software. For example, a university hospital may allow experimentation with new software products while a private hospital in a multi-hospital network may not allow any type of experimentation. Such factors may have an effect on the adoption of OSS by hospitals. Hospital size is likely to be related to organizational characteristics such as slack in resources or a large professional workforce that can also have a positive effect on the adoption of OSS in hospitals. Hospital IT budget is another contextual factor that emerged in our study as a meaningful factor since hospitals with smaller relative IT budgets (with 3% or less of the total hospital budget) have a propensity to adopt OSS.

Other contextual factors within the hospital such as organizational culture and organizational structure can also have an effect on the adoption of OSS. As one manager commented, "the organizational design of the hospitals has a major influence on the adoption of software within the hospital, I don't want to use the word power structure, but it is almost the political landscape of the organization that influences the way we adopt any technology."

Our findings support the effect that all the aforementioned factors have on the strategic actions (i.e., adoption or non-adoption) as depicted in Figure 1 with regards to OSS adoption within hospitals.

Intervening Conditions

Intervening conditions are those conditions that "mitigate or otherwise impact causal conditions" (Strauss & Corbin, 1998, p. 131). The intervening conditions identified in this study included: 1) patient-privacy protection and privacy legislation, 2) type of healthcare system, and 3) international development of OSS. These intervening conditions are factors, external to the immediate hospital setting that may inhibit the adoption of OSS in hospitals. IT managers we interviewed (see Table 3) report that factors such as patient-privacy protection and privacy legislation may act as deterrents for the adoption of OSS in general, especially with regards to the domain-specific OSS products. For example, hospital IT managers were reluctant to adopt domain-specific OSS products because they perceived OSS as posing a threat to patients' privacy and confidentiality as well as to HIPAA compliance mandates. Consequently, we conclude that the aforementioned three intervening conditions also mediate the adoption of OSS in hospitals.

Strategic Actions

Strategic actions are "purposeful or deliberated acts that are taken to resolve a specific problem" (Strauss & Corbin, 1998, p. 133). The interaction outcome of the core category (hospital software vendors) with the contextual factors and the intervening conditions may result in a decision by hospitals to make full use of a technology—in this case OSS—as a plausible or implausible alternative to proprietary (closed source) or commercial software products.

Consequences

Consequences are the outcomes of the interaction of the core category with the contextual factors, intervening conditions and the strategic actions. The outcomes of this empirical model are closely aligned with the potential benefits of OSS claimed in the literature reviewed in literature section of this article. However, we can only speculate about the actual consequences, as that part of the model is beyond

the scope and objectives of this research. However, investigating the consequences of OSS adoption in hospitals is a vital area for future research.

IMPLICATIONS

Implications for the Literature

OSS has created a stir of interest in many disciplines ranging from computer science to sociology, and a growing body of literature has emerged to explain many aspects of OSS. However, no work has investigated the adoption of OSS in hospitals. The research presented here addresses this gap.

A number of respondents from the interviews noted that the lack of IT personnel and the lack of medical informaticians are inhibiting factors for adoption of OSS by hospitals. This is consistent with previous authors (Yackel 2001; MacDonald et al., 2003; Fitzgerald & Kenny, 2004; Hogarth & Turner, 2005; Waring & Maddocks, 2005) who have noted the importance of IT personnel with high levels of technical expertise required in order to deal with OSS applications and the technological complexity in the medical domain that needs personnel that understand both medicine and information systems.

In contrast to other studies claiming that the reduction of ownership and development cost is one of the main advantages of adopting OSS in healthcare (Yackel, 2001; Kantor et al., 2003; Fitzgerald & Kenny, 2004; Glynn et al., 2005; Hogarth & Turner, 2005; Holck, Larsen, & Pedersen, 2005), the findings from this research indicated that cost factors are not a core, important category for hospital IT managers when deciding to adopt OSS. The IT managers in our study were found to be more concerned about the quality, security and liability issues surrounding OSS than about the potential cost-benefit factors associated with the adoption and use of OSS. This finding also compares with a prior study by Goode (2005), which also noted that managers see software with high cost as an indicator of quality.

Prior research (Fitzgerald & Kenny, 2004; Glynn et al. 2005; Holck et al. 2005; Waring & Maddocks, 2005) has noted the importance of top management support for the successful adoption of technology within organizations. Our findings, by contrast, show that not only top management support is important to the adoption of OSS by hospitals, but clinical personnel within hospitals (e.g., physicians, nurses, etc.) also exert a significant influence on the decision to adopt not only OSS but any technology. Many IT managers recognized the political influence of these groups as a critical factor in how OSS would be used in the future, even before getting to the technology portion of the adoption of OSS by hospitals.

This study also shows that the hospital industry is a very conservative industry when it concerns adopting new technologies. Managers repeatedly indicated the "conservative aspects and risk adverse" behavior of the hospital industry to adopt not only OSS but also any new technology. This finding is consistent with MacDonald et Al. (2003) and Glynn (2005) who also pointed out hospitals' risk averse behavior when adopting IT.

Finally, our core finding about the central role of software vendors in the adoption decision in hospitals has some relationship to prior literature. Some existing studies have indicated that avoiding vendor lock-in is perceived to be an advantage of adopting open source (Carr, 2003; 2004; Fink, 2003; Kantor et al., 2003; Fitzgerald, 2004; Goldman & Gabriel, 2005; Goode, 2005). In contrast, in our study, the role of vendors emerged quite differently. The role of vendors as OSS adopters, who then transfer their adoption decisions on to their client hospitals, has not previously been described in the literature. This finding describes vendors as innovating the way they develop, distribute, support and maintain software systems within hospitals. Prior studies have not shown software vendors to be such key enablers of OSS in the hospital industry.

Implications for Future Research

This research is unique within the field of OSS and healthcare. That is, there is no study that has been published to date presenting an empirical model for the adoption of OSS in hospitals. This model, grounded in empirical data collected from surveys and interviews, identifies the factors and relationships facilitating and inhibiting the adoption of OSS in hospitals. This model provides the basis for future testing of the interactions among the key concepts proposed in this study. Furthermore, there are numerous significant issues for researchers, including ourselves. Our findings, while not highly generalizable due to the limitations of the study, provide sufficient grounding for future confirmatory studies.

In particular, a number of very interesting propositions or hypotheses are suggested by our empirical model, and by the survey and interview data. Future research aimed at validating these hypotheses would be a significant contribution to the field. Examples of such propositions include:

- **Proposition**: Adoption of OSS is more likely to be found in hospitals that have in-house technical staff with experience in software development, OSS, and/or with medical informatics.
- **Proposition**: Hospitals with an existing relationship with a software vendor who offers OSS solutions are more likely to adopt OSS. The likelihood increases with the degree of dependence on the vendor and the degree of satisfaction with the vendor.
- **Proposition:** The adoption of OSS in a hospital is more likely when there is a centralized IT strategy within the hospital.
- **Proposition:** The likelihood of a hospital's adoption of OSS is negatively correlated with the IT manager's perception of the general quality and security of OSS products.

To further validate propositions such as those above, as well as the whole empirical framework derived from this study, the following future research is planned:

1. Validation of the model by collecting data from a large sample of hospitals, either in the U.S. and/or internationally, would allow for further conclusions about the causal relationships and interactions suggested by our empirical model.
2. A case study in a hospital setting to analyze the consequences of the adoption and implementation of OSS.
3. Further empirical investigation into the relationship between hospital software vendors and adoption of OSS.

Implications for Practice

The present research provides a better understanding to hospital IT managers and practitioners about the extent of OSS adoption in hospitals in conjunction with the factors facilitating or inhibiting this adoption process. Hospitals and healthcare organizations that are considering the adoption and implementation of OSS technologies need to understand how technological, environmental and organizational factors affect the adoption process. This way IT hospital practitioners can prepare against the expected barriers and can utilize the facilitators for successful OSS technology adoption.

The first implication for practitioners is that, contrary to theoretical and anecdotal expectations about the cost-benefit advantages of OSS versus proprietary or commercial-based software, the findings from this research indicated that financial factors are not deemed to be a core concern for IT managers when deciding to adopt OSS. The IT managers in our study were found to be more concerned about the quality, security and liability

issues surrounding OSS. This implies that, when building a business case, or justification, for the adoption of OSS, the analysis must take into account issues related to quality, security, and accountability with at least as much prominence as cost-benefit issues.

The second implication for hospital IT practitioners would be to involve all the stakeholders within the hospital in the adoption decision-making; for this particular point, our finding indicated that physicians, nurses, and other clinical personnel are key stakeholders to address in the adoption process of not only OSS but any type of technology introduced to a hospital. Thus the receptivity to the idea and philosophy of OSS must be assessed with these stakeholders, and any ideas and concerns that might surface during the assessment must be documented and taken into account.

The third recommendation for hospitals that are considering OSS is that they can start adopting OSS with a small pilot project in order to test and experiment with the quality issues of interest, as well as the costs and benefits, of OSS to the hospital. In addition, it is very important to collect data and metrics from the pilot project and communicate the results to all the stakeholders, including vendors, within the hospital. It is important to mention that OSS is not "free," and never will be without a cost.

Another implication for practitioners who want to promote the use of OSS within the hospital and healthcare industry is for them to liaise with hospital software vendors and the OSS community. Coordinating with hospital IT vendors is important because, as our findings reported, any tendency towards adoption of OSS in hospitals is occurring because healthcare IT vendors are embracing, providing, and maintaining OSS products. Under this business model, hospital software vendors are not only offering the software to hospitals but also offering services for installation, customization, and maintenance of OSS applications, either domain-specific or general-purpose. Furthermore, there are good examples of software partnerships amongst IT businesses, open source communities, and researchers such as Eclipse and even Linux (Capek, 2005; Goldman & Gabriel, 2005; Zeller & Krinke, 2005) that can be replicated in the hospital and healthcare industry. Moreover, the hospital industry is probably the most influential and powerful industry operating today in the healthcare area. If this industry sees the benefits from OSS, then partnerships between IT businesses, OSS communities, and universities could result in research, development and promotion of OSS hospital products and policies that further the evolution of the OSS movement, as well as provide substantial benefits to the hospital industry. Therefore, such partnerships could be a potentially transforming development in promoting and adopting OSS in hospitals.

LIMITATIONS OF THE STUDY

Notwithstanding the important contributions of the current study, it has its own shortcomings. For example, our findings may not apply to the full spectrum of U.S. hospitals. This research is exploratory in nature, so that the design, data collection methods, and analysis were broad by design, and not intended for confirmation. This research also examined the adoption or non-adoption of OSS in a limited geographical area and over a particular time period, which makes any attempts to generalize the results across hospitals in the U.S. difficult without further empirical analysis and investigation.

Another limitation was the modest sample size of response in the survey (n=30) and interviews (n=5). Through the evolution of this study, it became clear that IT managers in the hospital industry in the BWNV area were less than enthusiastic about discussing and sharing information about open source adoption within their hospitals. Many attempts to influence a higher rate of response and interview participation were made, including initial contacts, follow-up contacts, reminders, and even

financial incentives. While the small sample size affects the ability to generalize results, it does not affect what was the intent of the study, to explore and identify relevant issues and factors for further study. However, it is important to mention that these are important limitations for any future similar study because of the unwillingness of the managers and executives to share their views on issues concerning IT adoption.

Finally, another limitation of this research is that the data appears not to represent all types of OSS products. While it was not the intent of the study, it is clear from the responses (in particular the types of OSS that survey respondents report adopting) that our respondents were referring primarily to large, enterprise-level OSS applications (e.g., database servers, Web servers, operating systems, etc.). This limits our ability to extend our findings to the entire spectrum of OSS products available to hospitals. It also limits our ability to compare the results of this study to prior research, which mostly addresses the adoption of smaller, stand-alone, download-and-install types of OSS applications.

CONCLUSION

This research identifies the factors that could lead to more effective adoption of OSS by hospitals. In addition, this research sheds light and broadens the understanding of OSS adoption within hospitals by offering to IT practitioners information on the extent of that adoption currently.

Finally, the insight gained from this research serves as a guide and foundation for future work to investigate more determinants of OSS adoption in hospitals and healthcare organizations. It is also the researcher's hope that this study will be the seminal stone to pave the way for future studies on OSS adoption and implementation in organizations, public and private, national and international.

ACKNOWLEDGMENT

The authors would like to thank Khaled El Emam, Medha Umarji, Roy Rada, Katherine Stewart Dongsong Zhang and Stephen Russell, as well as the anonymous IJHISI reviewers for their editorial insights. Additionally, we deeply appreciate the time, interest, and participation of hospital IT managers in the Baltimore-Washington area surveyed and interviewed in this study.

REFERENCES

Bazeley, P., & Richards, L. (2000). *The NVivo qualitative project book*. London; Thousand Oaks, CA: Sage Publications.

Brown, S., Lincoln, M., Groen, P., & Kolodner, R. (2003). VistA—U.S. Department of Veterans Affairs national-scale HIS. *International Journal of Medical Informatics, 69*(2-3), 135-156.

Capek, P., Frank, S., Gerdt, S., & Shields. D. (2005). A history of IBM's open source involvement and strategy. *IBM Systems Journal*, *44*(2), 249-248.

Carnall, D. (2000). Medical software's free future. *BMJ, 321*(7267), 976.

Carr, N. (2003). IT doesn't matter. *Harvard Business Review, 81*(5), 41-49.

Carr, N. (2004). *Does IT matter?: Information technology and the corrosion of competitive advantage*. Boston, MA: Harvard Business School Press.

Charmaz, K. (2006). *Constructing grounded theory*. London; Thousand Oaks, CA: Sage Publications.

Chau, P., & Tam, K. (1997). Factors affecting the adoption of open systems: An exploratory study. *MIS Quarterly: Management Information Systems, 21*(1), 1-20.

Creswell, J. (1994). *Research design: Qualitative & quantitative approaches*. Thousand Oaks, CA: Sage Publications.

Creswell, J. (2005). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (2nd ed.). Upper Saddle River, NJ: Merrill.

Dedrick, J., & West, J. (2003). *Why firms adopt open source platforms: A grounded theory of innovation and standards adoption.* Paper presented at the Proceedings of the Workshop on Standard Making: A Critical Research Frontier for Information Systems. Seattle, WA.

Dedrick, J., & West, J. (2004). *An exploratory study into open source platform adoption.* Paper presented at the Proceedings of the 37th Hawaii International Conference on System Sciences.

DeLano, W. (2005). The case for open source software in drug discovery. *Drug Discovery Today, 10*(3).

Erickson, B., Langer, S., & Nagy, P. (2005). The role of open source software in innovation and standardization in radiology. *Journal of the American College of Radiology, 2*(11), 927-931.

Eyler, A., Mayer, J., Rafii, R., Housemann, R., Brownson, R., & King, A. (1999). Key informant surveys as a tool to implement and evaluate physical activity interventions in the community. *Health Educ. Res., 14*(2), 289-298.

Eysenbach, G. (2004). Improving the quality of Web surveys: The checklist for reporting results of Internet e-surveys (CHERRIES). *J Med Internet Res, 6*(3), e34.

Fichman, R., & Kemerer, C. (1997). Object technology and reuse: Lessons from early adopters. *Computer, 30*(10), 47-59.

Fink, M. (2003). *The business and economics of Linux and open source*. Upper Saddle River, NJ: Prentice Hall PTR.

Fitzgerald. (2004). A critical look at open source. *Computer, 37*(7), 92-94.

Fitzgerald, & Kenny. (2004). Developing an information systems infrastructure with open source software. *Software, IEEE, 21*(1), 50-55.

Gatignon, H., & Robertson, T. (1989). Technology diffusion: An empirical test of competitive effects. *Journal of Marketing, 53*(1), 35-49.

Glaser, B. (1978). *Theoretical sensitivity: Advances in the methodology of grounded theory.* Mill Valley, CA: Sociology Press.

Glaser, B. (1999). The future of grounded theory. *Qualitative Health Research, 9*(6), 836-845.

Glaser, B., & Strauss, A. (1967). *The discovery of grounded theory; strategies for qualitative research.* Chicago, IL: Aldine Publishing Company.

Glynn, E., Fitzgerald, B., & Exton, C. (2005). *Commercial adoption of open source software: An empirical study.* Paper presented at the Proceedings of the 2005 International Symposium on Empirical Software Engineering (ISESE'05). Noosa Heads, QLD, Australia.

Goldman, R., & Gabriel, R. (2005). *Innovation happens elsewhere: Open source as business strategy.* Amsterdam; Boston, MA: Morgan Kaufmann.

Goode, S. (2005). Something for nothing: Management rejection of open source software in Australia's top firms. *Information and Management, 42*(5), 669-681.

HIMSS. (2006). *HIMSS bylaws.* Retrieved June 1, 2006, from http://www.himss.org/content/files/himss_bylaws.pdf.

Hogarth, M., & Turner, S. (2005). *A study of clinical related open source software projects.* Paper presented at the American Medical Informatics Association Conference (AMIA). Washington, D.C.

Holck, J., Larsen, M., & Pedersen, M. (2005). *Managerial and technical barriers to the adoption of open source software*. Berlin-Heidelberg: Springer.

Huff, S., & Munro, M. (1985). Information technology assessment and adoption: A field study. *MIS Quarterly*, (December), 327-338.

Kantor, G., Wilson, W., & Midgley, A. (2003). Open source software and the primary care EMR. *J Am Med Inform Assoc, 10*(6), 616.

Longman, P. (2007). *Best care anywhere: How America's most socialized healthcare system became its finest: What the VA miracle means for you and your country*. Sausalito, CA: PoliPointPress.

McDonald, C., Schadow, G., Barnes, M., Dexter, P., Overhage, J., Mamlin, B., et al. (2003). Open source software in medical informatics—why, how and what. *International Journal of Medical Informatics, 69*(2-3), 175-184.

Nagy, P. (2007). Open source in imaging informatics. *Journal of Digital Imaging, 20*(0), 1-10.

Norris, J. (2004). Mission-critical development with open source software: Lessons learned. *Software, IEEE, 21*(1), 42-49.

Patton, M. (2002). *Qualitative research and evaluation methods* (3rd ed.). Thousand Oaks, CA: Sage Publications.

Raghupathi, W., & Gao, W. (2007). An eclipse-based development approach to health information technology. *International Journal of Electronic Healthcare, 3*(4), 433-452.

Seaman, C. B. (1999). Qualiative methods in empircal studies of software engineering. *Software Engineering, IEEE Transactions on, 25*(4), 557-572.

Scarsbrook, A. (2007). Open source software for radiologists: A primer. *Clinical Radiology, 62*(2), 120-130.

Strauss, A., & Corbin, J. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (2nd ed.). Thousand Oaks, CA: Sage Publications.

Tashakkori, A., & Teddlie, C. (1998). *Mixed methodology: Combining qualitative and quantitative approaches*. Thousand Oaks, CA: Sage Publications.

Valdes, I., Kibbe, D., Tolleson, G., Kunik, M., & Petersen, L. (2004). Barriers to proliferation of electronic medical records. *Informatics in Primary Care, 12*(1), 3-9.

Van Latum, F., Van Solingen, R., Oivo, M., Hoisl, B., Rombach, D., & Ruhe, G. (1998). Adopting GQM-based measurement in an industrial environment. *Software, IEEE, 15*(1), 78-86.

Waring, T., & Maddocks, P. (2005). Open source software implementation in the UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management, 25*(5), 411-428.

Yackel, T. (2001). How the open source development model can improve medical software. *Medinfo, 10*, 68-72.

Zeller, A., & Krinke, J. (2005). *Essential open source toolset: Programming with Eclipse, JUnit, CVS, Bugzilla, Ant, Tcl/Tk and more*. Chichester, England: John Wiley & Sons.

APPENDIX A. SURVEY INSTRUMENT

http://userpages.umbc.edu/~gimunoz1/Appendix%20A-%20Survey%20Instrument.pdf

APPENDIX B. CHERRIES

http://userpages.umbc.edu/~gimunoz1/Appendix%20B-%20CHERRIES.pdf

APPENDIX C. INTERVIEW GUIDE PROTOCOL

http://userpages.umbc.edu/~gimunoz1/Appendix%20C-%20Interview%20Guide.pdf

This work was previously published in International Journal of Healthcare Information Systems and Informatics, Vol. 3, Issue 3, edited by J. Tan, pp. 16-37, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 4.15
Open Source Software:
A Key Component of E-Health in Developing Nations

David Parry
Auckland University of Technology, New Zealand

Emma Parry
University of Auckland, New Zealand

Phurb Dorji
Jigme Dorji Wangchuck National Referral Hospital, Bhutan

Peter Stone
University of Auckland, New Zealand

ORGANIZATION OF THIS ARTICLE

This article is organized around a number of sections. The introduction outlines the rationale of the article and deals with some aspects of open source software (OSS) that make it attractive for software development in the health domain for low-income countries. The methodology section then introduces the framework of assessment that is being used. The majority of this article describes a case study of a project run by the authors in Bhutan in the obstetric domain. Critical success factors for such a project are then analyzed and some conclusions are drawn. The discussion covers some of the issues that have arisen from this experience, and articulates some lessons learned.

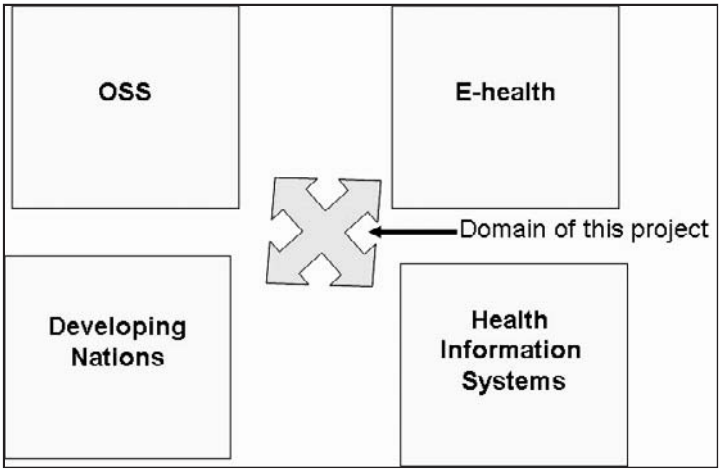
INTRODUCTION

This project deals with the intersection of a number of domains, as shown in Figure 1.

E-Health

E-health has become a popular term for the transformation of healthcare that has occurred through the use of electronic communications,

Figure 1. Research domains



in a conscious imitation of "e-business." E-health encompasses more than the traditional electronic health record. It involves the use of information and communications technologies in the widest sense, including telemedicine, Web-based health and mobile devices for healthcare. A definition has been proposed, after comprehensive analysis, in Pagliari et al. (2005):

E-health is an emerging field of medical informatics, referring to the organization and delivery of health services and information using the Internet and related technologies. In a broader sense, the term characterizes not only a technical development, but also a new way of working, an attitude, and a commitment for networked, global thinking, to improve healthcare locally, regionally, and worldwide by using information and communication technology.

This definition is actually adapted from a previous one in an editorial which discussed the scope of "e-health" (Eysenbach, 2001). The globalized and networked aspects are particularly important in our case study—the emphasis is on communication and collaboration rather than distance

Health Information Systems

Health information systems (HIS) often have three main objectives: to improve patient care, improve management and form part of a quality improvement program. However, these objectives—as described by Littlejohns, Wyatt and Garvican (2003), are not always achieved. As part of a HIS implementation there are often major changes to workflow and practice, large expenditures on hardware including computing and communications, and system integration, as well as software development, training and implementation. Failures occur in HIS development often due to a lack of understanding of the complexity of the project (Littlejohns et al., 2003). Interestingly, OSS appears to answer some of these issues by providing more stable—if less feature-rich—software, and providing a generally larger pool of developers and users than for proprietary software.

Open Source Software

Open source software (OSS) has gained very wide acceptance particularly in the Web server community. Projects such as Apache (Mockus,

Fielding, & Herbsleb, 2000) have involved large scale participation, and dominant market share. In the healthcare domain, Sourceforge.net lists 58 applications for download. Many of the applications are extremely specialized, but on the other hand, some like the Web Interface Repository server (WIRM), (Jakobovits, Rosse, & Brinkley, 2002) are effectively complete development environments. This article will argue that successful development and use of OSS in healthcare requires a number of critical "success" factors, and that these reflect both the nature of OSS projects in the wider world and particular aspects relevant to healthcare. OSS can be seen as part of a wider movement that has been characterized as innovation from the user community (Hippel, 2001). This emphasizes the point that OSS is not just "free" but also is able to be modified by the community that uses it.

DEVELOPING NATIONS AND THE CASE OF BHUTAN

Health information systems are important for developing nations as well as industrialized ones. A large review of the use of information technology in primary care in developing countries (Tomasi, 2004) identified five main areas of application—data processing in the healthcare system, decision support, electronic data transmission, electronic patient records and telemedicine. Many developing countries have low levels of trained clinical staff, and this can increase the load on secondary and tertiary providers. In order to audit their performance, and increase efficiency, electronic records and workflow systems can reduce the workload on the staff available.

Both of these aspects are particularly important for developing nations for a number of reasons:

- Developing nations have extremely limited health budgets, but the burden of disease on individual households can be very large. For example, up to 100% of household income being spent on end-stage care for AIDS patients in some nations in Africa (Russell, 2004).
- Developing nations often have a diverse mixture of groups within them, and it cannot be assumed that all citizens have a common language. Even when a common language exists, it may be spoken by a relatively small percentage of the world's population, and commercial development of software using that language may not be feasible.
- Infrastructure and resource constraints, in particular for network connectivity, may reduce the utility of high-performance systems routinely used in the west. For example, PACS systems involving transmission of large images via network connections may not be practical, but memory stick-based approaches may be feasible (Parry, Sood, & Parry, 2006).
- Open source approaches allow the development of expertise in multiple sites away from large commercial organizations. Therefore, they can encourage the upskilling of software developers in smaller centers. This expertise can be applied to the localization of standard packages and the development of a solid base for software support. In this aspect, both the development and the use of open source software can be beneficial in the education sector. Developing nations often have large and increasing numbers of young, educated people available for project development. OSS tools are attractive for teaching information systems development because of cost, wide availability of documentation and localized versions being available. For example, Debian translators are available in over 80 l language versions including Dzonga—the national language of Bhutan.
- Commercial software suppliers may be reluctant to sell advanced software packages in

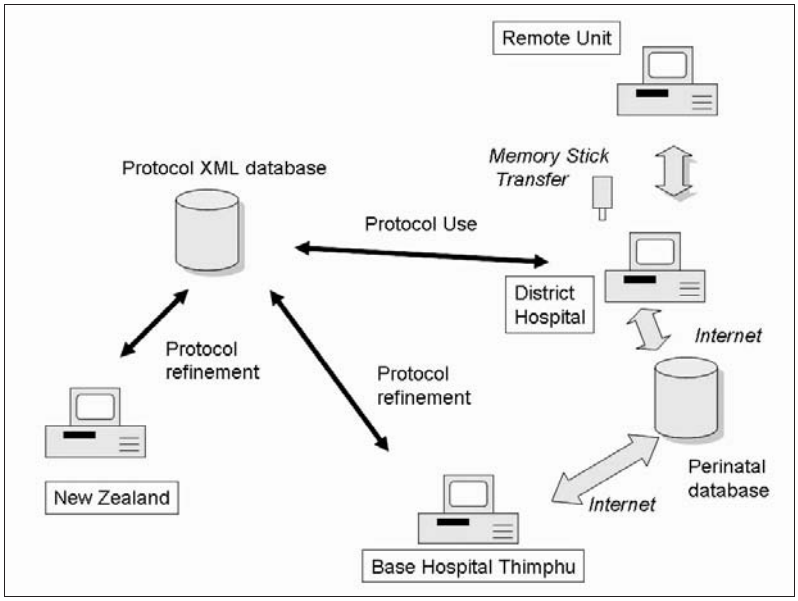
developing nations because of the difficulty of arranging support and the perceived threat of piracy.

- Developing nations' health systems often have a complex collection of groups working within them including governments, commercial organizations, local and international charities and international official organizations. The requirement for reporting and data analysis may well be more complex than in industrialized nations.
- Infrastructure developed to support call-center development or tourism, including Internet and telecommunications technology, is easily adapted to allow links between nations. Because OSS tools are supported via the Web, this approach avoids the reliance on expensive and out-of-date paper manuals and development kits. OSS's licensing structure allows cross-national projects to be completed much more easily.
- Mobile devices have particular promise for e-health in developing nations (Iluyemi, Fitch, Parry, & Briggs, 2007). Mobile OSS development is a particularly active area of research (Raento, Oulasvirta, Petit, & Toivonen, 2005).

The case study in this article deals with the intersection of a number of research domains, (Figure 1), which means that the choice of methodology for analysis may be challenging. There have been some recent papers on the use of OSS in health information systems, focused mainly on developed country applications (Kantor, Wilson, & Midgley, 2003; McDonald et al., 2003). Interestingly, these papers point out that the use of OSS in healthcare is not new and that although perhaps small-scale, this work has been consistently ongoing. However, these papers do emphasize the potential gains to be had by the use of 0SS in healthcare both in terms of health providers and also developers.

Because of the widely varying state of communications infrastructure in developing countries, western models of development which emphasize in-hospital systems linked by fixed line, high-capacity networks may not be appropriate. In the context of less developed countries, there have

Figure 2. Overall system plan



been a number of telemedicine projects, often concerned with communication from centers of excellence in western nations (e.g., Swinfen, Swinfen, Youngberry, & Wootton, 2005), or within developing countries (Deodhar, 2002), but a shared approach to development is vital (Wooton, 2001).

Methodology

In order to analyze a case study, some sort of framework of analysis should be adopted. The development project was actually quite complex with elements of telemedicine; knowledge management and information processing included in the overall design (see Figure 2).

A survey of telemedicine projects in India (Pal, Mbarika, Cobb-Payton, Datta, & McCoy, 2005) identified six critical success factors for telemedicine success; these were used as practical and simple measures that could be applied in this complex, yet small-scale project. The success factors identified were:

1. Set clear program objectives
2. Garner government support
3. Adapt user-friendly interfaces
4. Determine accessibility via telecommunications and Internet access
5. Implement standards and protocols
6. Measure cost-effectiveness and user satisfaction

The case study will deal with these areas, although the project is wider than a simple telemedicine project as it includes database development and integration with the audit system, along with Web-based protocols.

CASE REPORT: E-HEALTH SUPPORT FOR OBSTETRIC SERVICES IN BHUTAN

Bhutan is a small Buddhist kingdom located in the Himalayas, with a population of fewer than 700,000. Land transport is extremely slow because of the geography—for example it takes three days to travel across the country, a distance of around 300km. There is only one airport and no facility for helicopter transport. Seventy percent of the population live in rural areas with 30% more than a one-hour walk from the nearest road head. Bhutan has had major successes in increasing life expectancy and improving healthcare, but avoidable perinatal and maternal mortality and morbidity remains an issue. Current figures for Bhutan suggest an infant mortality rate of 67/1000 and a maternal mortality rate of 4.2/1000—compared with New Zealand's rates of 4/1000 and 0.07/1000 respectively (World Health Organization, 2006a).

Large numbers of preventable neonatal deaths continue to occur in the less developed countries. However, recent work has suggested (Darmstadt et al., 2005) that evidence- based interventions in antenatal and intrapartum care could reduce these rates by between 37% and 67%. These interventions are not complex and are relatively inexpensive. The overarching imperative is to ensure appropriate care for pregnant women that involves patient education and cooperation with antenatal and intrapartum services. Although there have been many studies on the use of e-health in obstetrics and perinatology and in less developed nations (Deodhar, 2002), there remains relatively little work on the evaluation of these systems. In particular, the outcome and integration

of these systems into existing structures, and the changes that occur because of their introduction have not been studied, although the 1 to 45 cost benefit ratio quoted in this study is impressive.

The World Health Organization has been running a "Making Pregnancy Safer" Initiative (World Health Organization) in order to reduce the level of neonatal and maternal mortality. Previous work in Bhutan had developed a protocol book for emergency obstetric care (EmOC). Other countries using EMOC have recorded improvement in outcomes, for example, Bangladesh (Islam MT, 2006) and Peru (Kayango et al., 2006). One of the major lessons learned in these trials was that a record of outcomes via a perinatal database, and the wide dissemination and use of protocols are vital for success. In general it is important that all health workers caring for pregnant women use health information that is based on clear evidence from rigorous studies (Tita, Stringer, Goldenberg, & Rouse, 2007). Surprisingly perhaps, the identification of appropriate procedures for dealing with high-risk patients has been shown to be effective in reducing the demand for interventions (Islam MT, 2006).

The Bhutanese Health System

Healthcare is free in Bhutan, and is delivered via a tertiary structure. The primary healthcare unit is the "Basic Health Unit" (BHU), of which there are approximately 170 around the country. These units do not always have medically qualified staff, but they run outreach and clinic services and usually a number of beds are available. Birthing services, run by nurses, are sometimes available. District health units (around 30) will have at least one generalist medical officer; some of these units have the capacity to perform caesarean sections and ultrasound scanning. There are three referral hospitals in the country which have at least one obstetrician and theatre services. The Jigme Dorji Wanchuck National Referral hospital (JDWNRH) in the capital has four obstetricians and is the tertiary referral unit for the whole country. Because there is no general practitioner service, patients have the opportunity to refer themselves directly to hospital-based consultants even in cases where primary care would be more appropriate. This and the paucity of qualified obstetricians, results in a large workload and the obstetricians are busy and often difficult to contact for advice. A current project is running to introduce the EmOC system to Bhutan, and the protocols are being integrated with these to provide seamless care.

Project Description

The aim of the project described in this article was to collaboratively develop a number of treatment and/or diagnostic protocols to allow the clinical staff to apply appropriate evidence-based care for the major problems that would be dealt with by a perinatal service. The role of the perinatal service is described in Mascarenhas, Eliot and MacKenzie (1992). Essentially, it provides care for mothers and babies from conception to birth, and aims to reduce the risks to mother and baby in this process, by appropriate intervention and monitoring. In addition to the staff applying the protocols in practice, the aim is to raise awareness of the issues that affect perinatal outcomes amongst others, for example, referring to clinicians. The development of a collegial editing and review process involving clinical staff in Bhutan and New Zealand was also seen as a vital part of the project. The project also included the development of a perinatal Web-based database to allow for more effective management of the service on a day-to-day basis and also allow for analysis of clinical performance.

OSS occurs in a number of places in the system. The perinatal database is written in PHP with a mySQL database engine. Web page development was done using open source tools, as was the XML protocol development. However, proprietary products were used for the operating systems and Web server software in Bhutan, along with the Linux/Apache

setup for the Web server in New Zealand. In addition, standard proprietary products were used to develop the protocols.

Review of Success Factors

Clear Objectives

The objectives of the project were identified in initial discussions and codified in the agreement signed between the stakeholders. The objectives included the development of a perinatal medicine service, continuing support for this service and standardization of treatment based on the best possible evidence. An additional objective was sustainability of the service. OSS supported this by allowing low- or no-cost technical documentation and development tools to be made available to local staff.

Government Support

The Royal Government of Bhutan (RGOB) is the sole supplier of healthcare in the kingdom. The RGOB runs a series of five-year plans which identify objectives and priorities as well as sources of funding. Plans developed by overseas providers are examined and extensive negotiation takes place to ensure that the country receives appropriate and sustainable help that is consistent with the RGOB objectives. This process began in the case of this project, two years before the initiation, when representatives of the funders—The Magee Family—met with other stakeholders including government representatives, clinical staff from New Zealand and Bhutan associated with the project, and UNICEF. This resulted in a project agreement that was signed off in a formal ceremony. The project composed a number of other elements including funding for hardware and training of clinical workers in the perinatal medicine area. Continuing involvement of the stakeholders has been a great asset to the project. The RGOB department of IT has been running a long-term project to support OSS and is getting closer to the development of a policy on its use (Bhutan Department of Information Technology, 2007).

Adapt User-Friendly Interfaces

The user interface adopted was a standard Web browser, whatever the source of the data—even locally stored protocols would display in a browser. The native protocols were stored as XML documents, which were then displayed in a human-readable format via a Web browser. XML was chosen for ease of updating—in that the editing

Figure 3. XML protocol fragment

```
<Root_Element>
  <Name>Cord prolapse</Name>
  <Definition>
The cord that normally presents itself is within intact membranes. When the membranes rupture, the cord
        prolapses. This is an emergency as cord compression and/or occlusion can cause fetal asphyxia.
 </Definition>
  <Keywords>
    <Keyword>Rupture of Membrane</Keyword>
    <Keyword>Prolapse</Keyword>
  </Keywords>
  <Diagnosis>
    <Diagnostic_step>Palpable cord on vaginal exam</Diagnostic_step>
    <Diagnostic_step>Observed cord protruding onto vulva</Diagnostic_step>
  </Diagnosis>
</Root_Element>
```

process could alter content without a great deal of formatting issues and with the awareness that other display methods such as voice responses or mobile devices may be used in the future. As an open standard, XML is very well suited to this approach. The XML design is intended to be expandable and able to represent both diagnostic and therapeutic protocols.

A fragment of the XML representation is shown in Figure 3. The initial outline was based on the PubMED (may need to define this) schema, but simplified to remove excessive bibliographic elements. The XML documents identify the responses to particular diagnoses or symptoms which would be expected to be encountered commonly. The aim is to allow clinical staff, who may be at a remote site, to identify what emergency care is needed, whether the patient needs to be referred or transferred and the degree of urgency of that referral. Also, the protocol can identify what additional tests or procedures need to be performed.

Determine Accessibility

Although land transport is difficult, Bhutan is in the process of increasing the availability of Internet access. Apart from dial-up connections, there are microwave links and recently the international telecommunications union (ITU) e-post initiative (International Telecommunications Union, 2006) has recently been launched in Bhutan using very small aperture (VSAT) satellite ground stations for rural access to electronic communications, and this may be useful for rollout to remote areas. OSS tools are often very efficient in terms of file size, and machine footprint (use of processor time and memory) so they can be used in a wider range of scenarios than might be possible for the latest proprietary operating systems or applications.

Implement Standards

The protocols themselves were developed in a standard format (Table 1), and as seen above, implemented using XML. In addition to this, an attempt was made to standardize the production of the protocols so that candidate protocols from other sources would go through an editorial process and be routinely revised. This process was formally followed using paper-based systems, but electronic approaches allow instantaneous updating of the live protocol without fear of version control issues and also allow a trail to be kept of previous versions that can be linked to any events linked historically to the implementation.

Measuring Cost-Effectiveness and User Satisfaction

This aspect is perhaps the most difficult part of the project. As part of the process, proto-

Table 1. Major elements of the protocol document

Element	Comment
Name	Name of protocol
Definition	
Keywords	Used for searching
Diagnosis	For diagnostic protocols
Diagnostic Step	
Procedure	For procedural protocols
Procedure Step	
Audience	Intended user, includes country and location
Evidence	A small selection of the supporting evidence
Author	Multiple authors possible
Last Update	
Review Date	

cols will be regularly reviewed by stakeholders. In addition, a perinatal database is being implemented in order to record outcomes, and assess performance against that expected in the protocol, in particular, areas where the protocols are not being followed, and whether the protocols or behavior or both should be modified. It is hoped that improvements in the mortality and morbidity figures will also be noticeable, because of the currently relatively high rates of morbidity. Substitute measures, such as adherence to protocol may also be used. Finally, a rise in awareness of the general maternity service and increased access to it by women, only half of whom currently have an attended birth, would be expected to accompany improved outcomes among those who have contact with the perinatal service.

Lessons Learned

Integration of protocols from diverse sources was one of the major challenges facing the team. Protocols were sourced from the National Women's Hospital Auckland, New Zealand, the World Health Organization and EmOC protocols in Bhutan.

Collaborative review of protocols was extremely important, as buy-in from clinical staff is vital. However, the process of maintaining a common electronic repository was technically difficult as each of the reviewers tended to work asynchronously using paper copies. The final approach used was to produce paper prototypes and distribute them, collect back annotated versions and then combine them in a final Word document. This was then converted to XML. Development of the perinatal database was restricted by the very small numbers of users available to test and comment on the system, and a wide user community, which may only be available online, may well increase the quality and speed of development..

DISCUSSION AND FUTURE WORK

There are some general issues that affect e-health initiatives, and the use of OSS in the developing world, in particular, connectivity, computing resources and skills.

Connectivity

Less developed nations have generally much lower availability of fixed telephone lines. In addition, geographic, economic and governmental issues often conspire to make conventional dial-up access less common than in western countries. However, wireless and satellite solutions such as VSATS including the international telecommunications union (ITU) e-post initiative (International Telecommunications Union, 2006) are overcoming these issues. It is important to recognize that not every nation's infrastructure is developing in the same way, and many nations may leapfrog to wireless solutions without the use of landline-based solutions. However, high bandwidth solutions may not be appropriate for developing countries. One of the most successful e-health projects has been the Swinfen Project, currently expanding in Iraq (Swinfen et al., 2005). This project uses e-mail in a store-and-forward model, between clinicians in various countries. The prospects of advanced tele-presence approaches being effective in routine care seem slight because of issues concerning quality of service, bandwidth and reliability of connection. Even though the "trauma pod" and other projects financed by the U.S. Department of Defense are beginning to show results (Romano, Lam, Moses, Gilbert, & Marchessault, 2006), costs are likely to render this approach problematic in other contexts.

Computing

Devices such as the Simputer (The Simputer Trust, 2000) and the sub $100 laptop (OLPC, 2006) promise much cheaper access to computing power.

It should be emphasized that for e-health applications, the computing device can be fairly simple; indeed mobile devices may become the preferred means of access. Along with cost, the ability to survive rough treatment, extremes of temperature and humidity and long battery life—or even the use of clockwork power in the case of the sub $100 laptop—are more important in developing countries than in the organization for economic cooperation and development (OECD) member countries. Parts supply and transport cost can make the repair of computers extremely expensive. However, organizations such as global assistance for medical equipment (GAME) (http://www.global-medical-equipment.org/whatwedo.html) have established links between professional organizations in the developed and less developed world. These approaches move beyond the shipping of obsolescent equipment to an integrated and well-thought and sustainable out collaboration between donors and recipients.

Skills and Information

At present, consumer e-health is of limited usefulness in the developing world. Low levels of literacy and information literacy cause difficulties. However, the fact that the vast majority of Web resources are written in English and are U.S.-centric in terms of organization of healthcare, availability of drugs and medical devices and naming, makes even materials designed for health consumers in the OECD countries less useful for those in other nations. However, these issues are much less important when the provision of e-health services for medical professionals is considered. Adapting general principles to specific cases is a key skill of medical professionals. Indeed the traffic is not all one-way; less developed nation professionals often have skills that are no longer available in more developed nations. Collaboration in training of medical professionals, where trainees from different nations are exchanged, can improve the training in both systems. This can be supported by the use of e-health tools such as Web sites, e-mail and instant messaging.

Other skills required include the support of the e-health infrastructure in terms of technical support for computing devices and connectivity. Fortunately, the requirement of tourists from western countries for Internet connectivity, wherever they are, along with the burgeoning industries of call centers and 'off-shoring' of software development are providing a strong push for training in these areas.

OSS use in education and training allows nations with limited resources to devote more funding to the human side of education, as well as allowing projects that involve software localization to advance quickly. Open source clinical protocols may become important repositories of clinical knowledge allowing rapid development and input from experience, especially based on standard electronic forms.

Another important aspect of skill transfer and collaboration is the use of early warning networks for disease surveillance such as the Global Outbreak Alert and Response Network (GORAN) that played a very large part in the early detection of SARS (Heymann & Rodier, 2004). Such networks link health workers throughout the world and the transfer of information is by no means one-way.

There remains a dearth of well-controlled studies of e-health initiatives in developing nations, but the need for effective collaboration remains paramount (Wooton, 2001). However, there are a number of pointers to success:

1. The e-health system must be compatible with existing organizational and cultural structures. Some "western" assumptions do not apply in less developed nations and vice versa. For example, routine ultrasound examination in early pregnancy has not been shown to be effective in reducing mortality in a Cochrane review (Neilson, 1998). How-

ever, an environment where mortality due to unsuspected problems is much greater, and the availability of on-demand scans is lower, may give different results.

2. Collaboration and training between the professionals involved is vital. This applies to both clinical and technical staff. This may in fact be the area of greatest benefit.
3. Ingenuity is more important than technology. Store-and-forward e-mail may be of greater utility than tele-presence.
4. Open source technology is particularly suited to this area of work. Lower costs, availability of technical skills, greater range of customized languages and often lower technology requirements make open source approaches and especially Web-based open source tools particularly attractive.

FUTURE RESEARCH

Future work in this area will include greater use of multicenter collaboration, both within existing networks such as GORAN and GAME and outside them. Lower bandwidth costs, and easier access to high bandwidths will enable richer media to be used, such as tele-sonography via store-and-forward (Parry et al., 2006). Common health problems are starting to afflict north and south: aging populations, the rapid spread of new infectious diseases and chronic conditions. Common approaches to these issues, including the use of low-cost assistive technology, and offshoring of medical procedures such as radiology (Larson & Janower, 2005), may be controversial, but at least the discussion has started. There are enormous potential benefits in the development of e-health in collaboration with developing nations, and the benefit to the people of world may be immense. An additional benefit of the open source approach may be an increased ability for IT specialists in developed nations to assist people around the world. As virtually every nation now has a Web presence, the technical barriers to such collaboration are much lower than they were even ten years ago. It is hoped that further work will refine the system sufficiently to allow the software to be placed in a repository such as Sourceforge.net. Furthermore, it is hoped that such an approach will encourage increasing collaboration and development in this area.

ACKNOWLEDGMENT

This project would not have been possible without the generosity of the Magee family and the hard work of the representatives of UNICEF and the Royal Government of Bhutan. Staff of Jigme Dorji Wanchuck National Referral hospital, and Thimphu assisted in a very wide range of roles and continue to work on this project

REFERENCES

Bhutan Department of Information Technology. (2007). *Bhutan's journey towards open source*. Paper presented at the DebConf7, Edinburgh, Scotland. https://penta.debconf.org/~joerg/events/25.en.html.

Darmstadt, G., Bhutta, Z., Cousens, S., Adam, T., Walker, N., & Berni, L. (2005). Evidence-based, cost-effective interventions: How many newborn babies can we save?. *The Lancet, 365*(9463), 977-988.

Deodhar, J. (2002). Telemedicine by e-mail—experience in neonatal care at a primary care facility in rural India. *Journal of Telemedicine and Telecare, 8*(Suppl 2), 20-21.

Eysenbach, G. (2001). What is e-health?. *J Med Internet Res, 3*(2), e20.

Heymann, D., & Rodier, G. (2004). Global surveillance, national surveillance, and SARS. *Emerging Infectious Diseases Journal, 10*(2).

Retrieved December 1, 2006, from http://origin.cdc.gov/ncidod/EID/vol10no2/03-1038.htm.

Hippel, E. (2001). Innovation by user communities: Learning from open source software. *MIT Sloan Management Review, 42*(4), 82.

Iluyemi, A., Fitch, C., Parry, D., & Briggs, J. (2007). *Health information system for community-based health workers: A case for mobile and wireless technologies.* Paper presented at the IST-Africa, Maputo Mozambique.

International Telecommunications Union. (2006). *Bhutan to be testbed for ITU's e-post venture with universal postal union*. Retrieved April 20, 2006, from http://www.itu.int/newsarchive/press_releases/2002/10.html.

Islam MT, H. Y., Waxman R, Bhuiyan AB. (2006). Implementation of emergency obstetric care training in Bangladesh: Lessons learned. *Reproductive Health Matters, 14*(27), 61-72.

Jakobovits, R., Rosse, C., & Brinkley, J. (2002). WIRM an open source toolkit for building biomedical Web applications. *J Am Med Inform Assoc, 9*(6), 557-570.

Kantor, G., Wilson, W., & Midgley, A. (2003). Open source software and the primary care EMR. *Journal of the American Medical Informatics Association, 10*(6), 616.

Kayango, M., Esquiche, E., Luna, M., Frias, G., Vega-Centeno, L., & Bailey, P. (2006). Strengthening emergency obstetric care in Ayacucho, Peru. *International Journal of Gynecology and Obstetrics, 92*, 299-307.

Larson, P., & Janower, M. (2005). The nighthawk: Bird of paradise or albatross?. *Journal of the American College of Radiology, 2*(12), 967-970.

Littlejohns, P., Wyatt, J., & Garvican, L. (2003). Evaluating computerised health information systems: Hard lessons still to be learnt. *British Medical Journal, 326*(7394), 860-863.

Mascarenhas, L., Eliot, B., & MacKenzie, I. (1992). A comparison of perinatal outcome, antenatal and intrapartum care between England and Wales, and France. *Br.J.Obstet.Gynaecol., 99*(12), 955-958.

McDonald, C., Schadow, G., Barnes, M., Dexter, P., Overhage, J., Mamlin, B., et al. (2003). Open source software in medical informatics—why, how and what. *International Journal of Medical Informatics, 69*(2-3), 175-184.

Mockus, A., Fielding, R., & Herbsleb, J. (2000). *A case study of open source software development: The Apache server.* Paper presented at the Software Engineering, 2000.

Neilson, J. (1998). Ultrasound for fetal assessment in early pregnancy. *Cochrane Database of Systematic Reviews* (4).

OLPC. (2006). *One laptop per child*. Retrieved December 1, 2006, from http://laptop.org/.

Pagliari, C., Sloan, D., Gregor, P., Sullivan, F., Detmer, D., Kahan, P., et al. (2005). What is e-health (4): A scoping exercise to map the field. *J Med Internet Res, 7*(1), e9.

Pal, A., Mbarika, V., Cobb-Payton, F., Datta, P., & McCoy, S. (2005). Telemedicine diffusion in a developing Country: The case of India. *Information Technology in Biomedicine, IEEE Transactions on, 9*(1), 59-65.

Parry, E., Sood, R., & Parry, D. (2006). Investigation of optimization techniques to prepare ultrasound images for electronic transfer [abstract]. *Ultrasound in Obstetrics and Gynecology, 28*(4), 487-488.

Raento, M., Oulasvirta, A., Petit, R., & Toivonen, H. (2005). ContextPhone: A prototyping platform for context-aware mobile applications, *4*, 51-59.

Romano, J., Lam, D., Moses, G., Gilbert, G., & Marchessault, R. (2006). The future of military medicine has not arrived yet, but we can see it

from here. *Telemedicine and e-Health, 12*(4), 417-425.

Russell, S. (2004). The economic burden of illness for households in developing countries: A review of studies focusing on malaria, tuberculosis, and human immunodeficiency virus/acquired immunodeficiency syndrome. *Am J Trop Med Hyg, 71*(2_suppl), 147-155.

Swinfen, P., Swinfen, R., Youngberry, K., & Wootton, R. (2005). Low-cost telemedicine in Iraq: An analysis of referrals in the first 15 months. *Telemedicine and Telecare, 11*(Suppl 2), 113.

The Simputer Trust. (2000). *Simputer(TM) - Welcome*. Retrieved December 1, 2006, from http://www.simputer.org/.

Tita, A., Stringer, J., Goldenberg, R., & Rouse, D. (2007). Two decades of the safe motherhood initiative: Time for another wooden spoon award?. *Obstet Gynecol, 110*(5), 972-976.

Tomasi, E., Augusto, L., & Maria de Fatima Santos, M. (2004). Health information technology in primary healthcare in developing countries: A literature review. *Bull World Health Organ, 82*(11), 867-874.

Wooton, R. (2001). Telemedicine and developing countries—successful implementation will require a shared approach. *Telemedicine and Telecare, 7*(Suppl 1), 1-6.

World Health Organization. (2006a). *Estimates of child and adult mortality and life expectancy at birth by country.* Retrieved January 17, 2007, from http://www.who.int/healthinfo/statistics/mortlifeexpectancy/en/index.html.

World Health Organization. (2006b). *Introduction to the 'Making Pregnancy Safer' initiative.* Retrieved October 1, 2007, from http://w3.whosea.org/pregnancy/introf.htm.

This work was previously published in International Journal of Healthcare Information Systems and Informatics, Vol. 3, Issue 3, edited by J. Tan, pp. 1-15, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 4.16
Patchwork Prototyping with Open Source Software

M. Cameron Jones
University of Illinois at Urbana-Champaign, USA

Ingbert R. Floyd
University of Illinois at Urbana-Champaign, USA

Michael B. Twidale
University of Illinois at Urbana-Champaign, USA

ABSTRACT

This chapter explores the concept of patchwork prototyping: the combining of open source software applications to rapidly create a rudimentary but fully functional prototype that can be used and hence evaluated in real-life situations. The use of a working prototype enables the capture of more realistic and informed requirements than traditional methods that rely on users trying to imagine how they might use the envisaged system in their work, and even more problematic, how that system in use may change how they work. Experiences with the use of the method in the development of two different collaborative applications are described. Patchwork prototyping is compared and contrasted with other prototyping methods including paper prototyping and the use of commercial off-the-shelf software.

INTRODUCTION

The potential for innovation with open source software (OSS) is unlimited. Like any entity in the world, OSS will inevitably be affected by its context in the world. As it migrates from one context to another, it will be appropriated by different users in different ways, possibly in ways in which the original stakeholders never expected. Thus, innovation is not only present during design and development, but also during use (Thomke & von Hippel, 2002). In this chapter, we explore an emerging innovation through use:

a rapid prototyping-based approach to requirements gathering using OSS. We call this approach *patchwork prototyping* because it involves patching together open source applications as a means of creating high-fidelity prototypes. Patchwork prototyping combines the speed and low cost of paper prototypes, the breadth of horizontal prototypes, and the depth and high functionality of vertical, high-fidelity prototypes. Such a prototype is necessarily crude as it is composed of stand-alone applications stitched together with visible seams. However, it is still extremely useful in eliciting requirements in ill-defined design contexts because of the robust and feature-rich nature of the component OSS applications.

One such design context is the development of systems for collaborative interaction, like "cybercollaboratories." The authors have been involved in several such research projects, developing cyberinfrastructure to support various communities, including communities of learners, educators, humanists, scientists, and engineers. Designing and developing such systems, however, is a significant challenge; as Finholt (2002) noted, collaboratory development must overcome the "enormous difficulties of supporting complex group work in virtual settings" (p. 93). Despite many past attempts to build collaborative environments for scientists (see Finholt for a list of collaboratory projects), little seems to have been learned about their effective design, and such environments are notorious for their failure (Grudin, 1988; Star & Ruhleder, 1996). Thus, the focus of this chapter is on a method of effective design through a form of rapid, iterative prototyping and evaluation.

Patchwork prototyping was developed from our experiences working on cybercollaboratory projects. It is an emergent practice we found being independently redeveloped in several projects; thus, we see it as an effective ad hoc behavior worthy of study, documentation, and formalization. Patchwork prototyping is fundamentally a user-driven process. In all of the cases where we saw it emerge, the projects were driven by user groups and communities eager to harness computational power to enhance their current activities or enable future activities. Additionally, the developers of the prototypes had no pretence of knowing what the users might need a priori. As a result, patchwork prototyping's success hinges on three critical components:

1. Rapid iteration of high-fidelity prototypes
2. Incorporation of the prototypes by the end users into their daily work activities
3. Extensive collection of feedback facilitated by an insider to the user community

In this chapter, we focus on how the method worked from the developers' point of view. It is from this perspective that the advantages of using OSS are most striking. However, one should bear in mind that the method is not just a software development method, but also a sociotechnical systems (Trist, 1981) development method: The social structures, workflows, and culture of the groups will be coevolving in concert with the software prototype.

REQUIREMENTS GATHERING IN COLLABORATIVE SOFTWARE DESIGN

Software engineering methods attempt to make software development resemble other engineering and manufacturing processes by making the process more predictable and consistent. However, software cannot always be engineered, especially Web-based applications (Pressman et al., 1998). Even when application development follows the practices of software engineering, it is possible to produce applications that fail to be used or adopted (Grudin, 1988; Star & Ruhleder, 1996). A major source of these problems is undetected failure in the initial step in building the system: the requirements-gathering phase. This is the

most difficult and important process in the entire engineering life cycle (Brooks, 1995).

In designing systems to support collaborative interaction, developers are faced with several complex challenges. First, the community of users for which the cyberinfrastructure is being developed may not yet exist and cannot be observed for one to see how the users interact. In fact, there is often a technological deterministic expectation that the computational infrastructure being created will cause a community to come into existence. Even in the case where there is a community to study, many of the activities expected to occur as part of the collaboration are not currently being practiced because the tools to support the activities do not yet exist. As a result, developers gain little understanding about how the users will be interacting with each other or what they will be accomplishing, aside from some general expectations that are often unrealistic.

Gathering requirements in such an environment is a highly equivocal task. While uncertainty is characterized by a lack of information, which can be remedied by researching an answer, collecting data, or asking an expert, equivocal tasks are those in which "an information stimulus may have several interpretations. New data may be confusing, and may even increase uncertainty" (Daft & Lengel, 1986, p. 554). Requirements gathering is one such situation in which the developers cannot articulate what information is missing, let alone how to set about obtaining it. The only resolution in equivocal situations is for the developers to "enact a solution. [Developers] reduce equivocality by defining or creating an answer rather than by learning the answer from the collection of additional data" (Daft & Lengel, p. 554). As Daft and Macintosh (1981) demonstrate, tasks with high equivocality are unanalyzable (or rather, have low analyzability; Lim & Benbasat, 2000), which means that people involved in the task have difficulty determining such things as alternative courses of action, costs, benefits, and outcomes.

RAPID PROTOTYPING

Rapid prototyping is a method for requirements gathering that has been designed both to improve communication between developers and users, and to help developers figure out the usefulness or consequences of particular designs before having built the entire system. The goal of rapid prototyping is to create a series of iterative mock-ups to explore the design space, facilitate creativity, and get feedback regarding the value of design ideas before spending significant time and money implementing a fully functional system (Nielsen, 1993). There are several dimensions to prototypes. One dimension is the range from low-fidelity to high-fidelity prototypes (see Table 1; Rudd, Stern, & Isensee, 1996). Low-fidelity prototypes have the advantages of being fast and cheap to develop and iterate. However, they are only able to garner a narrow range of insights. Perhaps the most popular low-fidelity prototyping technique is paper prototyping (Rettig, 1994). Paper prototypes are very fast and very cheap to produce. They can also generate a lot of information about how a system should be designed, what features would be helpful, and how those features should be presented to the users. However, paper prototypes do not allow developers to observe any real-world uses of the system, or understand complex interactions between various components and between the user and the system. Also, they do not help developers understand the details of the code needed to realize the system being prototyped.

High-fidelity prototypes, on the other hand, can simulate real functionality. They are usually computer programs themselves that are developed in rapid development environments (Visual Basic, Smalltalk, etc.) or with prototyping tool kits (CASE, I-CASE, etc). In either case, these prototypes, while allowing programmers to observe more complex interactions with users and to gain understanding about the underlying implementation of the system, are comparatively

Table 1. Advantages and disadvantages of low- and high-fidelity prototyping (Source: Rudd et al., 1996, p. 80)

	Advantages	Disadvantages
Low-Fidelity	• Lower development cost • Can create many alternatives quickly • Evaluate multiple design concepts • Useful communication device • Address screen layout issues • Useful for identifying market requirements • Proof of concept	• Limited error checking • Poor detailed specification to code to • Facilitator driven • Limited utility after requirements established • Limited usefulness for usability tests • Navigational and flow limitations • Weak at uncovering functionality- and integration-related issues
High-Fidelity	• Complete functionality • Fully interactive • User driven • Clearly defines navigational scheme • Use for exploration and tests • Look and feel of final product • Serves as a living specification • Marketing and sales tool	• More expensive to develop • Time consuming to create • Inefficient for proof of concept designs • Not effective for requirements gathering

slow and expensive to produce and iterate (Rudd et al., 1996). These costs can be offset somewhat by incorporating these prototypes into the development of the final system itself as advocated by RAD (rapid application development; Martin, 1991). However, critics of RAD methods are quick to point out the limited scalability of software built using source code from prototypes (Beynon-Davies, Carne, Mackay, & Tudhope, 1999). Typically low-fidelity and high-fidelity prototypes are used in succession, with developers increasing the fidelity of the prototypes as they develop the specifications. Due to their high cost, high-fidelity prototypes may only be built for a select number of designs generated by low-fidelity prototyping, which precludes the generation of a series of disposable high-fidelity proofs of concepts to test out alternative design ideas.

Another dimension to be considered in the prototyping discussion is scope. Software can be viewed as consisting of a number of layers, from the user interface to the base layer, which interacts with the underlying operating system or platform. Horizontal prototypes encompass a wide scope, spanning the breadth of a system but only within a particular layer (usually the user interface). Users can get a sense of the range of the system's available functions; however, the functionality is extremely limited. This can help both the user and the programmer understand the breadth of the system without plumbing its depths. Vertical prototypes, on the other hand, take a narrow slice of the system's functionality and explore it in depth through all layers. This allows users to interact with a particular piece of the system, and gives the programmer a detailed

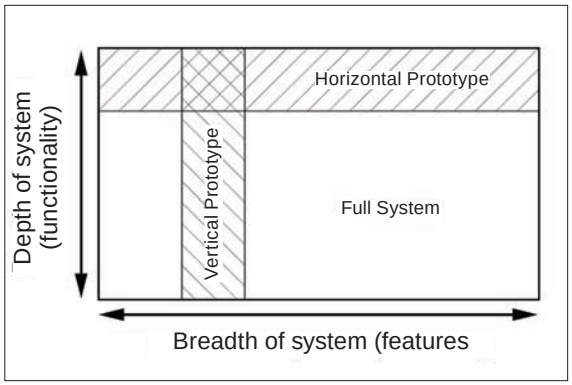
understanding of the subtle issues involved in its implementation (Floyd, 1984; Nielsen, 1993).

The high equivocality present when designing collaborative systems makes it difficult to apply rapid prototyping techniques effectively. Because users may not be able to articulate what they want or need, it helps to be able to collaboratively interact with high-fidelity systems in order to test them in real-world situations and see what requirements emerge. Without such an experience, it is unlikely that any feedback the developers get from the users, either through direct communication or observation, will be useful. Thus, low-fidelity prototypes are limited in their power to elicit requirements as the users have difficulty imagining how the system the prototypes represent will work, what it could do for them, or how they might use it. Also, since the majority of tasks involved in collaboration are quite complex and require multiple kinds of functionality to complete, the users need to be able to interact with the system as a whole and with considerable depth of implementation, thus requiring a prototype that is both horizontal and vertical.

The economics of developing high-fidelity prototypes that are both horizontal and vertical in scope, however, are problematic. Even if the developers were to build a series of high-fidelity, vertical prototypes, they would end up having built the equivalent of an entire system from scratch just to have a functionally sufficient prototype. Not only would it be expensive and time consuming, but the functionality and robustness would be minimal at best. Also, it is likely that the work would need to be discarded and replaced with something new since it is unlikely that the design would be correct on the first, second, or even third try. Thus, the typical methods of prototyping are not sufficient, either because developing all the code would be too expensive, or the prototypes that are developed do not have high enough fidelity.

The proliferation of production-scale OSS systems has created a vast field of growing, reliable, usable, and feature-rich programs, a large number of which support aspects of Web-based collaboration. These programs can be easily stitched together because the code is open and modifiable. Furthermore, they can be treated as disposable since one application can easily be discarded and replaced with another. This presents an opportunity for developers to rapidly build and evaluate a high-fidelity prototype of a collaborative environment comprising a patchwork of multiple open source applications. Such a prototype spans the breadth of a horizontal prototype and the depth of a vertical prototype within a single system.

Figure 1. Horizontal and vertical prototypes (Source: Nielsen, 1993, p. 94)



ORIGINS AND EXAMPLES OF PATCHWORK PROTOTYPING

Patchwork prototyping is a rapid prototyping approach to requirements gathering that was emergent from practice rather than designed a priori. We have been involved with several groups that were developing cyberinfrastructure to support collaboration, and in each group we observed ad hoc prototyping and development strategies

that were remarkably similar and that developed entirely independent of each other. Upon making these observations, we realized that there was a core process at work in each of these projects that could be abstracted out and described as a general approach to requirements gathering for developing cyberinfrastructure. Because patchwork prototyping evolved from practice, however, we believe that it will be much easier to understand our formal description of the approach after we describe some of the relevant details of our experiences. In this section, we describe two projects with which we were involved and the relevant dynamics of each project; in the following section, we describe the patchwork prototyping approach more abstractly.

Project Alpha: Building a Cybercollaboratory for Environmental Engineers

Project Alpha (a pseudonym used to preserve anonymity) was devoted to building a cybercollaboratory for environmental engineers. At the beginning, the project was intended to be a requirements-gathering project, and the goal was to build a functional prototype of the cyberinfrastructure that would be presented to the granting agency as part of a larger proposal. The effort was a success and now, more than a year after the project began, the prototype is being converted into a production-scale system. The cybercollaboratory prototypes were largely designed and built over a period of six months by a team of two developers, with significant contribution to the design by a team of around 12 to 13 other researchers (these researchers, plus the two developers, we call the design team), and some minor programming contributions by undergraduates employed by the project. By the end of the prototyping phase, there was a community of users that included 60 to 70 active users out of approximately 200 registered users, 10 of which comprised a core group of vocal users who provided significant feedback on the design.

The Project Alpha prototype was constructed on the Liferay portal server framework. In addition to using existing portlets, the developers also wrapped other OSS applications in portlet interfaces, enabling their rapid integration into the prototype. A number of different OSS applications were used, including the Heritrix Web crawler, the Lucene search engine, and the MediaWiki wiki system. Other applications were similarly integrated but were not necessarily publicly available OSS. Some were in-house applications developed by other projects for which the developers had source code. These applications were used to prototype data mining and knowledge management functionality in the cybercollaboratory.

The general process by which these tools were incorporated was very ad hoc. The development team might decide on prototyping a particular function, or the programmers might get some idea for a "cool" feature and would set about integrating the feature into the system. This approach had several unexpected benefits. First, minimal time was spent building portlets so that when a version of the prototype was presented to the design team, minimal effort was lost when particular features or portlets were rejected as being unsuitable. Second, it allowed the design team to choose between several different portlets that had essentially the same function but different interfaces (i.e., were optimized for different types of use). Third, it allowed the developers to easily switch features off when the interface for a portlet was too complex, or turn them back on if they were requested by either the design team or the active users. Fourth, the development community and the associated forums, mailing lists, and Web sites surrounding the OSS applications that were integrated into the prototype served as excellent technical support (Lakhani & von Hippel, 2002).

The fact that the prototype was fully functional was critical to its success in eliciting requirements. By using the prototypes over a period of 6 months, the users were able to incorporate them into their day-to-day work practices. This allowed them to evaluate the utility of the tool in various contexts of actual use. Without functionality, the developers feel that it would have been impossible to effectively gather requirements. However, it was also vital that the users communicate their experiences to the developers, both formally and informally. To this end, the developers conducted several surveys of the users, asking them about the prototype and features they found useful. The developers also used the prototype itself to solicit feedback. On the front page of the prototype was a poll asking users to vote for the features they liked the most. Additionally, on every page of the prototype was a feedback form that allowed users to send quick notes about the system as they experienced it. The users also communicated with the developers via informal means such as e-mail and face-to-face meetings. However, the most important method of obtaining feedback was that one of the PIs in the project acted as an intermediary, actively soliciting feedback from users as an insider to the community of environmental engineers. The PI position allowed the individual to receive more feedback of higher quality and honesty than the developers would have been able to collect on their own.

To illustrate the process in more detail, we describe how one particular piece of OSS was integrated with the cybercollaboratory. The developers wanted to allow users to be able to collaboratively edit documents in the system. The Liferay suite had a wiki system available that the programmers enabled; however, users found that tool to be too difficult to use, partly because of the unintuitive markup syntax of the particular wiki used, and partly because they had no tasks that clearly lent themselves to the use of such a tool. Later during the prototyping phase, some members of the design team wanted to demonstrate the usefulness of scenarios and personas in facilitating requirements gathering, and from prior experience suggested the use of a wiki. In response to this request and the prior difficulties in using the bundled tool, the developers installed MediaWiki on the server and added a link from the cybercollaboratory's menu next to the existing wiki tool pointing to the MediaWiki installation. No time was spent trying to integrate the Liferay and MediaWiki systems; each application had separate interfaces and user accounts.

One benefit of using the MediaWiki system was that it allows people to use the system without logging in, thereby mitigating the need to integrate authentication mechanisms. Users found the MediaWiki system easier to learn and use, and began using it exclusively over the in-built Liferay wiki. The developers then decided to embed the MediaWiki interface in the rest of the cybercollaboratory and wrote a simple portlet that generates an HTML (hypertext markup language) IFRAME to wrap the MediaWiki interface. Each step of integrating the MediaWiki installation took only minimal effort on the part of the developers (sometimes literally only a matter of minutes) and generated insights about the role and design of a collaborative editing tool in the cybercollaboratory. Among the design insights gained by the developers is that the tool should be easy to use with a simple syntax for editing. Also, the tool should support alternate views of the data, offering a unified view of all documents either uploaded to the site's document repository or created and edited on the wiki. The users were able to see how this tool could benefit their jobs, and that shaped the requirements of the tool. As a result of this process, the project is currently implementing a new collaborative editing component. This component will have features like integrated authentication, group- and project-based access control, and integration with other features (e.g., project views and wiki linking). Additionally,

the new collaborative writing component will deprecate redundant and confusing features like in-wiki file uploads.

Project Beta: Building Collaborative Tools to Support Inquiry-Based Learning

Project Beta is an ongoing research project aimed at designing and building Web-based tools to support processes of inquiry as described by John Dewey (Bishop et al., 2004). Initiated in 1997, the project has embraced a long-term perspective on the design process and produced a series of prototypes that support inquiry-based teaching and learning. In 2003 the project began exploring the development of tools to support collaborative inquiry within groups and communities. The current prototype is the third major revision of the collaborative cyberinfrastructure, with countless minor revisions on going. Throughout the project's life span, several generations of programmers have joined and left the development team. For a 30-month stretch, the majority of programming was sustained by a single graduate-student programmer. Between four and eight other researchers filled out the design team.

The prototypes are available for anyone to use, and the source code is also distributed under a Creative Commons license. To date, the prototypes have been used to support a large number of communities of users ranging from water-quality engineers to volunteers in a Puerto Rican community library in Chicago, from researchers studying the honeybee genome to undergraduates in the social sciences. There are numerous other groups using the system for any number of purposes. Given this scenario, it is practically impossible to design for the user community or any intended use.

The prototypes were developed in the PHP programming language on an open source platform consisting of Apache, MySQL, and RedHat Linux. In contrast to Project Alpha where the developers initially did very little programming and primarily used readily available tools, the developers of Project Beta spent considerable effort building an infrastructure from scratch, in part because the developers were initially unaware of relevant OSS. However, as the project progressed, several open source tools were incorporated into the prototypes including the JavaScript-based rich-text editors FCKEditor and TinyMCE, the phpBB bulletin board system, and MediaWiki.

To demonstrate the process in more detail, we describe how one particular piece of OSS was integrated with the prototypes. In the earliest version of the cyberinfrastructure, users expressed an interest in having a bulletin board system. The developers selected the phpBB system and manually installed copies of phpBB for each community that wanted a bulletin board; the bulletin board was simply hyperlinked from the community's home page. In the next iteration of the prototype, the phpBB system was modified to be more integrated with the rest of the prototype. Users could now install a bulletin board themselves, without involving the developers, by clicking a button on the interface. Furthermore, the authentication and account management of the bulletin board was integrated with the rest of the prototype, eliminating the need for users to log in twice. However, the full features of phpBB were more than the users needed. They primarily made use of the basic post and reply functions and the threaded-conversation structure. Users indicated that the overall organization of the board system into topics, threads, and posts made sense to them. In the most recent major revision of the prototype, the phpBB system was replaced by a simpler, more integrated homemade bulletin board prototype that supported these basic features. Had the development progressed in the opposite order (i.e., building the simple prototype first, then adding features), it is possible that developers could have wasted valuable time and energy prototyping features that would only be discarded later for lack of use.

GENERALIZED APPROACH TO PATCHWORK PROTOTYPING

Based on the experiences described above, we have outlined a general approach to building patchwork prototypes using OSS. While our experience has been primarily with Web-based tools, and this process has been defined with such tools in mind, it is likely that a similar approach could be taken with prototyping any kind of software. Like other prototyping methods, this is designed to be iterated, with the knowledge and experience gained from one step feeding into the next. The approach entails the following five stages:

1. Make an educated guess about what the target system might look like.
2. Select tools that support some aspect of the desired functionality.
3. Integrate the tools into a rough composite
4. Deploy the prototype and solicit feedback from users.
5. Reflect on the experience of building the prototype and the feedback given by users, and repeat.

For the most part, these steps are relatively straightforward. Making the first educated guess about what the target system might look like can be the hardest step in this process because it requires the design team to synthesize their collective knowledge and understanding of the problem into a coherent design. In this first iteration of the process, it is often helpful to use paper prototypes and scenarios, but their function is primarily to serve as communications devices and brainstorming aids. The high equivocality of the situation almost guarantees, however, that whatever design they produce will be insufficient. This is not a failure. It is an expected part of the process, and the design will be improved on subsequent iterations. The important thing is to have a starting point that can be made concrete, and not to spend too much time brainstorming ideas. It is essential not to become bogged down in controversies about how the software "ought" to look, but rather to put together a prototype and test it out with users in their everyday environments and let the users figure out what works, what does not, and what is missing.

Selection and Integration of Tools: The Benefits of Using Open Source Software

There are several important considerations to keep in mind when selecting the tools. On first glance, patchwork prototyping as a method does not require OSS; the same general process could theoretically be followed by using software that provides APIs, or by creating prototypes through adapting methodologies for creating production-scale software systems such as COTS (commercial off-the-shelf) integration (Boehm & Abts, 1999). However, using OSS confers several important advantages; in fact, we believe that patchwork prototyping is only now emerging as a design practice because of the recent availability of a significant number of mature, production-scale OSS systems.

Without access to source code, developers are limited in how well they can patch together different modules, the features they can enable or disable, their ability to visually integrate the module with the rest of the system, and their ability to understand the underlying complexity of the code needed to construct such systems on a production scale. High-profile OSS is often of high quality, which means that difficult design decisions have already been made. Given that it is built from the collective experiences of many programmers, less effective designs have already been tried and discarded. In fact, by using and delving into human-readable (compared to that generated by CASE tools, e.g.), open source code, the developers can get a grounded understanding

of how particular features can be implemented, which can enable them to better estimate development time and costs.

The Web-based nature of patchwork prototypes affords several ways of integrating the selected software into the prototype, ranging from shallow to deep. Shallow integration consists of either wrapping the tools in an HTML frame to provide a consistent navigation menu between the tools, or customizing the HTML interfaces of the tools themselves to add hyperlinks. Most open source Web applications use HTML templates, cascading style sheets, and other interface customization features, which make adding or removing hyperlinks and changing the look and feel very easy. The advantage of shallow integration is the ease and speed with which the developer is able to cobble together a prototype. A significant drawback to shallow integration is that each application remains independent.

Deeper integration usually requires writing some code or modifying existing source code. This may include using components or modules written for the extension mechanisms designed into the application or other modifications made to the application's source code. If the developers cannot find precisely what they are looking for, they can fashion the code they need by copying and modifying similar extension code, or, in the worst case, the developers will need to write new code to facilitate the integration. However, the amount of code needed is very little in comparison to the amount of code that would have been required of the developers building a prototype from scratch.

For any prototyping effort to be worthwhile, the costs of creating the prototypes must be minimal. OSS systems tend to be fully implemented, stand-alone applications with many features and capabilities that provide a wealth of options to play with when prototyping to elicit requirements. The minimal effort required to add features allows the programmers to treat the features as disposable: Because little effort was needed to implement them, little effort is wasted when they are switched off or discarded. That most OSS are free is also important, both for budgetary reasons and because the developers can avoid complicated licensing negotiations. Additionally, most OSS have very active development communities behind them with members who are often eager to answer the developer's questions in considerable depth, and do so for free, unlike the expensive technical support that is available for commercial products. All of this facilitates the requirements-gathering process because iterations of the prototype can be rapidly created with high functionality at minimal cost, and with minimal effort and emotional investment by the developers.

Deployment, Reflection, and Iteration

During the deployment of the prototype, future users integrate the cyberinfrastructure into their work practices for an extended period of time and explore what they can do with it collaboratively. The collection of feedback on user experiences allows requirements gathering that is not purely need based, but also opportunity and creativity based. By seeing a high-fidelity prototype of the entire system, users can develop new ideas of how to utilize features that go beyond their intended use, and conceptualize new ways of accomplishing their work. In addition, users will become aware of gaps in functionality that need to be filled, and can explain them in a manner that is more concrete and accessible to the developers.

When reflecting on the collected feedback, however, the design team must realize that the prototype does not simply elicit technical requirements; it elicits requirements for the collaborative sociotechnical system as a whole. The existence of the prototype creates a technological infrastructure that influences the negotiation of the social practices being developed by the users via the activities the infrastructure affords

and constrains (Kling, 2000). The design team must be aware of how various features affect the development of social practice, and must make explicit the type of interactions that are required but are not currently realized. By allowing the users to interact with the prototypes for extended periods, collecting feedback on their experiences, and paying attention to the social consequences of the cyberinfrastructure, a richer understanding of the sociotechnical system as a whole can

Table 2. Comparison of patchwork prototyping with other methods

Paper Prototyping	Patchwork Prototyping	COTS/API Prototyping
Speed		
Can iterate a prototype multiple times in an afternoon	Can iterate a prototype in less than a week	Can take weeks or months to iterate a prototype
Monetary Costs		
Cost of office supplies	Free, or minimal cost of licenses if in business setting	Purchasing and licensing software can be expensive
Availability of Materials		
Usually already lying around	Large number of high-quality OSS available for free download	Not all commercial systems have APIs
Functionality		
Nonfunctional	High	High
Accessibility		
Anyone can prototype systems using paper, including nontechnical end users	Requires skilled programmers to create patchwork prototypes	Requires skilled programmers to integrate commercial software
Interface		
Not polished, but can provide a consistent and/or innovative interface concept for consideration	Not renowned for excellent usability; assembled components may be inconsistent	Individual elements may be high quality and familiar; assembled components may be inconsistent
Flexibility		
High: can do anything with paper	High: can modify source to create any desired functionality	Low: restricted to what the API allows, which may be limited
Disposability		
High: little investment of time, money, emotions	High: little investment of time, money, emotions	Low: significant effort and money can result in high emotional investment
User Attachment		
Low: users can see it is rough and nonfunctional	Med. to High: upon using it, can get attached to the system unless iterated rapidly	High: cannot be iterated fast enough to avoid attachment

emerge. Thus, reflection is a process of attending to the consequences of the design for the broader sociotechnical system, and integrating those consequences into a holistic understanding of how the system is evolving.

Iteration is essential to the rapid prototyping approach. First, iteration allows for the exploration of more features and alternatives. This can uncover overlooked aspects of the system that might be of use. This can also reinforce the importance or necessity of particular features or requirements. Furthermore, iteration provides the users with a constant flow of new design possibilities, which prevents them from becoming overly attached to any single design, giving them the freedom to criticize particular instances of the prototype. Ultimately, it is impossible to reach complete understanding of the system given its evolving nature. However, by iterating the prototyping process, the design space may narrow, identifying a set of key requirements. At this point the design is not complete, but work on a flexible production-scale system can begin, and further exploration of the design space can be continued within that system.

STRENGTHS AND LIMITATIONS

Patchwork prototyping addresses two major problems that designers face when building new sociotechnical systems. First, it allows the design team to get feedback on the prototype's use in real-world situations. Users interact with the system in their daily activities, which focuses their feedback around task-related problems. In Project Alpha, when members of the design team started using the prototype, the feedback changed from general praise or criticism of the appearance of the interface to more detailed explanations of how particular functionality aided or inhibited task performance. Second, it reduces the equivocality of the design space. By creating a functional prototype, discussions change from being highly suppositional to being about concrete actions, or concrete functionality.

Integration into the real-world context is markedly different from other prototyping and requirements-capture methods. Paper prototypes are typically given to users in a laboratory setting (Nielsen, 1993), thus all the tasks are artificial. While this can give developers important design insights, the drawback is that prototypes can end up optimized for artificial tasks and not for real-world use. More expensive methods such as participatory design (Ehn & Kyng, 1991) and ethnography (Crabtree, Nichols, O'Brien, Rouncefield, & Twidale, 2000) try to incorporate real-world use into the design process, the former by bringing users into the design team, the latter by observing users in their natural work environment. However, when the technology that these methods were used to design is introduced, it inevitably changes the practices and social structures present in the work environment, often in a way that cannot be predicted. Patchwork prototyping overcomes these limitations by being cheap and by providing real-time feedback on both users' problems with the software and the effects the software is having on the broader work context.

The advantages of patchwork prototyping can be seen when comparing it to other prototyping techniques. In Table 2 we compare it to paper prototyping and to prototyping using COTS software. The advantages of patchwork prototyping are that it has many of the benefits of paper prototyping, including low cost and ready availability of materials, yet provides the high functionality of COTS/API prototyping; the effort needed to create the prototypes and the length of the iteration cycles lies somewhere in between. Thus, while we see the method as being yet another tool for developers and designers to have in their toolbox, in many ways, it combines the best of both worlds.

The patchwork prototyping approach is not without limitations, however. Despite our hope that the visibility of the seams between the ap-

plications would be interpreted by the users as an indication that the prototype is a work in progress, our experiences seem to indicate that the users still view it as a finished product due to the fact that it has real functionality. It is possible that such interpretations can be overcome through social means by emphasizing the fact that the system is a prototype to all users who are encouraged to test it. However, since none of the projects we participated in did this, we have no idea whether or not that would be sufficient. One thing that is clear, however, is that visual coherence between applications greatly facilitates the ease of use and positive perceptions of the system as a whole. In fact, in Project Alpha, it was realized that users need different views of the component modules and features depending on the context in which they access the applications, and in some of those views the distinctions between modules must be totally erased.

Patchwork prototyping requires highly skilled programmers to be implemented effectively. Programmers must have significant experience within the development environment in which the OSS applications are coded; otherwise, they will spend too much time reading code and learning the environment, and the speed of implementation will not be as fast. Also, OSS can have security vulnerabilities that can compromise the server on which they are hosted. Project Beta ran into this problem when multiple installations of phpBB succumbed to an Internet worm, bringing down the prototype for several days. Third, patchwork prototyping requires a long-term commitment by users, and a motivated facilitator who is able to convince the users to adopt the prototype and incorporate it into their work practices. The facilitator must collect feedback about the users' experiences. Without willing users and the collection of feedback, the prototyping process will likely fail.

FUTURE TRENDS

The use of patchwork prototyping is still in its infancy. The relative ease with which patchwork prototypes can be constructed means that the method itself affords appropriation into new contexts of use. For example, one of the biggest costs to organizations is buying software systems such as enterprise management systems. Patchwork prototyping offers a cheap and effective method for exploring a design space and evaluating features. Consequently, through prototyping, managers can be more informed when shopping for software vendors and can more effectively evaluate how effective a particular vendor's solution will be for their company (Boehm & Abts, 1999).

Because users have to integrate the prototype into their daily work practices, transitioning from the patchwork prototype to the production-scale system can be highly disruptive. One method of avoiding this is having a gradual transition from the prototype to the production-scale system by replacing prototype modules with production-scale modules. To do this, however, the prototypes must be built on a robust, extensible, modular framework because the latter component is not easily replaced. If this model is used, the system development process need never end. Prototypes of new features can constantly be introduced as new modules, and, as they mature, be transitioned into production-scale systems. As more developers and organizations support open source development, the number and availability of OSS applications will increase. As more modules are written for particular open source, component-based systems, the costs of doing patchwork prototyping will further decrease, as will the threshold for programming ability—perhaps to the point where users could prototype systems for themselves that embody specifications for software programmers to implement.

CONCLUSION

Patchwork prototyping is a rapid prototyping approach to requirements gathering that shares the advantages of speed and low cost with paper prototypes, breadth of scope with horizontal prototypes, and depth and high functionality with vertical, high-fidelity prototypes. This makes it particularly useful for requirements gathering in highly equivocal situations such as designing cyberinfrastructure where there is no existing practice to support because it allows future users to integrate the cyberinfrastructure into their work practices for an extended period of time and explore what they can do with it collaboratively. It has the benefit of allowing the design team to monitor the sociotechnical effects of the prototype as it is happening, and gives users the ability to provide detailed, concrete, task-relevant feedback.

Patchwork prototyping is an excellent example of how OSS can foster innovation. The affordances of open-source code and a devoted development team create opportunities to utilize OSS in ways that go beyond the functionality of any particular application's design. The cases presented here merely scratch the surface of a new paradigm of OSS use. Further research is needed to understand the specific features of technologies that afford such innovative integration.

REFERENCES

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): An empirical review. *European Journal of Information Systems, 8*(3), 211-223.

Bishop, A. P., Bruce, B. C., Lunsford, K. J., Jones, M. C., Nazarova, M., Linderman, D., et al. (2004). Supporting community inquiry with digital resources. *Journal of Digital Information, 5*(3). Retrieved from http://joko.tanu.edu/Articles/v05/i03/Bishop

Boehm, B. W., & Abts, C. (1999). COTS integration: Plug and pray? *IEEE Computer, 32*(1), 135-138.

Brooks, F. P. (1995). *The mythical man-mouth: Essays on software engineering* (Anniversary ed.). Boston: Addison-Wesley.

Crabtree, A., Nichols, D. M., O'Brien, J., Rouncefield, M., & Twidale, M. B. (2000). Ethnomethodologically-informed ethnography and information systems design. *JASIS, 51*(7), 666-682.

Daft, R. L., & Lengel, R. H. (1986). Organizational information requirements, media richness and structural design. *Management Science, 32*(5), 554-571.

Daft, R. L., & Macintosh, N. B. (1981). A tentative exploration into the amount and equivocality of information processing in organizational work units. *Administrative Sciences Quarterly, 26*(2), 207-224.

Ehn, P., & Kyng, M. (1991). Cardboard computers: Mocking-it-up or hands-on the future. In J. Greenbaum & M. Kyng (Eds.), *Design at work* (pp. 169-196). Hillsdale, NJ: Laurence Erlbaum Associates.

Finholt, T. A. (2002). Collaboratories. *Annual Review of Information Science and Technology, 36*(1), 73-107.

Floyd, C. (1984). A systematic look at prototyping. In R. Budde, K. Kuhlenkamp, L. Mathiassen, & H. Zullighoven (Eds.), *Approaches to prototyping* (pp. 1-18). Berlin, Germany: Springer-Verlag.

Grudin, J. (1988). Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *CSCW 88: Proceedings of the Conference on Computer-Supported Cooperative Work* (pp. 85-93).

Kling, R. (2000). Learning about information technologies and social change: The contribution

of social informatics. *The Information Society, 16*, 217-232.

Lakhani, K. R., & von Hippel, E. (2002). How open source software works: "Free" user-to-user assistance. *Research Policy, 1451*, 1-21.

Lim, K. H., & Benbasat, I. (2000). The effect of multimedia on perceived equivocality and perceived usefulness of information systems. *MIS Quarterly, 24*(3), 449-471.

Martin, J. (1991). *Rapid application development.* New York: Macmillan Publishing Co.

Nielsen, J. (1993). *Usability engineering.* San Diego, CA: Morgan Kaufman.

Pressman, R. S., Lewis, T., Adida, B., Ullman, E., DeMarco, T., Gilb, T., et al. (1998). Can Internet-based applications be engineered? *IEEE Software, 15*(5), 104-110.

Rettig, M. (1994). Prototyping for tiny fingers. *Communications of the ACM, 37*(4), 21-27.

Rudd, J., Stern, K., & Isensee, S. (1996). Low vs. high-fidelity prototyping debate. *Interactions, 3*(1), 76-85.

Star, S. L., & Ruhleder, K. (1996). Steps toward an ecology of infrastructure: Design and access for large information spaces. *Information Systems Research, 7*(1), 111-134.

Thomke, S., & von Hippel, E. (2002). Customers as innovators: New ways to create value. *Harvard Business Review, 80*(4), 74-81.

Trist, E. L. (1981). The sociotechnical perspective: The evolution of sociotechnical systems as a conceptual framework and as an action research program. In A. H. van de Ven & W. F. Joyce (Eds.), *Perspectives on organization design and behavior* (pp. 19-75). New York: John Wiley & Sons.

KEY TERMS

COTS Integration: The process by which most businesses integrate commercial off-the-shelf software systems in order to create a computing environment to support their business activities.

Equivocality: The name for a lack of knowledge that cannot be mitigated simply by doing research or gathering more information. In an equivocal situation, decisions often need to be made, definitions created, and procedures negotiated by various (often competing) stakeholders.

Paper Prototyping: A rapid prototyping method for creating low-fidelity prototypes using pencils, paper, sticky notes, and other low-tech materials that can be quickly iterated in order to explore a design space. It is often used in interface design.

Patchwork Prototyping: A rapid prototyping method for creating high-fidelity prototypes out of open source software that can be integrated by users into their everyday activities. This gives users something concrete to play with and facilitates a collaborative process of sociotechnical systems development. It is ideal for highly equivocal design situations.

Rapid Prototyping: Rapid prototyping is a method that involves creating a series of prototypes in rapid, iterative cycles. Normally, a prototype is created quickly, presented to users in order to obtain feedback on the design, and then a new prototype is created that incorporates that feedback. This cycle is continued until a fairly stable, satisfactory design emerges, which informs the design of a production-scale system.

Sociotechnical System: Refers to the concept that one cannot understand how a technology will be used in a particular environment without un-

derstanding the social aspects of the environment, and that one cannot understand the social aspects of the environment without understanding how the technology being used shapes and constrains social interaction. Thus, one can only understand what is going on in an environment by looking at it through a holistic lens of analysis.

Uncertainty: The name for a lack of knowledge that can be addressed by obtaining more information, such as by researching an answer, looking it up in reference materials, or collecting data.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant & B. Still, pp. 126-140, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 4.17
Evaluation of a Migration to Open Source Software

Bruno Rossi
Free University of Bozen-Bolzano, Italy

Barbara Russo
Free University of Bozen-Bolzano, Italy

Giancarlo Succi
Free University of Bozen-Bolzano, Italy

ABSTRACT

The chapter discusses the adoption and assimilation process of open source software as a new form of information technology. Specifically, the case reports a general positive attitude towards the widely used technology, the OpenOffice.org suite for office automation. Nevertheless, it shows the difficulties of the first early adopters to lead the innovation process and push other users. Different usage patterns, interoperability issues, and, in general, the reduction in personal productivity typical of the early phases of adoption are also remarked. The aim of this chapter is to give the reader an overview of the adoption process by means of the analysis of quantitative and qualitative data gathered during real world experimentation, and to shed some light on how empirical data can corroborate or challenge the existing literature about open source software and technology adoption.

INTRODUCTION

Open source software (OSS) and open data standards (ODS) have emerged in recent years as a viable alternative to proprietary solutions. There are many cases in which the adoption of OSS has proved advantageous for companies deciding to adopt it in replacement or in conjunction with closed source software (CSS). Unfortunately, at our knowledge, these studies often report only about server-side migrations or give very little empirical evidence of the benefits of the new solution. Among case studies that report success-

ful transitions on the desktop side we can surely mention as pioneers the Extremadura, Munich, and Vienna case studies (Marson, 2005; Landeshauptstadt München, 2003; Stadt Wien, 2004). All these cases have in common the intention of a large migration inside a single public administration (PA). Furthermore, the migration to OSS in all these cases has been already performed or is in the process of being deployed. We summarise the most famous deployments in Table 1, three are European, while one is U.S.-based.

One of the most remarkable deployments of OSS on the desktop side is surely the one of the Extremadura region in Spain, recently installing 80,000 Linux systems, 66,000 for the educational system and 14,000 for administrative workstations. The local administration even created their Linux distribution called gnuLinex[1]. According to their IT department, the savings have been of the order of €18M (Marson, 2005). Another case of success is the one of the city of Largo, FL (USA) where the migration has involved 900 clients; the savings have been estimated in $300,000-$400,000 (Miller, 2002). The migration of the city of Munich and the one of the city of Vienna are currently underway (Landeshauptstadt München, 2003; Stadt Wien, 2004). As the delay of the Munich migration seems to demonstrate, a transition to OSS is not a process to underestimate. There are also cases where the proprietary solution has been considered more convenient, like the city of Nürnberg, where according to their own migration study, the transition from Windows 2000/Office 2000 to Windows XP/Office XP was considered as €4.5M cheaper than the transition to Linux/OpenOffice.org (Stadt Nürnberg, 2004).

Another case of interest that emerged recently is the decision of the state of Massachusetts to abandon closed data standards (CDS) in favour of ODS, in particular to adopt the open document format for office automation documents exchange activities starting from January 2007 (Massachusetts State, 2005). According to the Organization for the Advancement of Structured Information Standards (OASIS) the purpose of the format is "to create an open, XML-based file format specification for office applications" (OASIS, 2005). Following this case and the increasingly requests coming from the European Commission to reduce e-government barriers, Microsoft decided to open the formats supported by its office automation suite in the upcoming months (Palmer, 2005).

The goal of this chapter is to provide an insight on two different experimental migrations to OSS inside European PAs. In particular, we don't consider a full migration, but the introduction of OSS in the office automation field. Throughout

Table 1. Large deployments of OSS inside public administrations

Region	Clients to migrate	Side	Distribution
Extremadura	80000	Desktop/ Servers	gnuLinex
Munich	14000	Desktop	Debian
Vienna	7500	Desktop	Wienux (Debian/ KDE)
Largo, FL	900	Desktop/ Servers	Linux KDE 2.1.1

a constant monitoring of the software employed, we derive some indications on software usage that can be useful to provide more information on the migration process and the adoption of OSS.

In the next sections, we will provide first an overview of the existing literature about technology adoption and then start reviewing the experimentation details providing background information about the two Public Administrations involved. The last part will be devoted to the discussion of the results.

Technology Adoption and Assimilation

Before entering the discussion about the experimentation and the migration performed, an overview of the existing literature about technology adoption and assimilation will be useful. This will also provide a framework in which the results of the experimentation will be inserted.

Technology adoption, diffusion and acceptance research bases its foundation on the early work of Everitt Rogers, in the book titled *Diffusion of Innovations.* Rogers (1995) interest lies in studying the diffusion process that characterises technology adoption. In his seminal work, technology adopters are categorised according to the phase in which they make the adoption decision. The main distinction is among innovators, early adopters, early majority, late majority, and laggards. In particular, the author models the diffusion as an S-shaped curve characterised by an initial adoption speed and a later growth rate. The claim is that different technologies will lead to different adoption patterns.

Interesting for our study are various factors that affect the level of technology adoption inside organisations, like the *organisational age* (Chatterjee, Grewal, & Sambamurthy, 2002), *organisational size* (Fichman, & Kemerer, 1997), *industry type* (Chatterjee, Grewal, & Sambamurthy, 2002; Fichman, & Kemerer, 1997), and *sophistication of the IT infrastructure* (Armstrong, 1999; Chau & Tam, 1997). To some extent, the evidence seems to report that organisations that are younger, larger and belong to certain industry types are more willing to invest and adopt new technology. The existence of a sophisticated IT infrastructure will also lead to an easier adoption path.

Furthermore, Fichman and Kemerer (1999) report two critical factors that influence the technology assimilation process: knowledge barriers and increasing returns. The first effect relates to the effort necessary to acquire the necessary knowledge and skills to properly adopt a certain technology. This effect leads to what are known as knowledge barriers (Attewell, 1992; Fichman & Kemerer, 1999). Being a new and still somewhat unexplored field, we think that OSS is subject heavily to knowledge acquisition barriers that can in some way hinder its adoption.

As a second macro-level phenomenon, the adoption of certain technologies is subject not only to supply-side benefits due to economies of scale (Shapiro & Varian, 1999) but also to a demand-side effect called increasing returns effect (Arthur, 1989). The effect leads to an increase of utility in adoption for each successive adopter, based on the number of previous adopters. Arthur (1989) goes further in this analysis, claiming that "[e]conomy, over time, can become locked-in by 'random' historical events to a technological path that is not necessarily efficient, not possible to predict from usual knowledge of supply and demand functions, and not easy to change by standard tax or subsidy policies" (p. 2). In this sense, it may not be possible to easily switch from a certain technology once a certain critical level of adoption has been reached.

Open source software and software in general is one of the goods that are particularly sensible to economies of scale, increasing returns and knowledge barriers. To understand fully the adoption process, all these effects have to be considered.

BACKGROUND INFORMATION

Experimentation on the migration to OSS in the office automation field has been performed in two different European public administrations (PAs). We will discuss briefly the background details of the two public administrations involved and for simplification purposes, we will refer to the first public administration as PA1 and the second as PA2.

PA1 is a large public administration, counting globally over 5,000 employees. The budget allocated for the ICT (information and communication technology) services is high, but the experience with OSS is still limited. The reason for the interest in a possible migration to OpenOffice.org and in general to other OS applications is threefold:

- Spare the money spent yearly to cover the license costs.
- Reduce the effort needed to handle the licenses.
- Provide a benefit to the local economy, by means of the adoption of OSS.

PA2 is composed by a large number of municipalities spread across the territory of its region. Nearly all the municipalities in the consortium are small and count on the average 50 desktop machines. The maintenance is performed remotely by the central IT (information technology) department. In this case, the budget available for ICT services in such small municipalities is low, but a great experience in OSS has been built in recent years, mostly based on server-side solutions. The objectives of a possible migration to OSS are the following:

- Reduce the costs of ICT services in the long term.
- Ensure the accessibility of generated documents also in the future, not relying on proprietary data standards.

To summarise the characteristics of the the PAs that took part to the experimentation, both share a similar organisational size, while differences exist in prior OSS experience and budget allocated for ICT services.

FOCUS OF CHAPTER

Experiment Design

The experimentation performed has involved the market leader Microsoft Office[2] and OpenOffice.org[3], an OSS suite offering ODS support. The decision to use these applications has been done in accordance with the relevance of office automation inside PAs (Drakos, Di Maio, & Simpson, 2003) and the guidelines given by IDABC (Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens) for a gradual transition to OSS. One of the main suggestions is to "introduce applications in a familiar environment" (IDBAC Report, 2003, p. 23). The introduction of OpenOffice.org is seen as a necessary step for a successive complete migration to OSS.

To monitor the behaviour of users with both solutions, we adopted the PRO Metrics (PROM) software as a mean to collect and analyse software metrics and personal software process data (Sillitti, Janes, Succi, & Vernazza, 2003), software that permits to collect metrics on software usage in a non-invasive manner. It allows the collection of the measures of time spent on documents, name of the document and other useful information about the general software usage. To protect the privacy of the users several measures were taken in accordance with the local union representatives:

- Data collected has been encrypted by means of the strong AES algorithm (Pfleeger & Pfleeger, 2002).

- Usernames were randomly generated.
- Data of single users were not given to single PAs, the analysis presented has been only given in aggregated form and with the aim to provide an evaluation of the migration.

In Table 2, a comparison of both experimentations is performed.

The number of users involved in the experimentation has been equivalent, both PAs decided to install OpenOffice.org in order to evaluate the possible future migration. The suite has been installed on a large number of workstations in both PAs; however our study has been performed on a smaller subset of users. The total events that are reported in table refer to the smallest unit that the data collection software details; a single event refers to the application's window release of focus.

This number details the amount of data that have been collected during the experimentation. The maturity row refers to the situation in which the experimentation has been performed; PA2 was already in a more advanced state of technology adoption, offering the open solution for several months prior to the experimentation. As a last annotation, the details of username generation have been slightly different between the two installations, in PA2 the usernames have been generated on a per machine basis, different users working on a single workstation are mapped as a single entity. This will result in higher documents per day or time per day per single username compared to PA1, where the usernames map directly to a single user. Nevertheless, the results have not been influenced by this approach, since the common practice in PA2 is to have a single workstation per user.

The experimentation protocol followed the same schema in both experimentations:

- Installation of OpenOffice.org; the version of the suite installed is OpenOffice.org 1.1.3 in both PAs; various versions of the Microsoft Office suite were available on the target systems
- Installation of the PROM agent to monitor the software adoption level
- Training on the OpenOffice.org suite, mostly performed to show how to perform the same task in the new office automation environment
- A questionnaire on the attitude towards Open Source Software submitted to users
- Support provided to users by means of forums and hot-lines

Table 2. Comparison of both experimentations

	PA1	**PA2**
Users experimenting	1486	1475
Total OOo installations	~4000	~2000
Total MSO installations	~4000	~2000
Days	30	40
Total events	1518150	1435553
Maturity	Starting Ooo introduction	Already using Ooo
ID generation	Per user	Per machine

Methodology and Limitations

The methodology applied is mainly empirical; the analysis is based on quantitative data collected through a non-invasive software agent and on qualitative data collected by means of questionnaires. A full controlled experiment could not be performed, as it would not have been possible to control all exogenous factors that could affect the final results (Campbell & Stanley, 1990). For a controlled experiment, but on a more limited number of users during a migration to OSS (see Rossi, Scotto, Sillitti, & Succi, 2005). A comparison of the functionalities of Sun StarOffice Writer and Microsoft Office Word[4] can be found in (Everitt & Lederer, 2001). Also in this case the comparison is on a limited number of users, focusing on the functionalities offered by both solutions and how users could perform the same task. Researchers found that "[w]hile overall ratings for both products were comparable, participants were more comfortable and satisfied with Microsoft Word and found it easier to use than StarOffice Writer" (Everitt & Lederer, 2001, p. 2).

In the following sections we will perform first a comparison of the initial attitude of users towards OSS, the comparison of the two solutions by means of the quantitative data collected and in the end evaluate possible interoperability issues that can raise in case of a full migration.

Initial Attitudes Toward OSS

The experimentation has been supported by qualitative data coming from one questionnaire submitted to users; the aim of the questionnaire was to evaluate the attitude of the users towards OSS, as it can have a great impact on the successive acceptance of OSS. The questionnaire has been submitted in electronic format. We report here the results that may be interesting to evaluate the attitude of users before entering the experimentation. Data in this section refers to 282 users of PA1.

The first two questions related to the knowledge of OSS, in particular the familiarity with the concept and the general users' perception. The answers are represented in Figure 1.

Surprisingly, more than 60% of the users that filled the questionnaire depict themselves either as very familiar or fairly familiar with the concept. One of the reasons can be that users with more attitude towards OSS were the ones that filled the questionnaire earlier. The second question about the perception of OSS leads to a group of users neutral or positive towards the new concept; after the experimentation it is possible that users acquire a more sharp view on the subject; in this sense, we should expect at that point, neutral users to represent the minority.

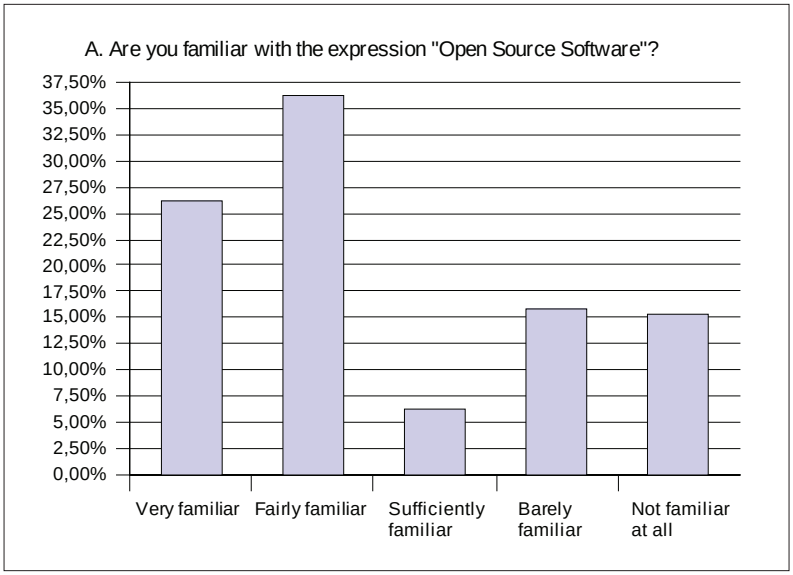
The third question in Figure 2 further investigates the knowledge of users in the field, we asked whether users know OS products and whether they can name at least one.

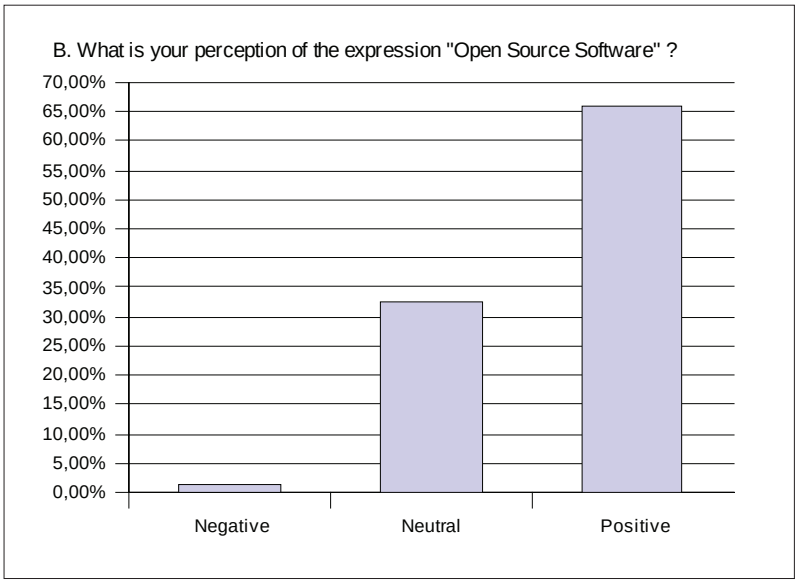
Not surprisingly, the majority of users report no application. The most known products are OpenOffice.org and the Linux operating system.

In showing the results of the remaining questions, we divided the users in two categories, users that had already an opinion on OSS and users that don't know the phenomenon. In this sense the first category consisted of all users considering themselves either as familiar or very familiar with OSS (see Figure 1, Question A) and naming at least one application (see Figure 2, Question C). For the reader's convenience, in the upcoming tables we named these groups OSS and non-OSS users. In Figure 3, the experimenters are questioned about the purchasing criteria of software in general, without particular reference to OSS.

In this case, users already with knowledge of OSS seem to be more aware of the customisation requirements of software inside PAs. The next two questions are related to a full migration to OSS. In Figure 4 users are posed in a situation of a generalized introduction of OSS and its effects on the organizational aspects of the PA.

Figure 1. PA1—Results of Question A





The results of Figure 4 are comparable across both groups: the majority of users consider the introduction of OSS as a chance of reorganization of the IT department of the PA. Furthermore, 15% of users consider the introduction as non important in terms of organizational impact. The last question in Figure 5 is very similar to the previous one, but this time is related to the impact of the migration on the single user.

Figure 2. PA1—Results of Question C and Question D

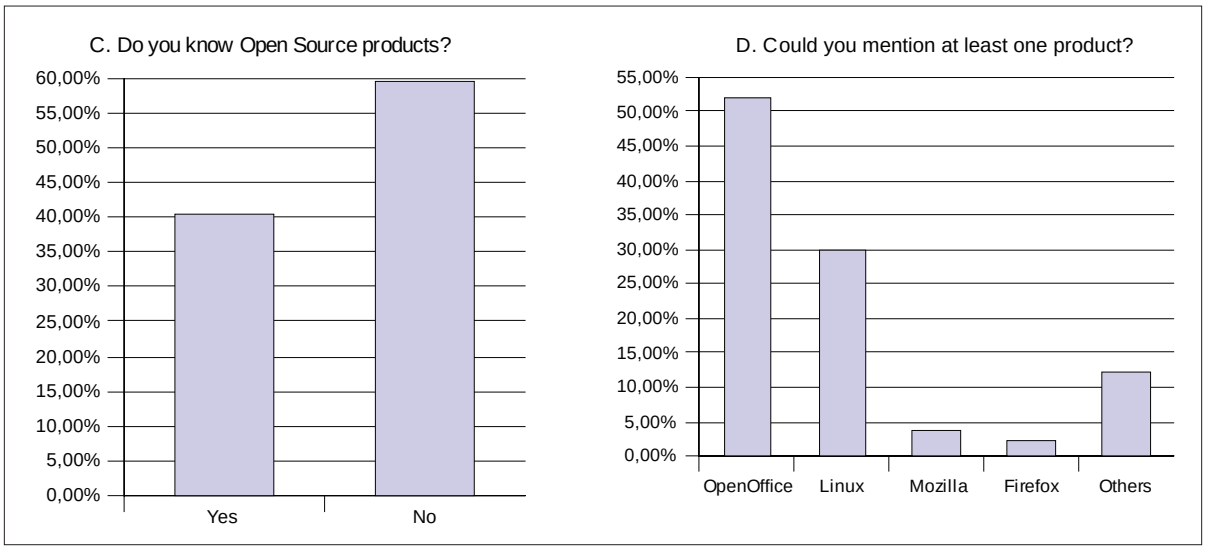
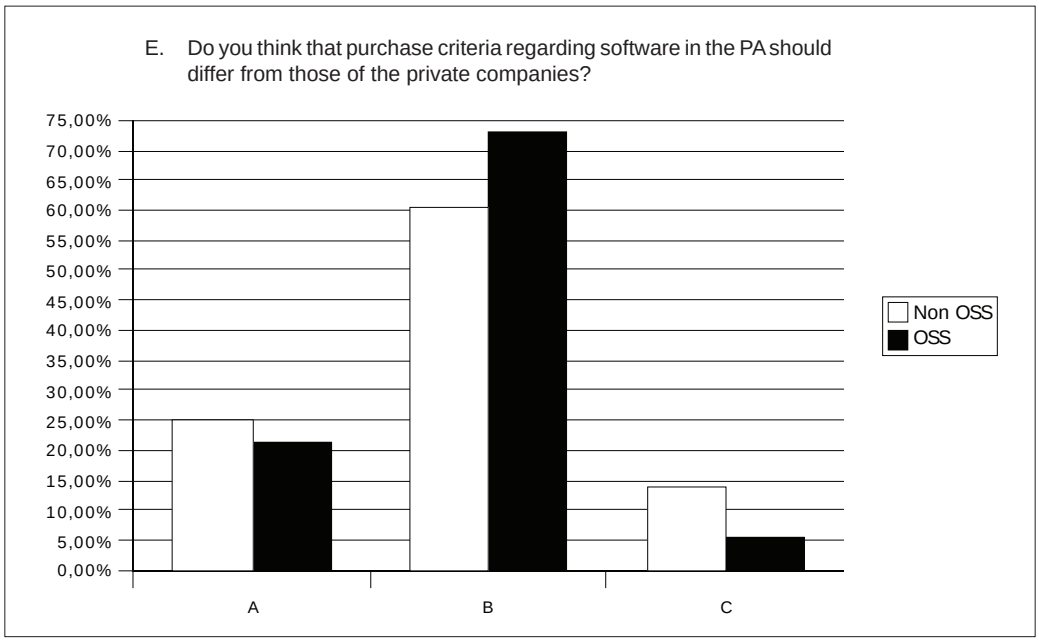
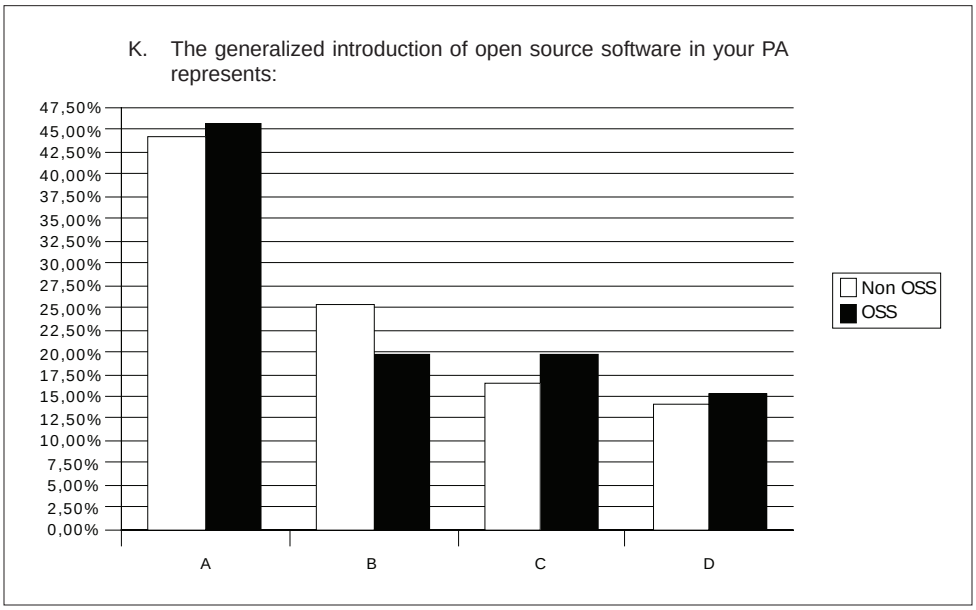


Figure 3. PA1—Results of Question E



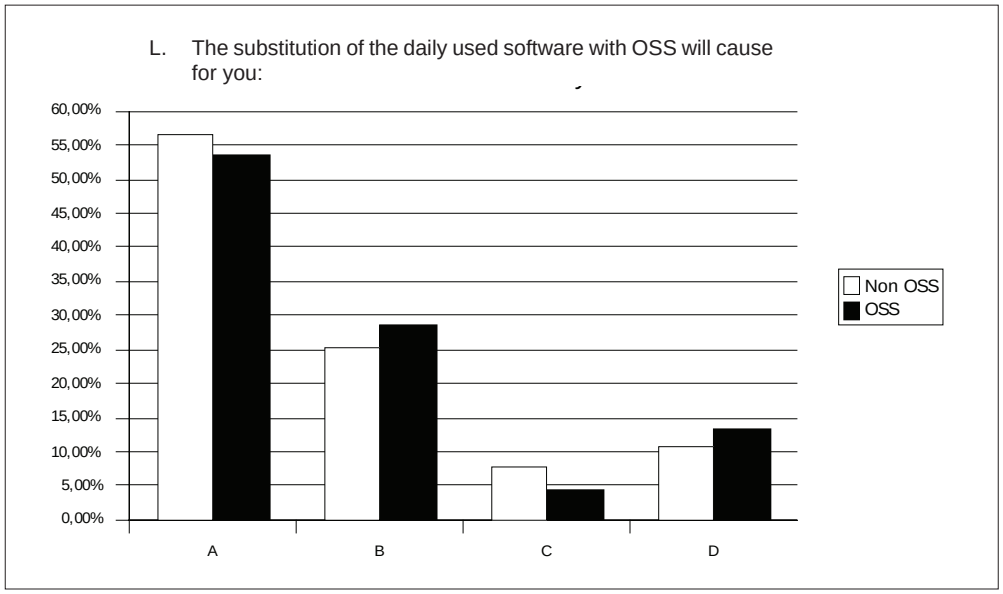
Note: Possible answers are (a) No, they should follow the same criteria; (b) Yes, they should take into account the peculiar needs of the PA; (c) I don't know.

Figure 4. PA1—Results of Question K



Note: Possible answers are (a) A chance for the reorganization of the IT structure; (b)A chance for the redefinition of the organizational structure in a wide perspective; (c) A further load of work for the single units; (d) The introduction will not be important.

Figure 5. PA1—Results of Question L



Note: Possible answers: (a) More work in the short run, but advantages in the long run; (b) More work in the short run and no advantages in the long run; (c) Less work; (d) More work.

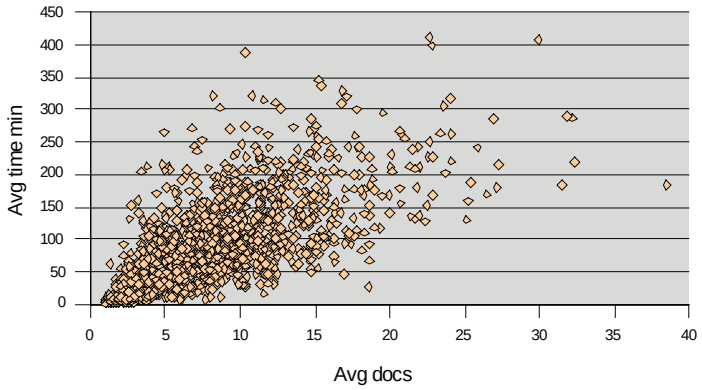
The results also in this case do not report a large difference between the two groups, more than half of the users are convinced that the substitution will have a negative impact on the workload in the short period, the advantages will be evident only in the long period. Users of the OSS group seem more conscious about the effort that a migration causes.

Overall the results of the questionnaires report users in general positive towards OSS. It would be interesting as an additional study to evaluate the impact of the experimentation on the users' attitude, to see how the perception of users changes after the influence of a full migration.

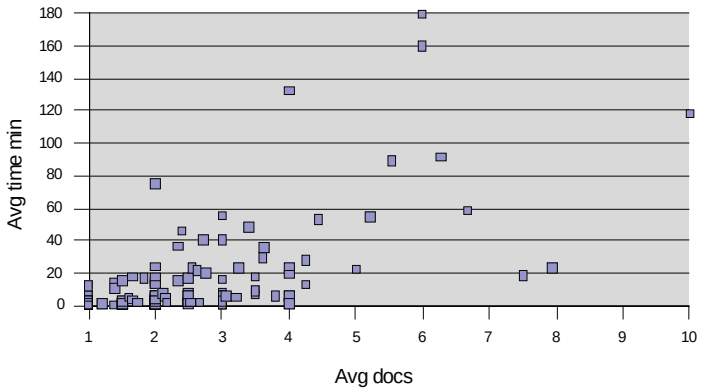
Comparison of the Solutions

Both softwares for office automation have been running during the whole experimentation, users were free to choose the solution more appropri-

Figure 6. PA1—Distribution of users across average documents (x-axis) and average time (y-axis)



(a) Microsoft Office documents handling



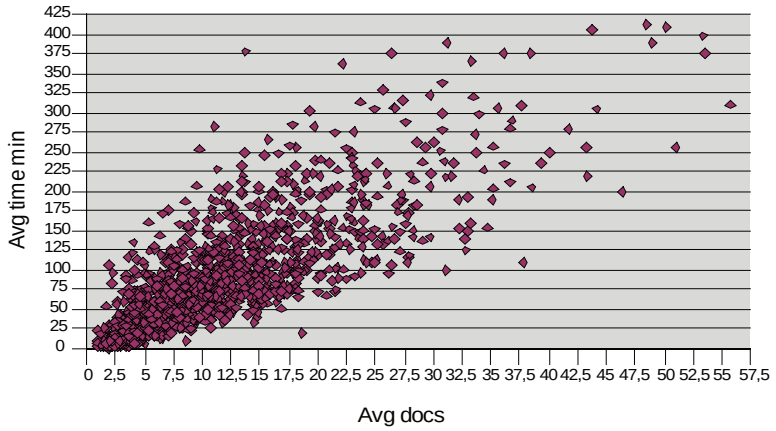
(b) OpenOffice.org documents

ate for the task to perform. A limitation on this decision was given by the large number of files already available in the original data format. More of these interoperability details will be analyzed in the apposite section.

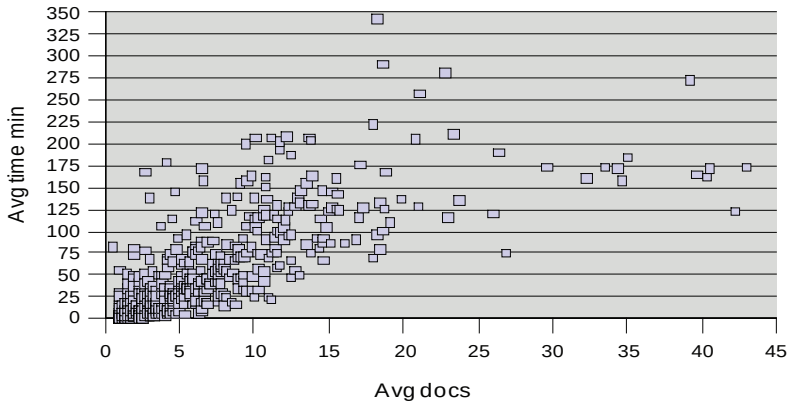
From the analysis performed, we observed that the average time spent with the new solution tends to be minimal in PA1, where the software has been introduced with the experimentation. In PA2 instead, where the solution has been installed for several months, daily average minutes per day tend to be above 50 minutes per user.

The events generated have been aggregated in two different kinds of measures, the average number of documents worked per day by each user during the whole period and the average time in minutes spent on the documents. In Figures 6 and 7 we see the mapping of each user for PA1 and workstation for PA2 in this space. In each figure on the left the mapping is for Microsoft Office,

Figure 7. PA2—Distribution of users across average documents (x-axis) and average time (y-axis)



(a) Microsoft Office documents handling



(b) OpenOffice.org documents

while on the right the mapping is for OpenOffice.org. Each point represents a user.

In PA1, 90% of all users lie in the space between 20 documents per day and 200 minutes per day spent using Microsoft Office. In PA2 90% of all clients lie between 24 documents and 240 minutes.

Regarding OpenOffice.org, 90% of PA1's users lie between four documents and less than 60 minutes of usage. In PA2, as we should expect, the usage is stabilized at higher levels, with the limit of eight documents and 135 minutes that encompasses 90% of the users.

From the temporal evolution of the software usage, we note that the software usage is constant in both PAs during the whole experimentation. This is due to the short time frame we are analysing. At this stage of the experimentation, the difference in usage of OpenOffice.org between the two PAs is clearly evident.

Furthermore, the distribution of users across time and documents gives a better idea of the grouping of users. As a further step, the application of clustering techniques in order to group users according to further variables and in accordance to their attitude, might also shed some lights on the usage pattern of the different applications (Duda, Hart, & Stork, 2001).

Functionalities

A further study on the functionalities[5] has been performed in both PAs, the goal was to gather information on how users evaluate the office automation application's features. Indeed, one of the major critics to OSS on the desktop-side, is its supposed lack of usability compared to CSS (Nichols & Twidale, 2003). The aim of this section is to evaluate the difference in functionalities usage between the two applications and whether from this distinction we can derive some indications about the usability.

In Figure 8, a first representation of the situation in PA1 is plotted. Users are mapped according to the average Microsoft Office functions per day (x-axis) and average OpenOffice.org functions per day (y-axis).

From the distribution of users in Figure 8, we can notice that users tend to use daily more functions in Microsoft Office. To further investigate this issue, we then compared both situations normalizing the functions used per time unit. To perform this operation, we set-up the following metric:

$$\frac{1}{n} \cdot \sum \left[\frac{\sum f}{t} \right]$$

Figure 8. PA1—Functions per office automation software

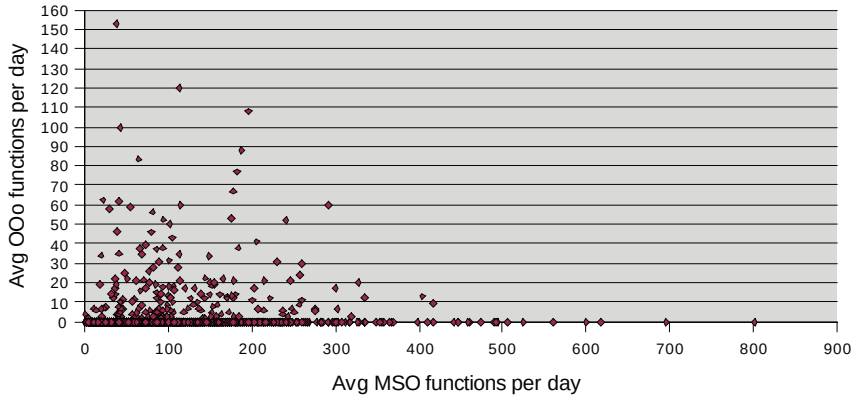


Table 3. PA1—Results of functions calling between Microsoft Office and OpenOffice.org

	MSO	OOo
Min	0,03	0.08
Max	5,45	5,45
Mean	0,41	1,71
Std. Dev	0,45	1,26

where f is a single function, t is the time spent on documents and n is the total number of users. By using the time we can compare the results among the two solutions. As a result of this formula, we get the distribution results shown in Table 3.

The number of normalised functionalities used is in general lower with Microsoft Office than with OpenOffice.org; an explanation can be the fact that users are more acquainted to shortcuts in order to perform certain operations. On the other side newcomers to OpenOffice.org have yet to acquire the necessary confidence in the functionalities offered.

These considerations cannot alone denote a possible usability problem of OpenOffice.org. However, they can indicate the difference in usage of the new technology introduced, a difference that will obviously reflect on users' productivity during the early phases of a migration.

FUTURE TRENDS

Interoperability Considerations

One of the strategies that a software vendor entering a market can exploit to emerge in a situation where users are in a situation of lock-in, is to provide higher compatibility with the standards already offered on the market. This strategy has its drawback in the fact that some performance of the application has to be sacrificed in favour of the compatibility, entering a mechanism of trade-off (Shapiro & Varian, 1999).

In this sense, OpenOffice.org offers compatibility also with the closed data standards of the Microsoft Office suite. It is interesting in the study proposed to see how users adopted this compatibility feature. To gain a measure of this interoperability issue, we computed as a first step the number of Microsoft Office proprietary formats documents opened by means of OpenOffice.org. Further data, as the time spent with the documents opened with this method and the number of users adopting the feature also add detail to this analysis.

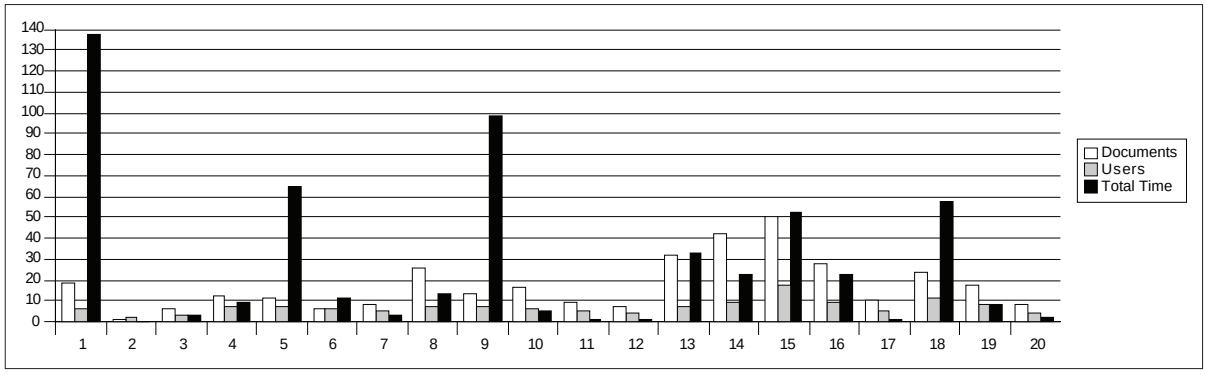
In Figure 9, data are reported for each day: the number of foreign documents opened per day (in white), the total time in minutes spent on the documents opened (in black) and the number of users adopting this solution at least once per day (in grey). The Microsoft Office formats considered are the ones handled by Microsoft Word, Excel and Powerpoint, namely files with doc, rtf, xls, and ppt extensions.

The results of this kind of analysis are not encouraging; probably one of the reasons is that users are not aware of this possibility. A minimal number of users is adopting this feature in his everyday work, to be precise only 10.90% of OpenOffice.org users (17 out of 156) with 6.76% of the global time spent in OpenOffice.org (nearly nine hours out of 138 hours). This last aspect seems to justify also that users tend to open documents of the foreign format for viewing purposes only; editing is seen as dangerous due to the different application used.

The same analysis is represented in Figure 10 also for the second PA. In this case we see that users more trained and adopting OpenOffice.org for a longer time have a clearer idea of the functionalities offered.

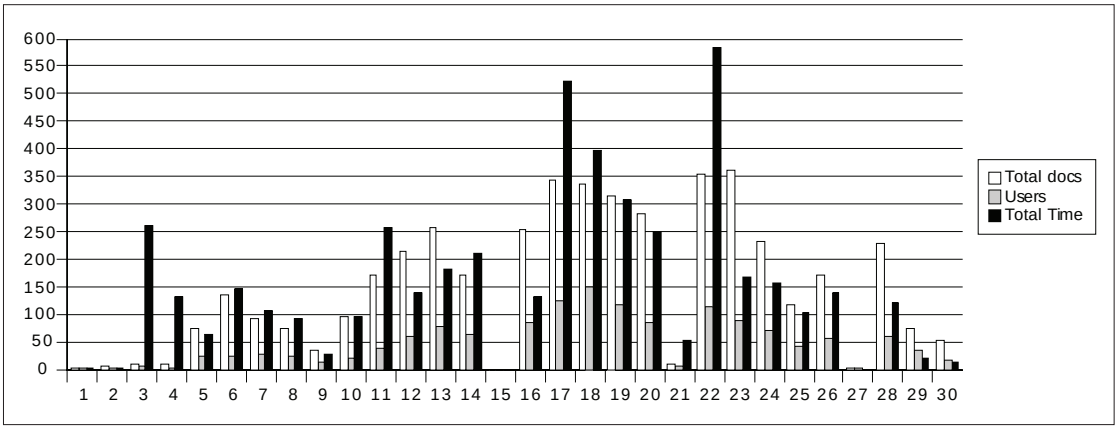
The same considerations of the previous group report that 57.46% of OpenOffice.org users (447 out

Figure 9. PA—Representation of the Microsoft Office documents opened by using OpenOffice.org



Note: For each day the figure reports number of documents (in white), users adopting this feature at least once in that day (grey) and the total time for the day spent in minutes on the documents after the opening (black). Extensions considered are .doc and .rtf (Microsoft Word), .ppt (Microsoft Powerpoint) and .xls (Microsoft Excel).

Figure 10. PA2—Representation of the Microsoft Office documents opened by using OpenOffice.org

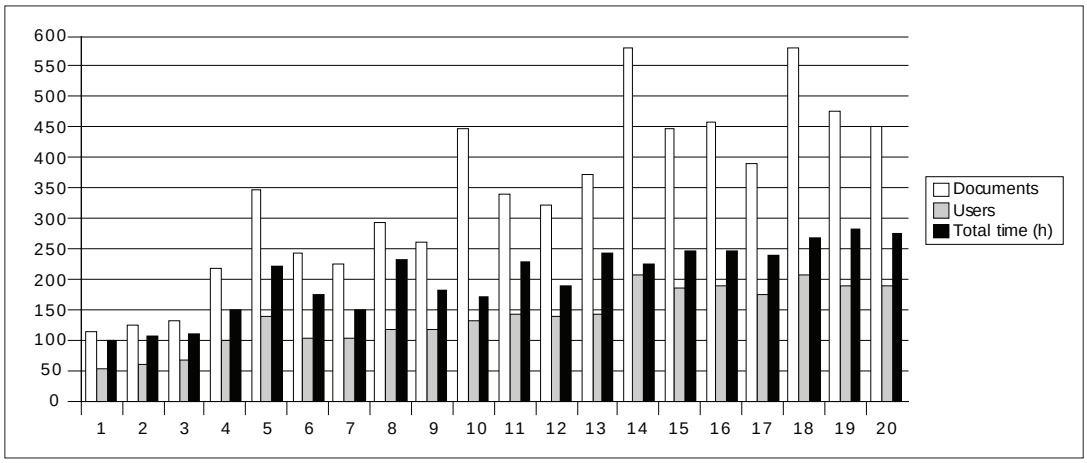


Note: For each day the figure reports number of documents (in white), users adopting this feature at least once in that day (in grey) and the total time for the day spent in minutes on the documents after the opening (in black). Extensions considered are .doc and .rtf (Microsoft Word), .ppt (Microsoft Powerpoint) and .xls (Microsoft Excel.)

of 778) used this feature, but only 2.26% of the time spent in OpenOffice.org (nearly 78 hour over 3.594 hours). In this case the result confirms that users are more aware of the interoperability features.

Another important interoperability issue in the migration in the office automation field is due to the different applications available in both suites. While Microsoft Office offers a small personal

Figure 11. PA1—Representation of the Microsoft Access documents handled by users



Note: For each day the figure reports number of documents (in white), users opening such a document at least once in that day (in grey) and the total time for the day spent in hours on the documents after the opening (in black).

database application called Access, OpenOffice.org in the version available to experimenters doesn't offer a comparable alternative[6]. In Figure 11 the use of Microsoft Access is reported, with number of documents (in white), number of users using the application in that particular day (in grey) and total time spent by all users in hours (in black). Time has been reported in hours to facilitate the reading.

What can be seen is that the software is still very used; nearly 30% of all Microsoft Office users used at least once the application during the experimentation period. If we consider only users employing it for a period greater than five days, the percentage drops to 15%. In this kind of analysis we cannot perform a comparison with PA2 as the software for data collection installed was not configured to collect this kind of information.

The results of this section report that a more focused training on the interoperability features offered by the OpenOffice.org suite can lead to a broader diffusion of the suite. It is still to understand the reasons of the lack of confidence in editing documents in the other application source format.

CONCLUSION

The results of both experimentations show that open source software (OSS) can represent a viable alternative to closed source software (CSS) even on the desktop side. The analysis was focused on four different levels of technology adoption, the level of the users' attitude towards OSS, level of adoption and usage of both solutions during the period, functionalities adopted and the interoperability issues. Where possible, all levels have been considered for both PAs that participated to the experimentation:

- The attitude was in general positive; users had a positive attitude before starting

the experimentation. However, we should expect a change of the attitude at the end of the experimentation. Neutral users will probably join the groups of enthusiastic or sceptics about OSS.

- The adoption and usage of both solutions has seen the predominance of the market-dominant Microsoft Office, although in the experimentation where OpenOffice.org was already introduced users started to use it in everyday work. This is due also to network effects in IT markets that have been exploited by the early adopter PA of our study (Katz & Shapiro, 1985). Implementing a strategy of documents exchange in the new format is a key decision to widen the diffusion of the new application. The results obtained show that the migration path will be more difficult in absence of a proper strategy of documents exchange.
- The analysis of the functionalities used has shown that there are different patterns between the groups of the two suites. The group of Microsoft Office users has more confidence in the software, performing their task mainly through shortcuts. Such a confidence is not present in OpenOffice.org users. The results cannot be used to evaluate the usability of OSS, however they do report the reduction in productivity that is typical of the early phases of software migration.
- The analysis on interoperability shows that there are still interoperability issues, mainly in the form of personal databases creation. Furthermore, users don't seem to evaluate positively the compatibility with the foreign format offered by OpenOffice.org. The strategy to increase the diffusion of the software by providing a greater level of compatibility with the existing data standards doesn't seem to provide the results expected.

Overall, the data collected have granted the possibility to evaluate the adoption levels of OSS inside two different PAs. In the cases reported, the initial levels of adoption are low and interoperability issues exist that can potentially hinder OSS adoption.

NOTE

This work has been partially supported by COSPA (Consortium for Open Source Software in the Public Administration), EU IST FP6 project nr. 2002-2164

REFERENCES

Armstrong, C. P., & Sambamurthy, V. (1999). Information technology assimilation in firms: The influence of senior leadership and IT infrastructure. *Information Systems Research, 10*(4), 304-327.

Arthur, W. B. (1989). Competing technologies, increasing returns, and lock-in by historical events. *Economic Journal, 99*, 116-131.

Attewell, P. (1992). Technology diffusion and organizational learning. *The Case of Business Computing. Organization Science, 3*(1), 1-19.

Campbell, D. T., & Stanley, T. D. (1990). *Experimental and quasi-experimental design*. Boston: Houghton Mifflin Company.

Chatterjee, D., Grewal, R., & Sambamurthy, V. (2002). Shaping up for e-commerce: Institutional enablers of the organizational assimilation of web technologies. *MIS Quarterly, 26*(2), 65-89.

Chau, P., & Tam, K. (1997). Factors affecting the adoption of open systems: An exploratory study. *MIS Quarterly, 21*(1), 1-24.

Drakos, N., Di Maio, A., & Simpson, R. (2003). *Open source software running for public office* (Gartner Research Report AV-19-5251). Retrieved

December 2005, from www4.gartner.com/resources/114500/114562/114562.pdf

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. New York: John Wiley & Sons.

Everitt, K., & Lederer, S. (2001). *A usability comparison of Sun StarOffice Writer 5.2 vs. Microsoft Word 2000*. Retrieved November 2, 2005, from http://www.sims.berkeley.edu/courses/is271/f01/projects/WordStar/

Fichman, R. G., & Kemerer, C. F. (1997). The assimilation of software process innovations: An organizational learning perspective. *Management Science*, *43*(10), 1345-1363.

Fichman, R. G., & Kemerer, C. F. (1999). The illusory diffusion of innovation: An examination of assimilation gaps. *Information Systems Research*, *10*(3), 255-275.

IDABC. (2003). *The IDA open source migration guidelines*. Retrieved February 15, 2006, from http://ec.europa.eu/idabc/servlets/Doc?id=1983

Katz, M. L., & Shapiro,C. (1985). Network externalities, competition, and compatibility. *The American Economic Review*, *75*(3), 424-440.

Landeshauptstadt München. (2003). *Clientstudie der Landeshauptstadt München*. Retrieved February 2, 2006, from http://www.muenchen.de/aktuell/clientstudie_kurz.pdf

Marson, I. (2005). *Linux brings hope to Spain's poorest region*. Retrieved January 10, 2006, from ZDNetUK Web site: http://insight.zdnet.co.uk/software/linuxunix/0,39020472,39197928,00.htm

Massachussets State. (2005). *Enterprise technical reference model*. Retrieved on February 2, 2006, from http://www.mass.gov/portal/site/massgovportal/menuitem.769ad13bebd831c/14db4a11030468a0c?pageID=itdsubtopic&L=4&L0=Home&L1=Policies%2c+Standards+%26+Legal&L2=Enterprise+Architecture&L3=Enterprise+Technical+Reference+Model+-+Version+3.5&sid=Aitd

Miller, R. (2002). *Largo loves Linux more than ever*. Retrieved February 2, 2006, from Newsforge Web site: http://www.newsforge.com/print.pl?sid=02/12/04/2346215

Nichols , M., & Twidale, M. B. (2003). The usability of open source software. *First Monday*, *8*(1), Retrieved July 6, 2006, from http://firstmonday.org/issues/issue8_l/nichols/

Palmer, M. (2005). *Microsoft to give Office access to rivals*. Retrieved February 2, 2006, from Financial Times Online Web site: http://news.ft.com/cms/s/e9f5c0f8-5ab7-11da-8628-0000779e2340.html

OASIS—Organization for the Advancement of Structured Information Standards. (2005). *OASIS Open Document Format for Office Applications (OpenDocument)*. Retrieved December 5, 2005, from http://www.oasis-open.org/home/index.php

Pfleeger, C. P., & Pfleeger, S. L. (2002). *Security in computing* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Rogers, E. (1995). *Diffusion of innovations*. New York: The Free Press.

Rossi, B., Scotto M., Sillitti, A., & Succi, G. (2005). Criteria for the non invasive transition to OpenOffice. In *Proceedings of OSS2005*, Genova, Italy.

Shapiro, C., & Varian H. R. (1999). *Information rules: A strategic guide to the network economy*. Cambridge, MA: Harvard Business School Press.

Sillitti, A., Janes, A., Succi, G., & Vernazza, T. (2003, September 1-6). Collecting, integrating and analyzing software metrics and personal software

process data. In *Proceedings of EUROMICRO 2003*, Belek-Antalya, Turkey.

Stadt Nürnberg. (2004). *Strategische Ausrichtung im Hinblick auf Systemunabhängigkeit und Open Source Software*. Retrieved February 2, 2006, from http://online-service.nuernberg.de/eris/agendaItem.do?id=49681

Stadt Wien. (2004). *Open Source Software am Arbeitsplatz im Magistrat Wien*. Retrieved February 15, 2006, from http://www.wien.gv.at/ma14/pdf/oss-studie-deutsch-langfassung.pdf.

KEY TERMS

Assimilation: Passive adoption of a new practice or behaviour, generally resulting from participating in activities where such behaviour is used or is expected.

Data Standard: Denotes a standard to store data in information science. The most important classification is between open/closed data standards according to the publishing of the specification, although the exact classification is still controversial.

Deployment: Use of an item on a relatively large scale.

Lock-In: In economics, denotes a situation in which a consumer cannot change his buying decision without incurring in high switching costs. For example, a user may be bound to a certain software provider for the services offered, by switching to another provider he may incur in high switching costs to change his system infrastructure.

Network Effect: In economics, denotes a demand-side effect, by which the utility given to a certain good increases with the number of successive users adopting it. Information goods are a typical example of good that manifest this behaviour.

Migration: Transitioning from one particular software package to another.

Office Automation: The set of software necessary to provide the necessary integration between the information system and the standard office activities. The minimal set of instruments includes a word-processor, a spreadsheet, software for presentations, and a small personal database application.

ENDNOTES

1. GnuLinex, http://www.linex.org/
2. Microsoft Office, http://www.microsoft.com/office/editions/prodinfo/default.mspx
3. OpenOffice.org, http://www.openoffice.org
4. The study is dated November-December 2001 and refers in particular to the comparison between Sun StarOffice Writer 5.2 and Microsoft Word 2000.
5. As functionalities we intend the opening of one window inside an application, as for example—to remain in the office automation field—the paragraph options or the Save As screen. At the time of both experimentations we could not collect more fine-grained data, like the invocation of keyboard shortcuts that gives us the exact correspondence with the functions used in a program.
6. Starting from version 2.0 OpenOffice.org offers also the Base component to provide simple database functionalities.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited byK. St.Amant & B. Still, pp. 309-326, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 4.18
Open Source Software Adoption:
Anatomy of Success and Failure

Brian Fitzgerald
Lero – Irish Software Engineering Research Centre and University of Limerick, Ireland

ABSTRACT

Current estimates suggest widespread adoption of open source software (OSS) in organizations worldwide. However, the problematic nature of OSS adoption is readily evidenced in the fairly frequent reports of problems, unforeseen hold-ups, and outright abandonment of OSS implementation over time. Hibernia Hospital, an Irish public sector organization, have embarked on the adoption of a range of OSS applications over several years, some of which have been successfully deployed and remain in live use within the organisation, whereas others, despite achieving high levels of assimilation over a number of years, have not been ultimately retained in live use in the organization. Using a longitudinal case study, we discuss in depth the deployment process for two OSS applications – the desktop application suite whose deployment was unsuccessful ultimately, and the email application which was successfully deployed. To our knowledge, this is the first such in-depth study into successful and unsuccessful OSS implementation.

INTRODUCTION

Open source software (OSS) has elicited a great deal of research interest across a range of disciplines since the term was introduced in 1998. Much of this research, however, has focused *inward* on the phenomenon itself, studying the motivations of individual developers to contribute to OSS projects, or investigating the characteristics of specific OSS products and projects, for example. Far less has been done in looking *outward* at the process of OSS adoption and implementation in organizations. The need for rigorous research into this process is important for several reasons: Firstly, recent estimates suggest widespread adoption of OSS: A survey of public administrations in 13 European countries reported that 78% were using open source (Ghosh and Glott,

2005). Similarly, a large-scale survey in the US estimated that 87% of organizations were using open source software (Walli et al., 2005). However, these surveys did not distinguish between primary adoption (the initial decision to adopt at the organizational level) and secondary OSS adoption (the actual implementation process which involves adoption by individuals throughout the organization). Primary and secondary adoption have been identified as quite different scenarios (Gallivan, 2001; Zaltman et al., 1973). This distinction and the problematic nature of OSS adoption is readily evidenced in the fairly frequent (and somewhat controversial) reports of problems, unforeseen hold-ups, and outright abandonment of OSS implementation over time (e.g. Birmingham City Council (Thurston, 2006); Crest Electronics (Turner, 2005); Scottish Police (Niccolai, 2005), Newham Council (McCue, 2004).

Here we present the case of Hibernia Hospital, an Irish public sector organization, who embarked on the adoption of a range of OSS applications. Some of these applications have been successfully deployed and remain in live use within the organisation, whereas others, despite achieving high levels of assimilation over a number of years, have not been ultimately retained in live use in the organization. Using a longitudinal case study, we discuss in depth the deployment process for two OSS applications—a desktop application suite whose deployment was ultimately unsuccessful and abandoned, and an email application which was successfully deployed. To our knowledge, this is the first such study into successful and unsuccessful OSS implementation,although there have been several studies of OSS adoption (e.g. Lundell et al., 2006; Rossi et al., 2006; Ven et al., 2006; Zuliani and Succi, 2004).

As a starting point, we drew on Gallivan's (2001) process framework for studying secondary adoption of technology. This framework extends the classical diffusion of innovation theory of Rogers (1962-2003) by drawing on critiques of this theory (e.g. Fichman, 1992; Fichman and Kemerer, 1999; Moore and Benbasat, 1991). Our goal in this study was not to test a factor model of OSS deployment but rather to provide a rich description of the process of successful and unsuccessful OSS adoption in a single organizational context, with a focus more on theory development rather than theory testing.

Furthermore, researchers have identified a tendency in traditional innovation adoption research towards a pro-innovation bias (Fichman, 2004; Rogers, 2003). As a result, innovation is invariably seen as beneficial and positive for all participants, and, indeed, more has been written about successful adoption than rejection. Thus, our study here of the successful and failed adoption of OSS products can provide useful insights and contrasts which can contribute to theory development in this area.

The remainder of the paper is structured as follows. In section 2, we discuss the process model approach adopted here and present the conceptual framework we use in the study. Following this, section 3 discusses the research approach adopted. Section 4 presents the adoption process trajectories for both OSS applications in Hibernia. Following this, in section 5 we discuss this deployment using the framework derived in Section 2. Finally, the conclusions and the implications of the study for a theory of OSS deployment are discussed.

CONCEPTUAL GROUNDING

Process versus Factor Research Models

Process and factor approaches have been identified as alternative but complementary approaches to research (e.g. Markus and Robey, 1988; Mohr, 1982). Briefly summarising, factor research is concerned with identifying predictor and outcome variables. These are cast as independent and dependent variables and the research focus tends towards rigorous measurement of the variables

and statistical analysis of the relationship between them. The variables are assumed to be causally related with the predictor/independent variable accounting for variation in the outcome/dependent variable. However, such research cannot provide any in-depth explanation as to how and why the variables may be related (Newman and Robey, 1992). Process model research, on the other hand, seeks to elaborate the story of the underlying dynamics which reveals how and why outcomes are reached over time. In this study, given the lack of research on organizational adoption of OSS, successful or otherwise, a process model which could afford increased understanding of significant OSS adoption events was important.

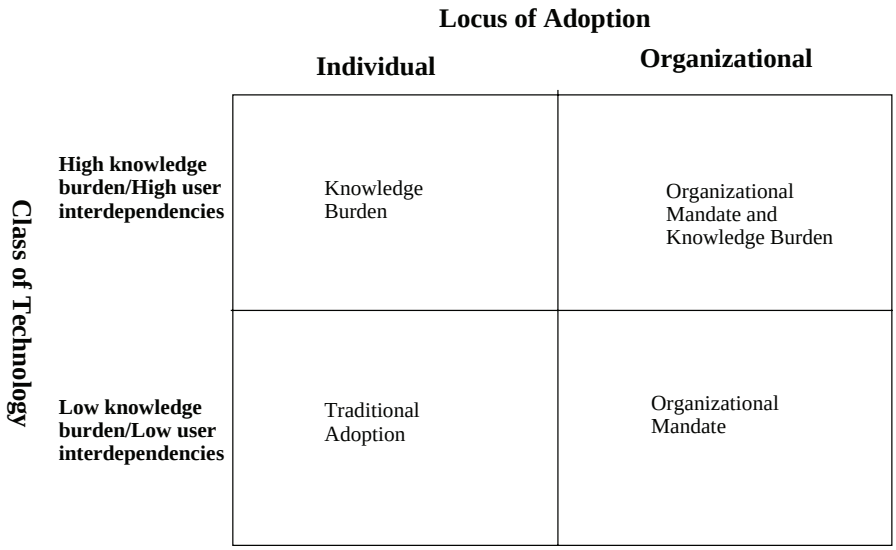
While process and factor models are acknowledged as complementary, researchers have warned against combining into a single model (Markus and Robey, 1988; Newman and Robey, 1992; Mohr, 1982). This argumentation is based on the fact that the models differ in form and operate a different model of causality. That is rather than a 'push type' causality of factor models where the levels of the independent variables cause the levels of the dependent variables, in the process model approach, outcomes are implied by preceding events—a 'pull-type' causality.

Notwithstanding this argument, several researchers have combined process and factor models to good effect (Gallivan, 2001; Sambamurthy and Poole, 1992; Shaw and Jarvenpaa, 1997). Indeed, combining factor and process models has been advocated when the focus is on understanding the adoption events and the factors that promote or constrain adoption outcomes (Gallivan, 2001; Shaw and Jarvenpaa, 1997). Therefore a somewhat hybrid model was followed here in that an overall conceptual framework of innovation adoption was identified, primarily as a conceptual lens to theoretically ground the study (Klein and Myers, 1999) and also as a means of bounding the study focus (Newman and Robey, 1992).

Innovation Adoption Research

In a review of technology diffusion research, Fichman (1992) proposes a 2x2 matrix of innovation adoption contexts where the axes are locus of

Figure 1. IT diffusion classification matrix (from Fichman, 1992; Gallivan, 2001)



innovation adoption (individual or organization) and class of technology to be adopted (low user interdependencies and knowledge burden versus high user interdependencies and knowledge burden). The model is presented in Figure 1.

Fichman argues that the assumptions underpinning traditional innovation adoption models hold best for the lower-left quadrant in Figure 1. In our study, we focus on organizational adoption of open source which is best characterised by organizational mandate to use the technology and also extensive knowledge required to overcome barriers to implementation and use. This is represented by the upper-right quadrant in Figure 1.

Such a characterisation of OSS as a technology subject to organizational mandate, high user interdependencies and high knowledge burden is justifiable for a number of reasons. Firstly, the OSS products that we focus on in this study include desktop and email application platforms. Both of these represent horizontal infrastructure systems in widespread use within organizations. As such they would be subject to the IT governance policy within an organization, and use of these systems would typically be organization-wide.

Furthermore in terms of knowledge burden, Fichman and Kemerer (1999) argue that IT assimilation may be hindered by knowledge barriers due to the learning required to obtain the necessary deep knowledge and skills to successfully deploy complex technologies. These knowledge barriers cause deployment to be a risky venture for an organization, but it may still undertake deployment so as to be in a position to avail of benefits at the appropriate time.

These issues are especially pertinent in the case of OSS. Given the fact that OSS is quite a new phenomenon, there is no well-established and codified base of knowledge that can guarantee successful deployment. OSS adoption represents a significant risk and a fundamental change in how software is acquired and maintained (Agerfalk and Fitzgerald, 2008). For example, there is usually no vendor to market an OSS product and verify that the product meets required functionality. Nor is there the automatic provision of the guaranteed maintenance contract that comes with the acquisition of proprietary software. These issues represent a considerable knowledge burden for organizations that embark on the process of OSS adoption.

A Conceptual Framework for the Innovation Adoption Process

Gallivan (2001) draws on a wide range of innovation adoption research, including Rogers diffusion of innovation, Davis' (1989) TAM model and, particularly the Theory of Planned Behavior (Ajzen, 1985) to propose a process framework specifically addressing secondary adoption and organizational assimilation of technology (see Figure 2).

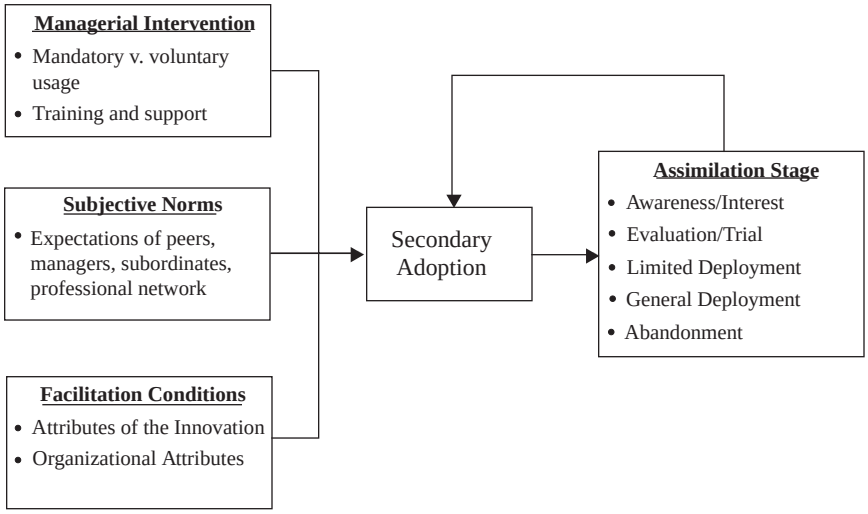
This framework operates at quite a high-level in identifying issues relevant to the IT adoption process. Here we briefly discuss the components of the framework and how they are relevant in an open source context. Later, we use this framework to structure our discussion of the deployment process for the various OSS applications in Hibernia.

Managerial Intervention

Managerial intervention refers to the actions taken and resources made available by management to expedite secondary adoption. Gallivan identifies issues such as voluntariness of adoption, training and support here. Voluntariness has also been proposed as a significant factor in other innovation research (Moore and Benbasat, 1991), and the issue of organizational mandate in relation to OSS adoption was discussed above.

Management support is undoubtedly critical for radical, high-risk initiatives such as OSS deployment since it contravenes the traditional model where ongoing support is legally guaranteed by a vendor. Indeed, management support

Figure 2. Secondary adoption process (adapted from Gallivan, 2001)



is likely to become even more important in the future as OSS adoption moves out of the domain of invisible infrastructure systems to more visible, high-profile applications.

Subjective Norms

Subjective norms have to do with individual beliefs about how relevant peers and co-workers expect them to behave in relation to the technology. This can lead to greater effort to learn about and adopt an innovation or even cause abandonment of a technology. This issue has resonances with attributes of the innovation, such as compatibility and image, discussed below.

From a values and norms perspective, the ideology represented by OSS may have significant implications. The importance of ideological values has been illustrated in several studies of OSS. For example, Stewart and Gosain (2006) identify how adherence to an overarching OSS community ideology facilitates team effectiveness. Similarly, the protracted and heated dispute over several years among the Linux kernel development community concerning the use of a proprietary version control system (BitKeeper) represented an ideological crisis for many in that community, and certainly influenced the choice of adoption and non-adoption of the technology (Shaikh, 2006).

Facilitating Conditions: Attributes of the Innovation and Organization

Much prior research on innovation adoption has focused on attributes of the technology and the organization. Rather than discuss exhaustively the range of attributes that have been identified, a number of attributes are briefly presented here and we discuss how they are relevant to an OSS context.

Attributes of the Innovation

Rogers (2003) identifies five key perceived attributes of an innovation that influence the outcome of the adoption process:

- **Relative advantage:** the extent to which an innovation is perceived as being better than its precursor.
- **Compatibility:** the degree to which an innovation is perceived as being consistent with the existing values, norms, needs and past experiences of potential adopters.
- **Complexity:** the degree to which an innovation is perceived as difficult to understand and use.
- **Trialability:** the degree to which an innovation can be experimented with.
- **Observability:** the degree to which the results of an innovation are visible to others.

In short, Rogers suggests that innovations will diffuse more quickly and successfully when they are readily trialable, of high relative advantage, compatible with the status quo, not too complex to use, and where use is readily observable to others. These attributes have been confirmed in many studies. Additional attributes, such as image and voluntariness, have been identified (Moore and Benbasat, 1991), and indeed some attributes have been found to overlap—relative advantage and compatibility, for example (Moore and Benbasat, 1991; Carter & Belanger, 2006). While Rogers' work is applicable to innovation in general, in the specific category of IT adoption, the technology assessment model (TAM) has been proposed by Davis (1989) with two central attributes—perceived usefulness and perceived ease of use. These are subsumed by the Rogers attributes of relative advantage and complexity respectively.

These attributes are readily apparent in the context of OSS. In terms of relative advantage, compatibility and complexity, for example, many OSS products have been purposefully designed to replicate proprietary counterparts. There should therefore be a sense of familiarity thus mitigating adoption problems in relation to these attributes. On the other hand, the observability of OSS use is less obvious due to the strategy of replicating proprietary software. For example, it is very difficult to tell the difference between MS Word, Excel and Powerpoint and the respective OpenOffice counterparts, Writer, Calc and Impress, merely by looking at users working online on these applications.

Given that acquisition of OSS products is usually extremely straightforward, often as simple as a zero-cost download from a web site, trialability is greatly facilitated in the specific case of OSS. Indeed, many OSS implementations up to now have been deployed by technologically-literate IT personnel who have not sought organizational approval to acquire the products.

In Rogers's work, image is considered to be subsumed in relative advantage, but Moore and Benbasat concur with previous studies which have shown image to be a separate factor (Tornatzky and Klein, 1982). Image is defined as the degree to which an innovation can enhance one's image or social status. This has emerged as a complex issue in relation to open source. Studies of the motivation of OSS developers reveal that the intrinsic satisfaction of belonging to a meritocratic community where talented developers can progress to become core developers is a powerful force (e.g. Kuk, 2006; Lakhani and Wolf, 2006). Similarly, from a user perspective, public administrations, particularly in Europe, have been enthusiastically seeking to deploy open source, seeing it as a positive initiative which frees them from the constraints of a proprietary software industry. However, other reports have found that developers do not necessarily embrace open source (Zachary, 2003), and equally, from a user perspective, there may be resistance to the use of open source products (van Reijswoud, 2005).

Organizational Attributes: Absorptive Capacity

Fichman (1992) recommends that theoretical frameworks of traditional innovation research be complemented by additional perspectives, including absorptive capacity (Cohen and Levinthal, 1990). Absorptive capacity refers to an organization's ability to recognise the value of new information, absorb it and subsequently leverage it productively. An absorptive capacity perspective has been used by Daniel et al. (2006) to study OSS development group performance in relation to knowledge acquisition and transfer. However, absorptive capacity certainly seems relevant for OSS adoption more generally. The ever increasing number of OSS applications appearing in the marketplace represents a significant knowledge challenge to be overcome—for example, the knowledge of what applications exist, which ones are most viable, how well they are supported, what functionality they offer, how they may be integrated with other OSS or proprietary applications. Indeed, developers in the past have referred to the "exhilarating succession of problem-solving challenges" when installing OSS products (Sanders, 1998). Furthermore, given that there is no tried and tested roadmap indicating a clear series of steps to guarantee successful deployment, organizations cannot expect to have the type of lengthy experience with OSS deployment that could guarantee success. Thus, the process of OSS implementation is clearly one where absorptive capacity may play a crucial role.

Secondary (Individual) Adoption Process

Gallivan suggests this component to address the details of the organizational implementation process whereby individuals throughout the organization adopt the innovation. This is taken to include when and how the innovation is adopted, what obstacles are encountered and how these influence the outcome and the degree of organizational assimilation.

Level of OSS Assimilation

Given that technology acquisition and deployment represent different assimilation events, the level or degree of assimilation can be viewed as a staged process from awareness/interest through to general deployment. The following, adapted from Fichman & Kemerer (1997), indicate the range of OSS assimilation levels experienced over time in Hibernia.

- **Awareness/Interest:** Key decision makers in organization aware of OSS and actively committed to learning more
- **Evaluation/Trial:** Organization has acquired specific OSS products and has initiated evaluation or trial
- **Limited Deployment:** Organization has established a program of regular but limited use of OSS products
- **General Deployment:** Organization is using OSS products for at least one large and mission critical system
- **Abandonment:** Organization has discontinued live use of OSS products

RESEARCH APPROACH

At a high level, research epistemologies may be classified as positivist, interpretivist or critical (Chua, 1986), although Klein and Myers (1999) recognise that classifying individual research studies is not always straightforward. To the extent that positivist research involves quantifiable measures, formal hypothesis testing and the pursuit of statistical generalization, this research study is not primarily a positivist one. Likewise given that critical research seeks to elucidate the negative and discriminatory conditions inherent in the status quo, this study does not follow a criti-

cal approach. Interpretivist research assumes the social construction of reality through language and shared meanings, and explicitly recognises the importance of a deep understanding of the context in all its inherent complexity. This research is largely compatible with these assumptions and our epistemology is thus closest to the interpretivist one. However, this classification should be tempered with our use of a high-level conceptual framework to ground the research, and to which we also link our findings. Nevertheless, Klein and Myers (1999) recommend the use of a conceptual framework in interpretivist research for such a purpose.

We sought to develop a rich understanding and insight based on a deep analysis of a single case context—what has been termed a "revelatory case" (Yin, 1994). This is also relevant given that there are undoubtedly political and social factors at play in IT assimilation (Fichman & Kemerer, 1999), which are difficult to elucidate in survey research, for example. Also, by definition postal surveys usually only elicit information from a single key informant (or perhaps two) in an organisation. Thus, there is merit in investigating the view of multiple stakeholders in a particular case context, particularly for the complex secondary adoption process (Fichman, 1992; Gallivan, 2001; Rogers, 2003). Also, just as quantitative research highlights findings that are of greatest statistical significance, in qualitative research, the aptness of a respondent's quote can memorably highlight the essence of the research.

Given that the OSS implementation process in Hibernia was not uniformly successful, we chose to focus on two example implementations—the desktop application suite which was unsuccessfully deployed and the email suite which was successful, thus representing 'extreme cases' (Miles and Huberman, 1994). Both applications are broadly similar—they are applicable to users throughout the organization, and there are strong proprietary alternatives in each case. Furthermore, by limiting our study to a single case context, certain factors are controlled to some extent—organizational attributes, for example. This makes it easier to isolate the salient elements influencing the success or failure of the process.

Data Collection and Analysis

In terms of data collection, a number of sources were drawn on (see Table 1). Over a three-year period, a series of formal face-to-face interviews, and more informal telephone interviews and meetings, were conducted with IT staff, key users and relevant management. In addition, interviews were conducted with external consultants from local firms who provided technical support for Hibernia's OSS implementation, and also with external experts who were familiar with overarching IT policy issues in the hospital sector. Formal interviews were generally of one to two-hour duration. These interviews were complemented by comprehensive reviews of documents and presentations, and fortnightly project workshops of half-day duration over a 12-month period. Furthermore, in the context of a collaborative funded research project between the author's university and Hibernia, there was prolonged and extensive access and interaction with the relevant personnel. Thus, clarification and refinement of emergent issues happened frequently through informal interviews and meetings with key personnel.

While the initial primary adoption of OSS in Hibernia was a straight-forward organizational decision, it soon became obvious that the secondary adoption of specific OSS applications by individuals throughout the organization would not be straightforward. Given this, we drew on the conceptual framework (Figure 2) which had been specifically designed to investigate secondary adoption (Gallivan, 2001). Data analysis occurred over two phases. Firstly, all the data gathered over the entire duration of the study was analysed from the high level perspective of the conceptual framework. Examples of issues which related to managerial intervention, subjective

Table 1. Data sources

Activity	Criteria
22 interviews in Hibernia and with relevant external experts	Interviews with 17 IT staff, OSS users and management in Hibernia over the period Feb 2003 to Nov 2006 Interviews with three consultants from three local organisations providing support to Hibernia (Feb 2004, Jun 2004) Interviews with two Government and Health Board personnel about general OSS implementation policy issues in the health sector in Ireland (Nov 2004 and Jun 2005)
Fortnightly half-day workshops	In context of a joint research project, half-day OSS implementation workshops held fortnightly in the period Mar - Dec 2004
Informal meetings/interviews	Frequent informal interviews/meetings with relevant staff refine/clarify issues in the period Feb 2003 to Nov 2006
Project Documentation	Various reports and presentations relevant to the OSS implementation process
Feedback presentations	Findings were presented at three workshops attended by relevant staff

norms, organizational attributes, and attributes of the innovation were identified. Following this, in a second coding phase, the specific details which underpinned the high-level constructs were identified thereby elaborating the high-level constructs of the framework. The method of constant comparison (Glaser and Strauss, 1967; Miles and Huberman, 1994) was used here as both cases of successful and unsuccessful OSS deployment were also drawn on to help isolate the most salient issues.

While generally guided by an interview protocol which specified the specific topics of research interest, interviews were conducted in a reflexive manner, in that it was accepted that responses to certain questions could stimulate new awareness and interest in particular issues which could then require additional probing. This strategy is also recommended by Eisenhardt (1989) who labels it "controlled opportunism". This probing was also a feature of the informal interviews and meetings which followed the formal interviews.

Reliability and Validity Issues

Research reliability is concerned with the consistency with which research results can be replicated. A frequent criticism of interpretivist research is that due to its subjective nature, replication is problematic. While acknowledging that interpretivist analysis would not expect all researchers to interpret the findings in exactly the same way, it is important that the research process be transparent and accessible to others. To help address research reliability, Yin (1994) recommends the use of a case study database and protocol. This strategy has been operationalised in other interpretivist case studies (e.g. Kirsch, 2004)

and we followed a similar approach here. A case study database was established which contained the raw field notes, transcribed interviews, and coding of this data according to our conceptual framework. The case study protocol specifies the criteria for selecting the case applications, the choice of whom to interview, and the interview protocol in terms of broad interview questions.

Research validity is concerned with whether the actual research in practice matches what it purports to be about. In interpretive research this is primarily concerned with the "truth value" of the research (Miles and Huberman, 1994).

Construct validity deals with the extent to which the constructs as operationalized relate to the research phenomenon being studied. In this study, given the lack of research on OSS adoption, and our goal of theory development, construct validity was important. Yin (1994) describes three tactics to deal with construct validity: the use of multiple sources of evidence, the establishment of a chain of evidence, and key informants reviewing draft findings. In this case, the collection of data on the same phenomenon from multiple interviewees both within and external to Hibernia, together with information gleaned from project documents and presentations, helped address the multiple sources of evidence criterion. In relation to the chain of evidence criterion, this was addressed through the establishment of a case study database, the rigorous analysis and coding of data according to the conceptual framework, illustration of the theoretical constructs with quotes from interviewees who fulfilled a variety of roles in the implementation, and the process description of the deployment trajectory over time. Finally, key informant review and feedback was addressed in several workshops in the context of a joint research project on OSS implementation in Hibernia, and also several draft reports and presentations on the topic were reviewed by Hibernia staff.

External validity is concerned with the extent to which a study's findings can be generalised. One of the limitations of this study might appear to be the fact that it is based on a single case and thus there is limited scope for generalization. However, Lee and Baskerville (2003) identify a fundamental and long-standing misapplication of generalization whereby researchers have solely focused on statistical sampling-based generalizability from a sample to a population, and have sought to overcome the perceived problem of attempting to generalize to other settings beyond the current one. Following this conventional model, researchers have suggested increasing sample size or number of case study organizations, but Lee and Baskerville argue cogently for the ultimate futility of this flawed strategy. They propose an overarching framework that proposes four distinct categories of generalizing, only one of which corresponds to statistical sampling-based generalization. One of the other categories in their framework, that of generalizing from empirical description to theoretical statements, is more applicable to our research study. This view of generalizing from thick description to theoretical concepts, specific implications and rich insight is also recommended as a strategy by Walsham (1993) and Klein and Myers (1999, p.75) who argue for such a theoretical link as being key to distinguish "interpretive research…from just anecdotes". In this study, the findings were analysed and integrated using the theoretical framework derived from Gallivan (2001).

OSS ADOPTION IN HIBERNIA HOSPITAL

Hibernia Hospital, which began as a merger of two of the oldest hospitals in Ireland, operates in a public sector environment, employing around 3,000 staff directly, which would make it quite a large organization by Irish standards. Similar to many other organizations worldwide, Hibernia's IT budget had undergone a significant contraction since 2000 in the wake of the increased budget

in the lead up to the Y2K. For example, in 2003, Hibernia faced an overall budgetary shortfall of €17 million. Further compounding this issue, was the fact that Hibernia would face an annual expenditure in the region of €1 million just to achieve compliance with the licensing conditions in the proprietary software products in use. It was clear that this level of funding would not be available. Thus, Hibernia was faced with the choice of either reducing the overall level of service to cope with cost restrictions, or embarking on some radical innovation in implementing less costly alternatives. Consequently, it began to investigate what could be found in the open source market-place. The IT staff in Hibernia undertook an extensive phase of desk research into various OSS products over a six-month period. The quality of the exchanges on SourceForge and Slashdot were sufficient to convince the IT Manager that OSS was worth investigating further. Some direct experimentation with downloaded OSS programs was then sufficient to convince him that the risk involved was acceptable.

StarOffice Desktop Suite

StarOffice is available from Sun Microsystems who were also the driving force behind OpenOffice. Some proprietary software is bundled with StarOffice, which prevents it being offered on the same terms as the pure open source, OpenOffice, with which it shares a common code base. Hibernia decided to implement StarOffice, as Hibernia could then purchase support from Sun. This was considered important to mitigate the risk in embarking on a radical new initiative such as OSS deployment.

In February 2002, Hibernia began the roll-out of Sun's StarOffice 5.2 desktop suite. This deployment was very problematic for users and the technical staff. However, this was felt to be largely due to problems in that version of StarOffice. In September 2002, StarOffice 6.0 was deployed with some support from Sun. However this was also troublesome. The IT Manager wanted to pursue a thin client strategy based around the concept that all applications should be downloaded from the network where practical. The StarOffice package was initially loaded onto a single Linux server, but this became overwhelmed, and it was then clustered to sustain a dual server strategy. Despite this, users continued to lose network connections in an unpredictable fashion. This inevitably increased frustration and tension amongst the entire workforce who were dependent on these tools. The IT Manager conceded that:

"we stuck with the network solution too long. It was only after a series of ferocious encounters with users—and with my own staff—that I recognised that we had to shift".

StarOffice was reinstalled on the desktop instead for those who wanted it, which did improve the situation somewhat according to technical staff, In November 2003, Hibernia installed StarOffice 7.0. This solved many of the existing problems, to the extent that the IT Manager could report that there were no open bug reports in Hibernia for StarOffice 7.0. Nevertheless, the users' perception of the StarOffice system appears to have been damaged irreparably.

Further compounding the problems was the fact that when Hibernia started StarOffice implementation in 2002, there was very little by way of training material. Thus, a lot of material had to be prepared internally which increased the workload for IT staff and trainers.

Even though the move to StarOffice was mandated, not everyone was obliged to migrate. The CEO, although a committed supporter who mandated the move to OSS, did not become a StarOffice user. In addition to this, Hibernia comprises many largely autonomous units which behave independently and raise research funds to support their activities. Across these units about 120 users chose to ignore the overall move to StarOffice. Typically, these users had sufficient

funds to remain independent of central IT support. However the IT Manager informed them that this would have consequences in that they would have to assume responsibility themselves for ensuring that the hardware which they use is upgraded, and provide resources for future maintenance upgrades, etc.

Email Platform

Prior to the move to OSS, Hibernia's email system was a proprietary one with a 500-user license limit. This limit had been reached and the IT Manager had to refuse recurring requests for new email accounts. Hibernia initially adopted the SuSE eMail application which was an open source email platform supported by Novell following acquisition of SuSE Linux. Given that there was no upper limit on the number of user email accounts with the SuSE eMail application, Hibernia sought to satisfy increased user demand for extra email accounts. However, when it reached about 700 user email accounts, the SuSE eMail system became prone to frequent problems of hanging and crashing. Hibernia had paid a consultant a once-off fee to implement the SuSE application initially. As with StarOffice, Hibernia sought to establish a support contract for SuSE eMail with Novell. However, the IT Manager reported that Novell at the time did not appear to be interested in offering an ongoing support contract for SuSE eMail. In the absence of a solution to the problems with SuSE eMail, Hibernia began to look for an alternative open source email platform. A multi-product open source email platform was established, comprising the Postfix mail transport agent, OpenLDAP directory access protocol service, SpamAssassin mail filter, and the SquirrelMail email client. After some initial teething problems with integration, this mixed architecture emerged as an extremely stable and scaleable email solution. Given that there were no license-imposed constraints on the number of users, Hibernia initiated a policy whereby all staff were entitled to an email account. Hibernia's IT staff were also able to add functionality to re-route emails to mobile phones and user PDAs. This, together with the impressive filtering capability of SpamAssassin, caused the email platform to be received very favourably by the general user base. At present, Hibernia supports more than 3,000 email accounts. Also, the system scope has been expanded to incorporate certificate-based external email access for about 350 authorised users. Overall, the IT Manager believes that "it would be unthinkable and completely unacceptable" to revert to a 500-user license again.

DISCUSSION OF OSS ADOPTION IN HIBERNIA HOSPITAL

Here we discuss the different implementation trajectories for both open source applications within Hibernia using the conceptual process framework derived above.

Managerial Intervention

Mandatory versus Voluntary Usage

As already mentioned, the decision to move to OSS was given full support by the CEO, largely on the basis that there was no other choice given the cuts in the IT capital budget. Thus, the use of StarOffice was seen as mandatory. This had significant negative implications. Firstly, as the Secretary Manager put it:

> We did not think that StarOffice had been given to us as a bonus. Rather we felt that Microsoft Office had been taken away.

However, even in the case of StarOffice, as already mentioned, a number of users who had sufficient resources were able to opt out of the migration. Also, one department who dealt primarily with fund-raising from external stakeholders

argued for the need to remain with the proprietary system due to having to liaise with these external agencies who solely used proprietary software.

On the other hand, this issue of mandatory usage did not arise in the case of the email platform suite. Hibernia was offering an additional service in terms of email access to those who sought it and who had not been able to get email access in the past. Thus, the email platform was implemented in the context of voluntary user demand rather than there being any perception of mandatory usage by management.

Training and Support

The Secretary Manager was critical of the process by which StarOffice was initially implemented. There was no effective buy-in process in her opinion. A small pilot group which included just one secretary comprised the initial trial. This was inadequate given that the most active users of StarOffice would be the cohort of secretaries in Hibernia. The Secretary Manager suggested that

StarOffice was sold as the same thing as Microsoft Office. A two-page brochure was provided and it was suggested that no training would be needed really.

However, even though StarOffice and MS Office are largely functionally equivalent, menus are constructed differently and terminology is slightly different. Thus, commonly-used options such as Print Preview or Track Changes are labelled differently or are in different sub-menus, with different key-stroke short-cuts. This contributed to a greater feeling of unfamiliarity and incompatibility than is probably warranted given the similarities between the applications.

Given these problems with the deployment of earlier versions of StarOffice, a widespread training and awareness program was created to ensure that the user community could be briefed on the new features in StarOffice version 7.0. While this could certainly address user perceptions in relation to issues of complexity, relative advantage and compatibility, it was not enough to overcome the very negative perceptions associated with StarOffice in Hibernia—this despite the fact that Hibernia have no unresolved problem reports for StarOffice 7.0.

While there was no specific training or extra support in the case of email, any differences between the original proprietary application and the subsequent OSS application have not been perceived as problematic. However, since the user base from email climbed from 500 to over 3,000, the vast majority of the users did not have an existing email application in their work context which they had learned and now needed to unlearn. Also, the fact that there are no alternative email applications elsewhere in Hibernia with which unfavourable comparisons could be drawn helps to minimise this as a problem.

Subjective Norms

In the case of StarOffice, the user base perceived usage as mandatory for those who did not have the resources to maintain an alternative. This led to feelings of resentment which were quick to emerge when problems became apparent. Interestingly, rather than being seen as renegades who failed to comply, the departments and users who were able to remain on the proprietary platform were envied by their colleagues. The Secretary Manager described it:

You meet people and hear that they are using Microsoft, and immediately you ask them how they managed to do that.

One of the key complaints from the administrative staff in Hibernia who moved to the StarOffice platform was that they feared being de-skilled

in relation to their employment prospects if they didn't have skills in popular proprietary applications. In fact, users readily admitted that they would have preferred not to have switched from the proprietary desktop systems to OSS. Additionally, there was further resentment in some quarters to the move to OSS systems, in that some staff appear to feel somewhat 'short-changed' and believe their work is under-valued if they are asked to use OSS systems which cost less that those being used by their counterparts in hospitals elsewhere using proprietary systems.

Attributes of the Innovation

The discussion above identified several innovation attributes that have been found in previous research to influence innovation adoption. Here we discuss the ones most salient to the OSS adoption in this study—image, relative advantage, trialability and observability.

Image

Perhaps the most significant issue for StarOffice was the fact that it quickly gained a negative image, and despite improvements in newer versions of the software, this negative image persisted. One user admitted that when StarOffice was proposed, there was a widespread perception that this was a cheap and antiquated package from "Jurassic Park" which would have limited functionality. This user was genuinely surprised to hear that StarOffice was a modern application which was actively being developed. This negative view was confirmed by an Informatics Nurse who suggested that StarOffice ran into "bad publicity from the outset".

There was a fairly widespread perception within Hibernia that it is prone to disadvantage due to its being on the North side of Dublin, an area traditionally perceived as being disadvantaged (at least by those who are from there), and that StarOffice was just another example of this disadvantage working against them. Indeed, in typical Northside Dublin fashion, users have coined the succinct and disparaging term, "Star Bleedin' Office", to refer to the system.

Significant in this perception was the fact that no other hospital in Ireland had chosen to implement an open source desktop. The Secretary Manager suggested that the budget-cutting rationale behind the implementation of StarOffice caused it to be perceived as a "poor man's Microsoft", and as a result there was a pre-conceived expectation that it would be problematic.

The negative effect of StarOffice was even suggested to underpin an increased level of absenteeism and stress-related sick leave, according to the Occupation Health department. While, there was no rigorous analysis of employee absences to support this, there was a belief that the stress of moving to StarOffice had been a factor in many stress-related and work leave/absences. It will be interesting to see if the level of stress-related absences also increases when Hibernia revert to a proprietary platform.

The StarOffice image has become quite notorious within Hibernia, to the extent that at meetings to discuss new IT projects, managers have been heard to express the hope that it would not be "another StarOffice". Also, the negative image of StarOffice extended beyond Hibernia. One user described emailing an attachment, which had been saved in StarOffice's proprietary format by default, to a colleague externally. This colleague couldn't open the attachment, and emailed a response saying that the attachment was in that "StarOffice gobbledy-gook".

In sharp contrast, the email platform has no such similar baggage of negative image. While there were problems during the implementation of SuSE eMail, these were quickly overcome when an alternative email system was implemented. Also, these problems only manifested themselves when more than 200 additional users had been given email access. Thus, there was no sense in which the user service had been disimproved in

any way. This has resonances with the relative advantage issue discussed next.

Relative Advantage

Clearly, the initial problems with StarOffice caused users to perceive their original proprietary system as better. There were several problems, particularly with Impress, the StarOffice equivalent of MS PowerPoint. An Informatics Nurse described it:

"I have seen people crying because of Impress. One day I was working on a presentation which I was due to give at 8:30 the following morning. At 5:30pm I checked it and it had become just one blank sheet. I had to go home and recreate it from memory in PowerPoint".

Interestingly, the Informatics Nurse also recalled losing several chapters of her thesis when using MS Word in the past, but there was a sense in which she felt less vulnerable about that. The IT Manager also recalled giving a seminar on OSS at an IT conference attended by several hundred delegates, and his Impress presentation stalled. It was not a happy experience, and certainly, the software which supports people publicly presenting, is not one where problems will be tolerated to say the least. This issue is interesting since only a very small number of actual users would need to deliver presentations, and thus the problems experienced due to the use of Impress were not all that widespread overall. Nevertheless, users seemed to very readily empathise with the negative scenario of problems with a public presentation.

While StarOffice and MS Office are more or less equivalent functionally, there are some differences, and these were were cited in some cases as a reason for not migrating to StarOffice. For example, the Finance Department cited the row number limit in StarOffice Calc which is less than that of MS Excel, as a reason for not migrating.

However, when things settled, particularly following the installation of StarOffice 7.0, a number of benefits became evident in the OSS solution. For example, one of the benefits has been the capacity of StarOffice to exploit its in-built XML capabilities. This is a very powerful feature of the application which enables documents to be structured in such a way that processing logic is built into different sections of the document, i.e. an on-line HR form request, for example, which is then automatically routed to the HR department for processing. This is a significant new feature and provides additional functionality over what was previously offered in Hibernia's proprietary desktop applications.

Also, the StarOffice suite contained an option to create PDF output, which was not available in the MS Office implementation in Hibernia. This was mentioned as a positive benefit by several interviewees. While the Impress application was clearly the most notable problem point, there was support for the other StarOffice applications. Indeed, an interviewee expressed a distinct preference for StarOffice's Calc over MS Office Excel spreadsheet software. However, such perceptions did not scale into an overarching perception of the relative advantage of StarOffice over the proprietary system it replaced.

In the case of email, Hibernia was able to satisfy additional requests for email accounts, thus offering an improved service. While there were significant problems around the use of SuSE eMail, these were very short-lived as Hibernia was able to overcome them quite quickly by implementing an alternative OSS email suite. Again, these problems only occurred after more than 200 additional email accounts had been added, so there was no real perception that the OSS system was operating at a disadvantage. This option of replacing StarOffice was not possible, and even though the problems were ironed out in subsequent installations of StarOffice, it was still perceived as StarOffice and viewed with suspicion.

Also, there were differences with the StarOffice scenario in that users were not aware of al-

ternative email applications in simultaneous use in other departments. Furthermore, users did not typically have an alternative email application at home, which was frequently the case where MS Office was installed on home computers.

Trialability

Trialabilty was a very salient issue in Hibernia's OSS deployment. At the initial stage, Hibernia's IT staff were able to download and experiment with several OSS applications of potential interest. Given the budget situation, the fact that this was a zero-cost exercise was important. Also when Hibernia experienced problems with the SuSE eMail implementation, IT staff were again able to experiment with a range of alternative OSS email applications and quickly implement a very successful and scaleable email solution.

This mode of OSS implementation has continued. When selecting an online e-learning system, Hibernia trialed a number of OSS e-learning systems before selecting the one which appeared to meet their needs best.

Interestingly, this easy trialability appears to have implications for the training and support process in that there was less attention paid to it. If it had been a high cost initiative it would certainly have had a higher profile within the organization and, as a consequence, more attention would have been paid to implementation issues such as pilot testing, training and support. The IT Manager summarised the dilemma:

"If you have a product which costs €1 million—it may seem appropriate to spend €500K on consulting. However if the product costs nothing—then spending €500K somehow seems to be a more difficult decision to take—yet the saving is still €1 million".

Hibernia have learned this lesson and, for example, created a more comprehensive user awareness and training package to support the implementation of StarOffice 7.0.

Observability

Rogers suggests that the extent to which results of an innovation are observable to others will affect its rate of diffusion. However, given that Hibernia wanted to achieve as smooth a transition as possible, the goal was to minimise and downplay the observability of the differences between StarOffice and MS Office to try ensure they would be perceived identically. This is often not difficult in an OSS context since applications have typically been designed to replicate the functionality of proprietary systems. Thus, rather than trying to publicly triumph the use of StarOffice as progressive and something to be enthusiastically yearned for, the emphasis was on downplaying the issue of observability. Given the negative image that has come to be associated with OSS in Hibernia, there is a conscious move to not identify IT applications as open source. Thus, the issue was not highlighted in the case of the email application. Similarly, when Hibernia implemented an OSS e-learning system subsequently, the fact that the systems being trialed were open source was deliberately downplayed as much as possible.

Organizational Attributes: Absorptive Capacity

Hibernia's absorptive capacity in relation to open source adoption was extremely important. The IT Manager accepted that the initial roll-out of StarOffice had been poorly conceived, and Hibernia had learned from that for subsequent implementations of OSS. Clearly, there was an element of risk in proceeding on the OSS path, since ongoing product support would not be provided in the usual way. Thus, there was a need for a complete rethink of the support strategy. In the past Hibernia had always purchased support

from a competent third-party provider. While with OSS this option still existed to some extent, there was a significant difference in expectation associated with OSS, as support was essentially derived from a series of bulletin boards, complemented with external consultancy initially until Hibernia became competent.

Also, it helped that a number of key staff—particularly in the computer operations department—rapidly adapted to the new OSS environment, and the IT Manager described the operations team as the "leaders in the overall adoption of OSS". The bulk of the overall OSS search selection and implementation was actually carried out by the hospital staff. This necessarily involved a process of learning/experimentation. As the staff confidence and familiarity with OSS products grows, the learning cycles were correspondingly shortened. It also helped that Hibernia already had a strong experience of UNIX applications to draw on. So the transition was not as radical as it would have been if staff experience was simply based on GUI-enabled systems administration. In the words of the Linux Systems Administrator, "We are not afraid of the command line interface".

Evidence of increased absorptive capacity in relation to open source is readily evident in the email application deployments. When Hibernia encountered insurmountable problems in relation to the open source SuSE eMail application, IT staff quickly sourced an alternative suite of email applications. This integration of an entire suite of disparate open source email applications into a single integrated email platform represented a significant technological challenge, from identifying suitable applications in the first place, to integrating them into an overall working application.

CONCLUSION

Table 2 summarises the differences in the deployment process for both the OSS desktop and email applications within Hibernia. Rather than elaborating the individual issues here, we will focus more holistically on interaction among the framework elements, as this had a significant influence on OSS implementation. Following this we discuss the implications of the study for research and practice, and discuss the limitations of the study.

Firstly, we focus on trialability and absorptive capacity as these served primarily to facilitate OSS adoption in this study. Trialability of OSS ensured that Hibernia could experiment with OSS applications in the first place and be reasonably confident that the OSS applications available could meet their needs. Also, when problems occurred as in the case of the initial OSS email implementation, an alternative could be found which solved the problem. However, while trialability certainly facilitates the primary adoption of OSS, it is absorptive capacity which ensures that the best OSS candidates are selected and successfully integrated and implemented, thereby facilitating successful secondary adoption.

However, other interlinked elements, such as voluntary versus mandatory adoption and image of the innovation, manifest themselves in such a way as to impede the assimilation of OSS within Hibernia.

Firstly, by being perceived as mandatory due to the necessity of cost-cutting, the adoption of StarOffice was inevitably perceived as reactive. Then when it emerged that some 'more privileged' users could opt out of the move, this two-tier scenario significantly contributed to the negative image bestowed upon StarOffice. When problems occurred, these served to fuel a disproportionately negative perception of StarOffice, despite the fact that it offered certain extra functionality, and that a steady state with no open bug reports was eventually reached following the implementation of StarOffice 7.0. Interestingly, the email application shared a similar deployment trajectory in that it too faced problems initially, which likewise were subsequently overcome, and also there were

Table 2. OSS deployment within Hibernia

	StarOffice Desktop	Email Platform
Managerial Intervention		
- Mandatory v voluntary usage	Usage seen as mandatory for those who could not afford to maintain proprietary alternative.	Access to email application provided upon request, thus usage not perceived as mandatory.
- Training and support	Differences between OSS and proprietary systems downplayed. Low level of training initially using in-house developed material.	No specialised training necessary. No incumbent proprietary system to unlearn.
Subjective Norms		
	Mandatory usage for users who could not afford to maintain proprietary led to StarOffice being perceived as inferior. Staff fear of being deskilled if using OSS, and also that work undervalued if using 'cheap' OSS.	More than 2500 additional users requests for email accounts were satisfied. Thus, uniformly perceived as beneficial.
	Those who opted out of the move to StarOffice envied rather than resented.	Also no alternative email system with against which unfavourable comparisons could be drawn.
Innovation Attributes		
- Image	StarOffice seen as cheap and antiquated "Jurassic Park" option for the disadvantaged. Widespread negative image of StarOffice both within and external to Hibernia.	Email access seen by many as a new privilege which hadn't been available in the past.
- Relative Advantage	Problems and instability led to StarOffice being perceived as inferior. Impress problems particularly cited. Benefits of StarOffice not widely appreciated.	Email a new application for the majority, thus no relative comparison. Also, problems with intermediate SuSE email quickly resolved, and new functionality (routing of email to PDAs) appreciated.
- Trialability	Trialability important, but limited due to lack of alternative OSS desktop suites.	Trialablity critical as Hibernia experimented with a number of OSS email applications.
- Observability	StarOffice and MS Office appear identical on casual observation. Thus, OSS usage is not readily apparent and observable.	Downplayed due to negative image associated with OSS. Not a major issue as no alternative email application in use to compare against.
Organization Attributes		
- Absorptive Capacity	Important as OSS represents new model of software acquisition, implementation and support. Prior learning evident in implementation of StarOffice 7.0.	Very relevant in this case as the first OSS email application had to be replaced by a suite of individual OSS email applications in a novel mixed architecture. High knowledge burden in selecting right applications to include in this architecture and configuring to work successfully together.

advantages in the OSS email system over the original proprietary system. However, the critical difference appears to be that the move to email was not seen as a top-down mandate, rather users could request an email account. Furthermore, there was no cohort using an alternative system who might be perceived as privileged.

The issue of observability was interesting in the Hibernia OSS adoption process. It would not be obvious at a casual glance wthether a user was using StarOffice or MS Office. In the case of email, the fact that the vast majority of users got access to email for the first time within Hibernia would have highlighted the observability issue, in that users were emailing who had not done so before. Thus, this was quickly evident and led to more requests for email accounts. However, due to the negative image of open source that arose from the StarOffice experience, Hibernia sought to downplay the fact that proposed applications, such as e-learning, were open source.

Implications for Research

The study identifies several issues and streams of research which could be further elaborated. To our knowledge, it is the first rigorous analysis of successful and unsuccessful OSS adoption. Our focus on a single case context is also noteworthy as certain important factors inevitably differ across organizations in a multiple case study context, thus making it more difficult to interpret the actual influence and role of individual elements. The study illustrates how a hybrid process variance model can shed light on the innovation adoption process, and in particular, illustrating the complex interaction between the various elements.

The link between trialability and actual deployment of OSS was significant in this study, particularly in the case of email. This also seems to be borne out in OSS adoption more generally. Onetti and Capobianco (2005) report a case study of a software company who offered both traditional proprietary and OSS products. The company found that the ratio of prospects who eventually become actual customers was markedly higher in the case of OSS. Funambol found that in contrast to its traditional sales process for proprietary software, when contacted about OSS products, the prospective customers had already downloaded and actually trialed the OSS product, and were far more likely to become customers paying for support. This altered the business flow from "sales push" to "user pull" (Onetti and Capobianco, 2005).

The Hibernia study also supports the contention by Fichman (1992) and Gallivan (2001) that innovations which involve organizational mandate and high knowledge require the integration of new metaphors and constructs into the research model. The high knowledge burden in successfully deploying OSS supports the view that absorptive capacity is an important issue, and this facet could be further elaborated to potential good effect. Another potentially promising perspective identified by Fichman (1992) is that of critical mass theory (Markus, 1987). This was clearly an issue in this study as the cohort of users who opted out of the move to StarOffice served to weaken the critical mass, whereas the increased number of users who received email accounts worked in the opposite direction.

The critical mass issue is linked to the issue of network externality effects. Some technologies become more valuable through the increasing returns to adoption that arise from the incremental contribution of other adopters. The basic argument is that for some technologies, the potential benefits are greatest when the entire ecosystem of users, suppliers and mediating institutions are in place to fully leverage the deployment of the technology. These increasing returns can arise through positive network externality effects (Katz & Shapiro, 1986), which are readily apparent in OSS, as the phenomenon is fundamentally predicated upon drawing sufficient voluntary interest from a worldwide network of talented hackers with complementary skills to produce industry-

quality software products (Feller et al., 2008). Furthermore, the commercial business model of open source is frequently based on creating a lucrative service and support market by leveraging the zero purchase cost to create a large base of potential customers.

It is also abundantly clear that in the case of OSS a focus on secondary adoption is important. As already mentioned, estimates of OSS adoption by organizations vary greatly. However, it is certainly the case that there could be a marked gap between the initial acquisition of OSS and its eventual large-scale secondary adoption by a critical mass of individual users. Indeed, the essential characteristics of OSS render it very prone to such an assimilation gap: The widespread media coverage leads to high awareness of the concept, while its zero cost results in a very low barrier to initial acquisition. However, the newness of the phenomenon, the manner in which it transgresses traditional software support options, and the lack of any tried and tested approach which could guarantee successful implementation—these all serve to exacerbate the potential assimilation gap between initial acquisition and widespread adoption.

Again, related to this are two elements which were found to be extremely influential in this study—voluntariness of adoption and image of the innovation. While these factors have been identified in some prior research (e.g. Moore and Benbasat, 1991), the issues, and in particular the inter-relationship between them, have not been studied in detail. Voluntariness of adoption is linked to critical mass as organizational mandate can decree that a technology be universally adopted. However, this has implications for how the innovation will be perceived especially by those who feel compelled to use it, thereby affecting the image of the innovation. Furthermore, in previous research, image is assumed to have a positive effect—the use of the innovation is expected to be image-enhancing. In this study, it was certainly the case that innovations are not always seen as conveying a positive image and universally welcomed by those who are expected to use them—the fear of deskilling and the perception of work being undervalued, for example, This has resonances with Fichman's (2004) critique of the dominant paradigm which typically assumes that technology innovation is universally welcomed and perceived as beneficial by all stakeholders. This was certainly not the universal perception from the outset in Hibernia.

Implications for Practice

The study also has a number of implications for organizations who are embarking on OSS adoption. At a higher-level, an open question in prior research has been whether IT implementation should follow a 'big bang' or phased approach, as successful implementations have been reported with both approaches (Fichman, 2004). This is also an open question for OSS migration with researchers recommending both 'big bang' approaches (Ven et al., 2006) and phased pilot approaches (Zuliani and Succi, 2004). The findings of this study would support the 'big bang' approach for each individual OSS application, primarily to avoid the situation where opting out of migration is seen as the preserve of those more privileged, thereby creating image and relative advantage problems subsequently.

This is also related to the issue of whether an organization treats OSS adoption as a mandatory or voluntary initiative. If mandatory, then it is important that OSS is not perceived as a low-cost 'second-rate' alternative, thereby relegating it to an inferior status which individual adopters seek to avoid.

Trialability is more or less a given in OSS, thus ensuring that the initial experimentation with OSS is facilitated. However, the zero cost trialability of OSS should not cause organizations to downplay the importance of implementation issues such as pilot tests, training and support. Also, in the absence of any comprehensive ven-

dor support and marketing, absorptive capacity becomes critical. Identifying potential OSS solutions in the first place is not a trivial issue. Generally, there are no vendors who can answer questions on the suitability and functionality of the software or provide details on reference implementation sites. Similarly, porting OSS to new platforms and integrating OSS systems with other proprietary and OSS systems is far from trivial, as is ongoing support. In this study, some expert consultancy was sourced locally to help with initial implementation issues, but these abilities were acquired in-house over time.

Limitations of Study

One of the possible limitations of this research is that it is a single case study, although we would argue that this should be tempered by the fact that this also afforded an in-depth insight into the process, and also allowed for the keeping constant of potentially confounding factors. Of more importance perhaps is the fact that the organization is a public sector one, and there could be important differences in the OSS adoption process for organizations in other industry sectors.

Also, this study focused on desktop and email applications, both of which are highly visible mass-market applications with strong market-leading proprietary alternatives. By contrast, less visible back-office infrastructure applications such as servers running Linux, Apache, Samba, and the like, may operate differently. Our experience would suggest that the OSS is already dominant in that sector. Similarly, the Hibernia experience would suggest that OSS can be a perfectly acceptable solution for niche applications such as e-learning, particularly when these systems are introduced without having any incumbent system to replace.

Overall, one can conclude that OSS is a very viable alternative for organizations. The main problems arise in the implementation process, rather than arising due to problems of a technical nature, as the latter are usually ironed out very quickly. Indeed, the acid test is perhaps the fact that despite any problems with StarOffice, Hibernia operated effectively as a hospital throughout this period.

ACKNOWLEDGMENT

I would like to acknowledge in particular the helpful feedback from Geoff Walsham, and also the suggestions of the Editor and two anonymous reviewers. This work has been financially supported by the Science Foundation Ireland (SFI) award to Lero—the Irish Software Engineering Research Centre, and the EU FP6 project OPAALS and EU FP 7 project NEXOF-RA.

REFERENCES

Agerfalk, P and Fitzgerald, B (2008) Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy, *MIS Quarterly*, Vol 32, No. 3, pp. 385-410.

Ajzen, I. (1985), From Intentions to Actions: A Theory of Planned Behavior, in *Action Control: From Cognition to Behavior*, ed. Julius Kuhl and Jeurgen Beckmann, New York, NY: Springer, pp.11-39.

Carter, L and Belanger, F (2006) The Influence of Perceived Characteristics of Innovating on e-Government Adoption, *Electronic Journal of E-Government*, Vol 1, No 2.

Chua, W. (1986) Radical Developments in Accounting Thought. Accounting Review, 61, 601.

Cohen, W. M. & Levinthal, D. A. Absorptive Capacity: A New Perspective on Learning and Innovation, *ASQ*, 35 (1990), 128-152.

Daniel, S, Agarwal, R and Stewart, K (2006) An absorptive capacity perspective on OSS devel-

opment group performance, *27th International Conference on Information Systems*, Milwaukee Dec 2006.

Davis, F. (1989) Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology, *MIS Quarterly*, Volume 13, pp. 319-340.

Eisenhardt, K. (1989) Building theory from case study research, *Academy of Management Review*, *14*(4), 532-550

Feller, J, Finnegan, P, Fitzgerald, B and Hayes, J (2008) From peer production to productization: a study of socially-enabled business exchanges in open source service networks, *Information Systems Research*, Vol 19, No. 4.

Fichman, R.G. (1992). Information Technology Diffusion: A Review of Empirical Research," in J.l. DeGross, J.D. Becker, and J.J. Elam (Eds.), *13th International Conference on Information Systems*, Dallas, TX, pp. 195-206.

Fichman, R. G., (2004) "Going Beyond the Dominant Paradigm for IT Innovation Research: Emerging Concepts and Methods", *Journal of the Association for Information Systems*, 5(8)

Fichman, R. G., and Kemerer, C. F. The Assimilation of Software Process Innovations: An Organizational Learning Perspective, *Management Science* (43:10), 1997, pp. 1345-1363.

Fichman, R.G. and Kemerer, C.F., (1999) "The Illusory Diffusion of Innovation: An Examination of Assimilation Gaps," *Information Systems Research*, 10(3), 255-275.

Gallivan, M (2001) Organizational adoption and assimilation of complex technological innovations: development and application of a new framework, *Data Base*, Vol 32, No 3, pp. 51-85.

Ghosh, R and Glott, R (2005) Results and policy paper from survey of Government authorities, Technical report, MERIT, University of Maastricht, Free/Libre and Open Source Software: Policy Support.

Glaser, B. & Strauss, A. (1967) *The Discovery of Grounded Theory,*, Aldine, Chicago.

Kaplan, B. and Duchon, D. (1988) Combining qualitative and quantitative methods in IS research: a case study, *MIS Quarterly, 12*(4), 571-587.

Katz, M.L., and Shapiro, C. "Technology Adoption in the Presence of Network Externalities," *Journal of Political Economy* (94:4) 1986, pp 822-841.

Kirsch, L. (2004) Deploying common systems globally: the dynamics of control, *Information Systems Research*, Vol. 15, No. 4, pp. 374-395.

Klein, H.K., and Myers, M.D. (1999) A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems, *MIS Quarterly*, Vol. 23, No. 1, pp.67-93.

Kuk, G. (2006) Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List," *Management Science*, (52:7), pp 1031-1042

Lakhani, K. and Wolf, B. (2005) Motivation and Effort in Free/Open Source Software Projects: The Interplay of Intrinsic and Extrinsic Motivations, in Feller, J, Fitzgerald, B, Hissam, S, and Lakhani, K. (2005) (Eds) *Perspectives on Free and Open Source Software*, MIT Press, Cambridge.

Lee, AS and Baskerville, RL (2003). Generalizing Generalizability in Information Systems Research. *Information Systems Research*, 14 (3), 221-243

Lundell, B, Lings, B and Lindqvist, E (2006) Perceptions and uptake of open source in Swedish organizations, in Damiani, E, Fitzgerald, B, Scacchi, W and Succi, G (2006) *Open Source Systems*, Springer-Verlag, New York, pp. 155-164.

Markus, M.L. (1987). Toward a 'Critical Mass' Theory of Interactive Media: Universal Access,

Interdependence and Diffusion, *Communications Research*, Vol. 14, pp. 491-511.

Markus, M.L., and Robey, D. (1988). Information Technology and Organizational Change: Causal Structure in Theory and Research, *Management Science*, Vol. 34, No. 5, pp. 583-598.

McCue, (2004) London council ditches Linux plans, http://news.zdnet.co.uk/software/0,1000000121,39118909,00.htm, last accessed on 8 Jan 2007.

Miles, M. and Huberman, A. (1994) *Qualitative Data Analysis: A Sourcebook of New Methods*, Sage, Beverley Hills.

Mohr, L.B. (1982). *Explaining Organizational Behavior*, San Francisco, CA: Jossey-Bass.

Moore, G.C., and Benbasat, I. (1991) Development of an Instrument to Measure Perceptions of Adapting an Information Technology Innovation, *Information Systems Research*, Vol. 2, No. 3, pp. 192-222.

Newman, M., and Robey, D. (1992) A Social-Process Model of User-Analyst Relationships, *MIS Quarterly*, Vol. 16, No. 2, pp. 249-265.

Niccolai, J (2005) Scottish police pick Windows in software line-up, *InfoWorld*, http://www.infoworld.com/article/05/08/11/HNscottishpolice_1.html, last accessed 20 Dec 2006

Onetti, A & Capobianco, F (2005) Open source and business model innovation. the Funambol case, in Scotto, M and Succi, G (Eds) *Proceedings of First International Conference on Open Source (OSS2005)*, Genoa, 11-15 July 205, pp. 224-227.

Rogers, E. (1962) *Diffusion of Innovations*, The Free Press, NY.

Rogers, E. (2003) *Diffusion of Innovations* (5th ed), The Free Press, NY.

Rossi, B, Russo, B and Succi, G (2006) A study of the introduction of OSS in public administration, in Damiani, E, Fitzgerald, B, Scacchi, W and Succi, G (2006) *Open Source Systems*, Springer-Verlag, New York, pp. 165-172.

Sambamurthy, V., and Poole, M.S. (1992) The Effects of Variations in Capabilities of GDSS Designs on Management of Cognitive Conflict in Groups, *Information Systems Research*, Vol. 3, No. 3, pp. 225-251.

Sanders, J. (1998) Linux, open source, and software's future, *IEEE Software*, Vol. 15, No. 5, pp. 88-91.

Shaikh, M (2006) Version Control Software in the Open Source Process: A Performative View of learning and Organizing in the Linux Collectif, Unpublished Thesis, London School of Economics.

Shaw, T., and Jarvenpaa, S.L. (1997). Process Models in Information Systems, *IFIP WG8.2 Working Conference on Information Systems and Qualitative Research*, May 31-June 3, Philadelphia, PA.

Stewart, K and Gosain, S (2006) The impact of ideology on effectiveness in open source software development teams, *MIS Quarterly*, Vol 30, No 2, pp. 291-314.

Thurston, R. (2006), Criticism mounts over Birmingham's Linux project, last accessed on 15 Jan 2007 at: http://www.zdnet.com.au/news/software/soa/Criticism_mounts_over_Birmingham_s_Linux_project/0,130061733,339272293,00.htm

Tornatzky, L. and Klein, K. (1982) Innovation Characteristics & Innovation Adoption Implementation: A Meta-Analysis of Findings, *IEEE Transactions on Engineering Management*, Vol. EM-29, pp. 28-45.

Turner, A (2005), Linux misses Windows of opportunity, last accessed on 15 Jan 2007 at http://www.theage.com.au/articles/2005/09/26/1127586780339html?from=top5

Van Reijswoud, V (2005) OSS for development: myth or reality? last accessed 17 Jan 2007 at http://www.calibre.ie/events/limerick/docs/calibre_Reijswoud_presentation.pdf

Ven, K, Van Nuffel, D and Verelst, J (2006) The introduction of OpenOffice.org on the Brussels Public Administration, in Damiani, E, Fitzgerald, B, Scacchi, W and Succi, G (2006) *Open Source Systems*, Springer-Verlag, New York, pp. 123-134.

Walli, S, Gynn, D and von Rotz, B (2005) The Growth of Open Source Software in Organizations, available at http://www.optaros.com/en/publications/white_papers_reports (last accessed 31 Jul 2006)

Walsham, G. (1993) *Interpreting Information Systems in Organizations*, Wiley, UK.

Yin, R. (1994) *Case Study Research: Design and Methods*, 2nd Ed, Sage Publications, California.

Zachary, G (2003) Ghana, Information Technology and Development in Africa, http://www.cspo.org/products/articles/BlackStar.PDF (Current 11 Aug 2006)

Zaltman, G., Duncan, R., & Holbeck, J. (1973), *Innovations & Organizations*, New York: Wiley & Sons.

Zuliani, P and Succi, G (2004) Migrating public administrations to open source software, *Proceedings of e-Society IADIS International Conference*, Avila, Spain, 2004.

This work was previously published in the International Journal of Open Source Software & Processes, edited by S. Koch, Volume 1, Issue 1, pp. 1-23, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 4.19
Media Centric Knowledge Sharing on the Web 2.0

Marc Spaniol
Max Planck Institute for Computer Science, Germany

Ralf Klamma
RWTH Aachen University, Germany

Yiwei Cao
RWTH Aachen University, Germany

ABSTRACT

The success of knowledge sharing heavily depends on the capabilities of an information system to reproduce the ongoing discourses within a community. In order to illustrate the artifacts of a discourse as authentic as possible it is not sufficient to store the plain information, but also to reflect the context they have been used in. An ideal representation to do so is non-linear storytelling. The Web 2.0 in its "bi-directional" design therefore is an ideal basis for media centric knowledge sharing. In this article we present a novel solution to this issue by non-linear storytelling in the Virtual Campfire system. Virtual Campfire is a social software that allows a modular composition of web services based on a Lightweight Application Server in community engine called LAS. Hence, Virtual Campfire is capable of fully exploiting the features of the Web 2.0 in a comprehensive community information system covering web-services for geo-spatial content sharing, multimedia tagging and collaborative authoring of hypermedia artifacts.

INTRODUCTION

The development of information systems for communities of practice (Lave & Wenger 1991; Nonaka & Takeuchi 1995; Wenger 1998) in different application domains is a challenging issue for several reasons. Principles like legitimate peripheral participation, group knowledge, situated learning, informality and co-location have to be taken seriously in the design of the community engine. For

that reason, the community engine has to reflect the social learning processes taking place, which differ from community to community. Even more, the information systems need a careful design of the digital media and the related communication/collaboration tools in order to reflect the discursive hypermedia knowledge contained in text, pictures, videos etc. Furthermore, communities are usually not able to express their needs in the very beginning of information system usage. Thus, the communities have to gain experiences "on their own" while applying the technologies in use. In addition, multimedia technologies and the Web 2.0 are rapidly developing, thus creating new requirements on hardware and software. In combination with a trend for multidisciplinary work and research novel approaches for flexible, evolving, adaptable, and interoperable community engines are required. Social software for technology enhanced learning therefore need to reflect the nature of the underlying community processes and their discourses. Consequently, the question is: How to design and orchestrate community information systems in order to fully exploit the features of the Web 2.0?

In order to meet these requirements we have developed in recent years a Lightweight Application Server [LAS] for community information system, which is capable of supporting communities by multimedia services on the basis of the multimedia content description interface MPEG-7. On top of it, Virtual Campfire is a community information system that allows a modular composition of web services for media centric knowledge sharing on the Web 2.0.

In this paper we first introduce a theoretical framework for working and learning in media-supported communities of practice. After that, we introduce concepts of knowledge sharing on the Web 2.0 and explain how these technologies help to create, manage and share knowledge in communities. Then we present Virtual Campfire and its core modules in a scenario of non-linear multimedia storytelling. Here, our social software is applied in a community of professionals for cultural heritage management. The paper closes with a summary and an outlook on further research.

A MEDIA CENTRIC KNOWLEDGE MANAGEMENT THEORY

Snow differentiates between two different trends in collaboration and learning within scientific communities (Snow 1959). First, the 'linear type' of learning that is goal-oriented and transmission-centered. This means, old information in scientific communities is being replaced by new one as soon as this appears. Second, there is a 'non-linear type' of learning. This type is media centric and reflects the nature of the ongoing discourse. It doesn't replace old information but keeps it and might be applied in a different context later on. Here, information is not simply transmitted for learning, but it is presented based on the underlying theory in use. Our collaborative research center on "Media and Cultural Communication" (cf. http://www.fk-427.de) has given us a detailed insight into the importance of proper media support in knowledge sharing. The description and [loose] classification of medial artifacts is probably the most important part of the methodological perception process to make social software work. This means that a continuous perception of activities in communities of practice is necessary for them in order to gain new knowledge. The question therefore is: How to resemble working practices in communities of practice by means of social software?

A media-specific theory developed in the center helped us understand digital media support for discourses in the cultural sciences. It is based on the following three media operations (Jäger & Stanitzek 2002; Fohrmann & Schüttpelz 2004):

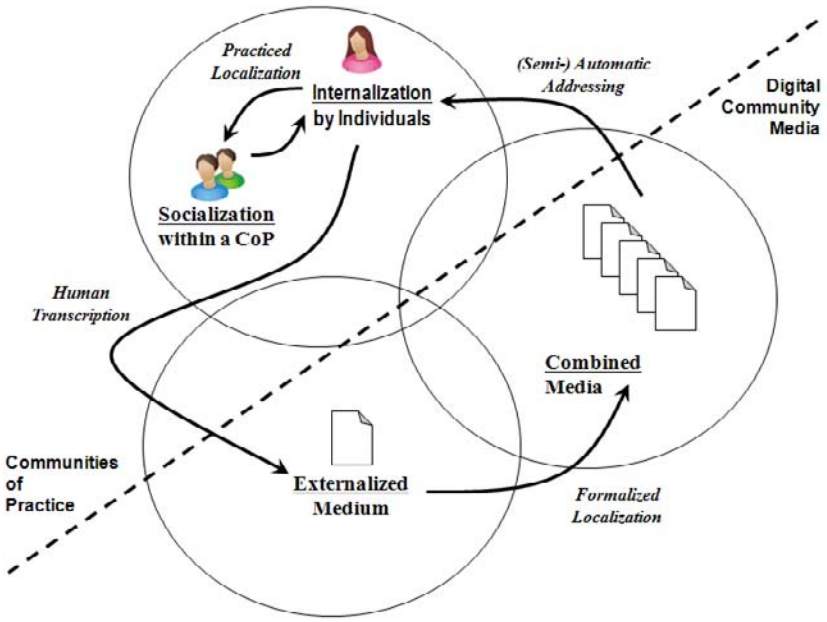
- *Transcription* is a media dependent operation to make media collections more readable.

- *Localization* means an operation to transfer global media into local practices. We distinguish between *formalized localization* within information systems [in digital community media] and *practiced localization* [in communities of practice] among humans.
- The term of *[re-] addressing* describes an operation that stabilizes and optimizes the accessibility in global communication.

In the following, we will now synthesize these media specific operations with learning processes of communities of practice. The result is a media centric re-formulation of the previously introduced media operations on knowledge creation and social learning processes adopted from Nonaka and Takeuchi (Nonaka & Takeuchi 1995) and Wenger (Wenger 1998).

Figure 1 brings together both approaches in media centric theory of learning and in communities of practice. It combines the two types of knowledge, tacit and explicit knowledge (Nonaka & Takeuchi 1995; Polanyi 1985), and the process within knowledge creation and learning processes with the media theory developed in our collaborative research center including its media specific operations [*transcription, formalized* as well as *practiced localization*, and *[re-] addressing*]. In the upper section we focus on actions performed by humans. Starting with an individual who has internalized some media-specific knowledge there are two ways to communicate with others. On the one hand, there is an option to present this information to others by human-human interaction in *practiced localization*, which allows the content's socialization within the communities of practice and vice versa which is equivalent to the development of a shared history. On the other hand, individuals may also perform a *human transcription* of their knowledge by generating new medial artifacts. This operation brings us into the lower section where digital community

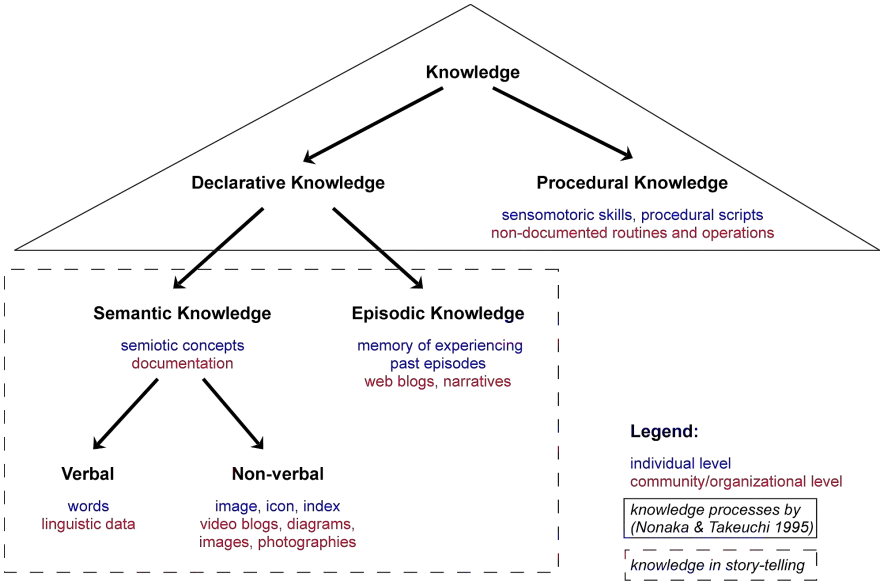
Figure 1. Media centric theory of learning in communities of practice



media are processed. The externalized artifacts of an individual are now further processed by the information system. This is done by *formalized localization* of the medial artifacts. In contrast to its high-level transcription by an individual, here, a technical computer supported recombination of medial artifacts takes place. As a result, the set of medial artifacts from various data types are combined within the information system. The final *[semi-] automatic addressing* closing the circle is the context depending presentation of the medial artifacts or a cross-medial concatenation. From then on, the process might be repeated infinitely oscillating between tacit and explicit knowledge on the epistemological axis and between individuals and the communities of practice on the ontological axis.

In order to make knowledge sharing a success for any kind of community of practice, independent of size or domain of interest, a generic community engine for social software is needed. That is exactly the point where social software is being applied in order to support the *formalized localization* process. While the previous media centric theory is based on the distinction between tacit or procedural and explicit or declarative knowledge, the importance of storytelling becomes visible after a further distinction between semantic and episodic knowledge (Tulving 1978; Ullman 2004) has been undertaken. In Figure 2 we depict a hierarchy of knowledge types with examples on the individual and community level. While semantic knowledge represents semiotic and conceptual knowledge such as documentation in organizational charts, business process definitions and so forth, episodic knowledge is knowledge about experiences such as episodes and narratives, e.g. war stories. This distinction is also being debated. Nevertheless, our claim is that a combination of semantic and episodic knowledge can be used more effectively in organizations. While situational context may be lost by external-

Figure 2. Individual and community/organizational levels of knowledge processing [Adapted from (Nonaka & Takeuchi 1995) with refinements on declarative knowledge done by (Ullman 2004)]



izing stories, outreach and impact of stories may be enhanced by this process. Documentation as a means of semantic knowledge can further be classified as verbal [linguistic data] and non-verbal [e.g. visual image, video, and diagram].

The Digital Storytelling Association [DSA] defines storytelling as follows: "Digital Storytelling uses digital media to create media-rich stories to tell, share and to preserve. Digital stories derive their power through weaving images, music, narrative and voice together, thereby giving deep dimension and vivid color to characters, situations, and insights" (DSA 2002). This illustrates that storytelling can be used not only for entertainment, but also for sharing knowledge. It intertwines semantic knowledge, i.e. already reified concepts of communities stored as documents, by linking it with the narrative experiences gained from episodic knowledge (Tulving 1978). Thus, storytelling can be seen as an approach to developing learning histories (Roth & Kleiner 1999) by creating knowledge hyper stories (Royrvik & Bygdas 2002). Consequently, storytelling is an important aspect for knowledge sharing and learning in communities of practice (Wenger 1998). Therefore, telling, sharing and experiencing stories are common ways to knowledge creation in communities of practice.

However, in the Web 2.0 (O'Reilly 2005) the power of storytelling has not been fully explored, yet. Particularly the opportunities of social software in combining contextualized knowledge with multimedia support in stories, is thus far marginally exploited. For that purpose we will introduce the key concepts of the Web 2.0 in the following.

KNOWLEDGE SHARING ON THE WEB 2.0 – THE POWER OF STORIES

While the Internet in general and the especially the web is assumed to be one of the really big media revolutions like the invention of book printing by Gutenberg in the 15th century, the wheel is still spinning. After only roughly 15 years of existence, the now so-called Web 1.0 is replaced by the Web 2.0, a term coined by Tim O'Reilly. Projects like Wikipedia let users become knowledge prosumers [consumer and producer "in parallel"] of wikis, replacing old-fashioned content management systems in organizations. Interoperability between content and services is realized by syndications tools [RSS]. In order to highlight the differences between the "new" and the "old" Web paradigms, we introduce the core Web 2.0 knowledge management presented in O'Reilly's seminal article (O'Reilly 2005). We will not repeat all features of the Web 2.0 here, but put forward the power of stories for knowledge sharing on the Web 2.0.

Participation instead of publishing. "Data are the new Intel Inside" is one of the slogans in the Web 2.0 (O'Reilly 2005), but communities get even more focus than data itself does. From del.icio.us, flickr, youTube, to mercora which are collections for bookmarks, photos, videos and music, the communities can subscribe to or rank the bookmarks' masters, photographers, video makers or DJs. Thus, social networks are built up to get a bundle of information instead of a single piece of information. In the information world, one can get whatever information he wants, when he finds the right community. Social navigation, social recommendation and social filtering techniques are even more important in a multimedia web, while classical information retrieval techniques deliver only limited support. Remix (cf. http://www.manovich.net/IA/index.html) of existing content is a technique which can easily be applied on existing Web 2.0 multimedia repositories like flickr and youTube. In the fact, digital storytelling can only happen within communities of story tellers and the audience. Storytelling as a community activity is based on participation, not necessarily similar to classical storytelling approaches but more to a many-to-many approach.

Syndication instead of stickiness. Lately, Web 2.0 is also featured as a kind of attitude with which people handle the web. More and more web sites support RSS instead of placing a button labeled with "Set this page to your home page". It has become natural and a kind of fashion to integrate third-party web services like google, yahoo and del.icio.us etc. Web services and syndication will be even more important in ubiquitous contexts when learners need support based on their location, their connectivity, their device capabilities and their usage context. Content has to be adapted to the various unreliable or unpredictable contexts of the learners instead of delivering the same content to every learner in every situation. For storytelling this implies more possibilities but also more care about media sets for stories. We already know that the narratives for mobile TV are much shorter than narratives in home TV where story parts e.g. in telenovelas are told in half hour rhythms. But based on adaptation strategies we have to prepare different media sets and even different narratives (Franz & Nischelwitzer 2004) for telling the same story in a different context.

Wikis instead of content management. Control freaks are worrying about the principal openness of the new social software applications like many wikis, but it turned out that participation is increased just by those low barriers. Even if inappropriate content is uploaded or created or existing content is modified or even vandalized, the communities have some self-repair strategies in place which are more flexible than any technological approach to protecting content. When these repair strategies do not work anymore, the community may have a problem by itself and will disappear eventually. This openness accepts various types of multimedia. It is possible to reuse multimedia, which takes advantage of well-rated community media. Open standards are widely employed but also here simple standards are preferred over complex ones. Storytelling in the Web 2.0 will depend on community strategies to maintain their media. Stories will evolve over time, but even the disruption of stories by other users has to be dealt with by the communities in the Web 2.0.

Folksonomies instead of taxonomies. Web 2.0 often uses tagging technologies to categorize multimedia content. For multimedia content-based retrieval techniques are of limited use, since they only work efficiently enough on a limited amount of materials, and even with large collections, only a limited number of retrieved materials has a high ratio in terms of precision and recall. Tags can inform multimedia retrieval and vice versa. Even if users misuse tagging and create false or misleading tags for multimedia content, content based retrieval techniques can be used to validate retrieval results based on simple keyword search. Multimedia and Web 2.0 technologies have to converge for storytelling on the Web 2.0. We need tagging technologies for retrieving interesting stories but we also need some kinds of emergent semantics for multimedia stories which are based on the content.

Contextualized Storytelling. Digital media allows fast creation, sharing and consumption of interactive content. What makes digital media most suitable for storytelling is the ability to recombine various media types, making stories more effective and interactive. Web-based systems are the ultimate step in the evolution of storytelling by making interactive multimedia contents not only available 24/7, but also allowing community-wide distribution.

As demonstrated in the concepts above, the Web 2.0 allows more and more knowledge to be created, managed and shared by communities themselves. Because Web 2.0 technologies were not intended specifically for digital storytelling, many challenges are raised by the readiness of communities to accept these technologies. However, already on the "classical" Web the power of storytelling was well recognized. There exist a lot of virtual communities like Fray.com (cf. http://fray.com/is/) whose content is solely

built of personal stories shared by the community members. Software like MemoryMiner (cf. http://www.memoryminer.com/) is capable of facilitating the authoring of digital stories even for non-experienced computer users. In oral history research web archives like the shoa archive (cf. http://www.shoahproject.org/) or the Densho project (cf. http://www.densho.org/) were created to preserve generational knowledge. In the context of corporate learning storytelling is also known and used (Nonaka & Takeuchi 1995; Davenport & Prusak 1998, Brown & Duguid 2000). A very comprehensive collection of resources is available on the Internet (cf. http://tech-head.com/dstory.htm). For the sake of clarity, we will now introduce and compare existing storytelling and analyze their capabilities for knowledge sharing on the Web 2.0.

Storytelling Environments Reviewed

There are many software tools and models for storytelling available on the Web. However, most of them are commercial products and do not incorporate any Web 2.0 methodologies. Even more, most of them aim at creation of fiction instead of sharing knowledge. Our overview will focus on these systems that are most suitable for sharing semantic and episodic knowledge in communities.

Dramatica is a comprehensive framework suitable to create multimedia stories (Phillips and Huntley 2001). However, it does not allow any kind of non-linearity. In Dramatica a story represents a particular model called the "story mind". It is left to the creativity of the authors to express their episodic knowledge as a linear story so that dedicated aspects of the story are filled with content. Dramatica is also capable of supporting semantic knowledge. Because of its mostly individual conception, Dramatica does not provide many Web 2.0 features.

Adaptive Digital Storytelling [Adaptive DST] is a computer-based form of narration that tries to integrate basic principles of narratives and dramatic art into interactive digital stories (Franz and Nischelwitzer 2004). Adaptive DST subdivides episodic knowledge into selected- and must-phases, and specifies their interdependencies. Another key concept in Adaptive DST is the option to manipulate the story a-priori. Here, a variation of a story can be generated based on pre-defined tags used to specify the level of information a user wants to obtain. Based on a 4-ary classification scheme users can select from a superficial to a fine-grained story adaptation. Thus, non-linearity is only supported to a certain extent. The existing "core story" might not be changed completely in its outcome, but might be altered depending on the user's interest in the topic. While the concept is applicable to knowledge sharing on scholarly level, in general, it is doubtful that such a labor-intense, and mostly unguided creation process might be applicable in a community, or at a larger scale. Even more, Adaptive DST does not support collaboration features for Web 2.0 technologies.

Storylining Suspense and **Story Engine** are closely related systems for the creation and consumption of non-linear multimedia stories. While Storylining Suspense is an approach to a new authoring method for interactive storytelling (Schneider et al. 2003), Story Engine is used to capture episodic knowledge by narrating interactive non-linear stories [e.g. created by Storylining Suspense] (Braun 2004). The focus in Storylining Suspense is on authoring of non-linear stories based on a set of morphological functions defined by Vladimir Propp (Propp 1958). These functions are mapped within the system based on a scene model thus creating variants of a story based on the underlying model and the user's interaction. Additionally, there are options to store semantic knowledge about multimedia contents, but it is left open, whether these contents are available only to support the creation process or will be accessible upon consumption as well. Thus, despite their client/server structure, Storylining Suspense

and Story Engine are suited only on a limited scale for multi-authoring and, consequently, the Web 2.0.

Hypermedia Novel [Hymn] is a new storytelling approach that extends the classical narration concept of Graphic Novel (Heiden et al. 2001). Hymn is a modular concept that allows the creation and consumption of hypermedia stories. The main concept of Hymn is the so-called narration module which can be accessed by an authoring tool. A narration module captures the episodic knowledge and stands for a scene within a story. These modules may be linked with other narration modules thus defining the story graph. Because of its openness in creating and sharing knowledge Hymn incorporates Web 2.0 methodologies. Despite its clear graph oriented narration structure, Hymn does not seem to apply any theoretical concepts. The "Hymnplayer" is a conventional web-browser using the Java Media Framework. Here, different media might be visualized but there is currently no support to store and retrieve semantic knowledge within media related metadata.

Table 1. Storytelling environments and methodologies compared

		Dramatica	Adaptive Digital Storytelling	Storylining Suspense & Story Engine	Hypermedia Novel	Digital Storytelling Cookbook and Travelling Companion [DSC]	Movement Oriented Design [MOD]	Storytelling in Virtual Campfire	
Story concept		n.a.	Must & should dependencies	Morphological functions	Extended Graphic Novel	Community "heuristics"	Motivation, Exigency and Structure	Motivation, Exigency and Structure	*Adopted from MOD*
Semantic Knowledge		Verbal & non verbal	Available	Available	n.a.	n.a.	Verbal & non verbal	Verbal & non verbal	
Episodic Knowledge		???	Linear	Linear & non-linear	Linear & non-linear	n.a.	Linear & non-linear	Linear & non-linear	
Product Type		Commercial	Viewer: Public Editor: Commercial / Proprietary	???	Viewer: Public Editor: ???	Not implemented	Not implemented	Research	
Validation		Advices only	n.a.	Automatic consistency checks	n.a.	Not implemented	Not implemented	Automatic validation of MOD compliance	
Web 2.0 features	Creation	Individual	Individual	Individual	Community wide	Not implemented	Not implemented	Community wide	
	Sharing	n.a.	Proprietary Webserver	Integrated Story Engine	Proprietary Webserver	Not implemented	Not implemented	Affiliated Community Webserver	

The **Digital Storytelling Cookbook and Travelling Companion** [DSC] is considered to be a handbook for the creation of digital stories based on the "heuristics" gathered in a community of users associated with the center for digital storytelling (Lambert 2003; Center for Digital Storytelling 2005). For that purpose, the DSC breaks down episodic knowledge of digital stories into subcomponents and gives practical advices how to make stories out of user experiences. Besides some practical advices on how to find ideas about stories there are seven theoretical elements specified which should be fulfilled in a good story. However, there are no concepts described suitable to process media related semantic knowledge. On the technical level, the DSC only gives hints on how to use proprietary software. A common technical platform to create and share stories has not yet been developed. For that reason, DSC can not be considered as Web 2.0 software. In general, the DSC is suitable to support digital storytelling in various areas of application without going into details.

The **Movement Oriented Design** [MOD] paradigm (Sharda 2005) is a new methodology for the creation of linear and non-linear multimedia stories. Its core idea is to bring different theories, models and tools under one roof. Thus, it integrates features from Dramatica (Phillips and Huntley 2005) as well as the Aristotelean Poetic (Aristotels 2000). The result is a novel methodology and formalism in order to create multimedia stories by combining three facets of stories: Motivation [verbal and non-verbal knowledge], Exigency [semantic knowledge] and Structure [episodic knowledge]. Thus, the MOD methodology is a comprehensive framework for the creation of non-linear digital stories. However, a prototypical implementation is missing yet. Consequently, MOD can not be considered as a Web 2.0 implementation.

In the previous subsections we have introduced several implementations and methodologies applied in the area of storytelling. As we have pointed out, current approaches are not suitable to combine Web 2.0 features with a comprehensive methodological concept to process **semantic** and **episodic knowledge**. Table 1 gives a condensed overview on these approaches by highlighting their key features. For the sake of a comparison among these features, our approach of a Storytelling in Virtual Campfire, which contains the theoretical concepts of MOD [cf. (Spaniol et al. 2006) for details], is included. Thus, Virtual Campfire is a social software that allows a modular composition of web services for media centric knowledge sharing. We will now introduce the core modules in a scenario of non-linear multimedia storytelling applied in a community of professionals for cultural heritage management.

VIRTUAL CAMPFIRE – SOCIAL SOFTWARE APPLIED

Despite the huge number of social software on the web, users face many problems when trying to apply these technologies for more sophisticated means of knowledge sharing than the simple tagging of pictures (e.g. http://flickr.com/) or exchange of bookmarks (e.g. http://del.icio.us/). However, what is really needed is to orchestrate services like these in an arbitrary manner. Therefore, our approach here is to go one step further by making multimedia contents available to others by interoperable multimedia metadata standards like MPEG-7 (ISO 2003; Kosch 2003). In order to allow community members browsing multimedia artifacts, collections and thus any kind of hypermedia, we use MPEG-7 for the capturing of explicit knowledge. For the purpose of combining interoperability and server side computations, Virtual Campfire is based on Lightweight Application Server [LAS] for MPEG-7 Services in community engines. Thus, we will first introduce the basic concepts of LAS before describing the core services applied in Virtual Campfire.

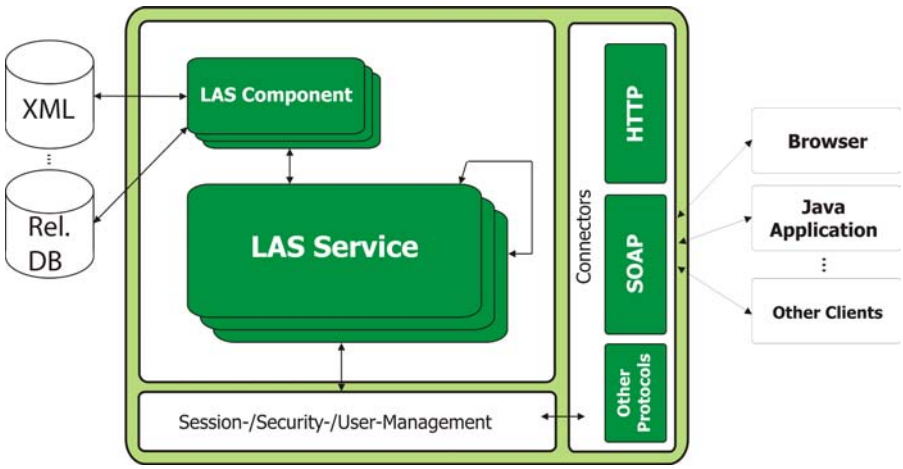
LAS: A Lightweight Application Server for Social Software

LAS is a platform independent Java implementation of a lightweight middleware platform for service oriented architectures [SOA] developed at our chair for the purpose of providing network services which can be shared among various tools supporting the work of communities in practice. The LAS Java API and its concepts are used to build the server's functionality and thus allow arbitrary server extensions by three basic element types: Connectors, components and services. Figure 3 shows a simplified diagram of the LAS architecture and the interrelations between server elements described in the following. A connector realizes the server side for client-server communication using a particular protocol, e.g. HTTP or SOAP. Components encapsulate functionality for common tasks shared by services or other components. Services define the actual functionality that LAS offers to its clients. Public service methods are available to clients through one of the connectors inside a session context. Service methods can be invoked by clients using a connector client for any of the available communication protocols [HTTP/SOAP]. Access to service methods as well as to arbitrary secured objects is controlled on server side by an internal security management that is based on users, groups and roles. Access rights can be defined on different levels of granularity, i.e. per-service, per-service-method or per-method-signature. Therefore, a client simply connects to the LAS using one of the available connector clients and invokes LAS service methods remotely, possibly involving secured objects.

Semantic Zapping Services: Exploration of Multimedia Contents

Semantic Zapping Services in Virtual Campfire tries to bridge the gap between "folksonomy-style" high-level semantic knowledge about multimedia and purely technical low-level content descriptions. These services are intended to support collaboration in communities by the exchange of multimedia contents and their low and high-level semantic descriptions. In order to ensure interoperability among the contents described multimedia

Figure 3. Simplified LAS architecture



metadata standards are being incorporated. In this aspect, the Dublin Core [DC] metadata standard (Dublin Core Metadata Initiative 1999) has been a step forward, as it is an easy to understand and concise method for media annotations. Nevertheless, DC is not suitable for temporal and media specific annotations of multimedia contents. For that reason, we try to overcome these limitations by a combination of the loose classifications in DC with more sophisticated description elements for time based media in MPEG-7. Thus, our Semantic Zapping Services are based on an excerpt of the extensive MPEG-7 multimedia metadata standard. Even more, we provide services for a semi-automatic conversion from DC to MPEG-7 while an affiliated FTP server is used for an automated up- and download of multimedia artifacts by the community to the common repository. Consequently, the Semantic Zapping Services of Virtual Campfire allow the community members to search and browse for multimedia contents described by MPEG-7.

Collaboration Services: Authoring of Multimedia Contents

Externalization of knowledge in Virtual Campfire is supported by annotating, tagging and sharing multimedia contents within the community. Contrary to a conventional categorization system, multiple concepts are used for one piece of information. Other than Flickr.com [cf. http://flickr.com) all metadata generated in Virtual Campfire is MPEG-7 compliant and generated using MPEG-7 LAS service methods. For that purpose, Virtual Campfire offers two types of tagging: keyword- and semantic tagging (cf. figure 4). Keyword tagging enables users to assign a set of plain keywords to an image like it can be done in Flickr. From the technical point of view,

Figure 4. Keyword and semantic tagging of multimedia contents

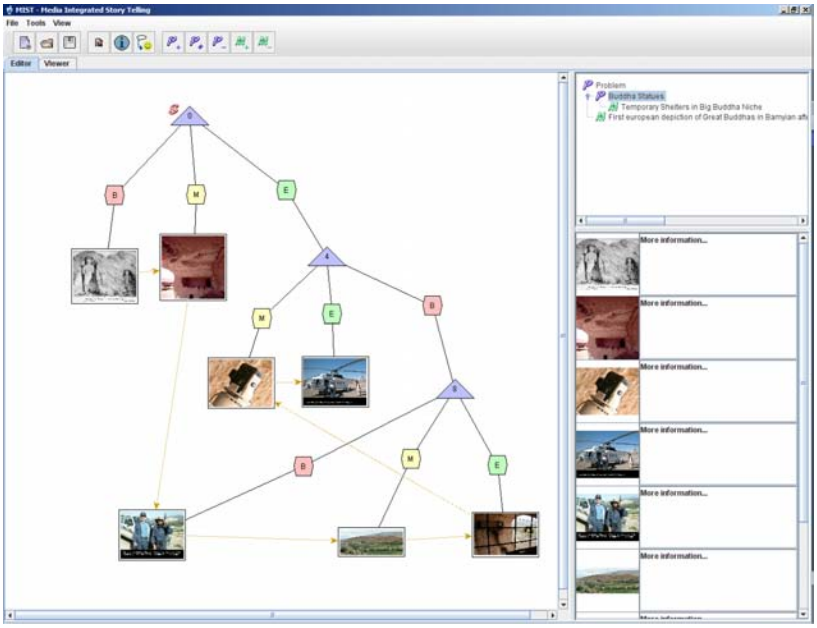
keyword tagging is covered by the methods of MultimediaContentService. Semantic tagging goes a step further by allowing users to define semantic entities and to assign semantic entity references to an image. These are more expressive than plain keywords, because they carry additional semantics. For example one could not derive from a plaintext keyword "Buddha", that it describes an agent, while for semantic tagging, "Buddha" has been modeled as a semantic entity of type agent. Semantic entities are defined using methods of SemanticBasetypeService. Semantic references are assigned to images using methods of MultimediaContentService.

Similarly, retrieval is based on the multimedia descriptions. For retrieval by plain keyword tags users can formulate keyword search expressions as propositional logic formulae using keywords as atomic propositions. For example, the keyword search expression "Buddha and Bamiyan and not(Destroyed or Taliban)"would retrieve all images having been assigned the keywords Buddha and Bamiyan, but none of the two keywords Destroyed or Taliban. The above concepts are transferable to the more expressive semantic tagging, which can be easily realized on the basis of the MPEG-7 services.

Storytelling Services: Re-Contextualization of Episodic Knowledge

For re-contextualization of episodic knowledge Virtual Campfire provides dedicated Storytelling Services. In order to help even an "untrained" user in creating useful stories (from a structural point of view), the MOD paradigm is being applied as a theoretical basis (cf. Section 3). For that purpose two dedicated user interfaces are available for the Storytelling Services: An editor and a player. The editor allows users to create new or edit already existing multimedia stories. The player is used for

Figure 5. A non-linear story created in Virtual Campfire



the consumption of existing multimedia stories. Besides the explicit knowledge contained in the multimedia contents themselves the high-level semantic tags are accessible. These contents can thereafter be temporally arranged as they depend on a certain context they belong to. When creating a story the author can now create paths covering different problematic aspects along the contents. Thus, the problems addressed depend on the path selected and lead consequently to different results in a story. Figure 5 shows the editor consisting of three main elements [from left to right]: Storyboard, plot and semantic annotations. The plot in the middle represents the declarative knowledge captured in a story. It is rendered as a tree hierarchy, which allows the further decomposition into sub-problems. In addition, problems addressed in a multimedia story can be linked to related multimedia contents. The storyboard on the left hand side shows a visualization of episodic knowledge as paths between content elements. In addition, the decomposition of stories according to MOD paradigm into begin (B), middle (M), and end (E) is shown. Finally, on the right hand side, additional semantic annotations can be added to any multimedia element. Thus, users may express verbal-knowledge being associated with non-verbal knowledge.

CONCLUSION AND OUTLOOK

With the further development of the Web 2.0 and social software (in particular) digital storytelling becomes a central knowledge sharing and learning technology again. Especially, in situations where direct interaction is not possible, the new social software application offers the possibility to share multimedia materials in a community-centered style. Like the creation of new knowledge is a discursive and multistage process, the user requirements are rapidly changing and several new features need to be integrated into community information systems. In contrast to existing implementations the methodology and architecture Virtual Campfire is more flexible to assess the community needs over time and to integrate the community members in the development process. Even more, the multimedia services of Virtual Campfire based on MPEG-7 provide interoperability and exchangeability of learning contents. Thus, the usage of LAS simplifies the community support process for the communities of practice drastically and on the same time offering more influence on the development process. However, the direct support of computer scientists and community designers is still needed. In future, graphical editing support for community web sites could leave even more responsibility on the community side.

Another topic of ongoing research is the assessment and analysis of stories. Since the stories are created by communities we need community-centered assessment tools beyond the level of simple rating tools. Our stories are related to a problem-solving space created by a hierarchical presentation of problems. Non-linear digital stories components have to cover at least all sub-problems in the problem space. This can be tested automatically by some algorithmic approach while the emotional movement of learners and their problem-solving skills are much harder to test. This is clearly interdisciplinary research which can be performed in an organizational or psychological framework.

ACKNOWLEDGMENT

This work was supported by German National Science Foundation (DFG) within the collaborative research centers SFB/FK 427 “Media and Cultural Communication”, within the research cluster established under the excellence initiative of the German government “Ultra High-Speed Mobile Information and Communication (UMIC)” and by the 6th Framework IST programme of the EC through the Network of Excellence in Professional

Learning (PROLEARN) IST-2003-507310. We thank our colleagues Nalin Sharda and Georgios Toubekis for the inspiring discussions. In addition, we thank our students D. Renzel, H. Janßen, M. Pienkos, P. M. Cuong, D. Andrikopoulos and A. Hahne for the implementation of the Virtual Campfire services.

REFERENCES

Andrienko, N. and Andrienko, G. (2005). Exploratory Analysis of Spatial and Temporal Data -- A Systematic Approach. Springer.

Aristoteles (2000). "Poetics" (translated by S. H. Butcher). http://classics.mit.edu/Aristotle/poetics.html, {25.7.2006}.

Braun, N. (2004). "Kontrolliertes Erzählen von Geschichten mit integrierten, Videobasierten Hyperstories". In: R. Keil-Slawik, H. Selke, and G. Szwillus (eds.), Mensch & Computer 2004: Allgegenw¨artige Interaktion, Oldenbourg. pp. 157–167.

Brown, J.S. and Duguid, P. (2000). The Social Life of Information. Harvard Business School Press. Boston, MA.

Center for Digital Storytelling (2005). Homepage, http://www.storycenter.org/index1.html, {25.7.2006}.

Cox, S. and Daisey, P. et al. (eds.) (2005). OpenGIS Geography Markup Language (GML) Encoding Specification. Open Geospatial Consortium, Inc.

Davenport, T. and Prusak, L. (1998). Working Knowledge: How Organizations Manage What they Know, Cambridge, MA, Harvard Business School Press.

Digital Storytelling Organization (DSA). (2002). "Defining Digital Storytelling". http://www.dsaweb.org/01associate/ds.html {25.7.2006}.

Dublin Core Metadata Initiative (1999). "Dublin core metadata element set, version 1.1: Reference description". Technical report, Dublin Core Metadata Initiative. http://dublincore.org/documents/dces/ {25.7.2006}.

Franz, K. and Nischelwitzer, A. (2004). "Adaptive Digital Storytelling: A Concept for Narrative Structures and Digital Storytelling build on Basic Storytelling Principles, Adaptive Story Schemas and Structure Mapping Techniques". In L. Zimmermann (ed.): Multimedia Applications in Education Conference (MApEC) Proceedings. Graz. pp. 28-33.

Fohrmann, J. and Schüttpelz, E. (eds.) (2004). "Die Kommunikation der Medien". Niemeyer, Tübingen (in German).

Gröger, G., Kolbe, T. H. and Czerwinski, A. (eds.) (2006). Candidata OpenGIS CityGML Implementation (City Geography Markup Language). OGC 06-057r1. Open Geospatial Consortium, Inc.

Heiden, W., Frühling, C. and Deuer, H. (2001). "Hypermedia Novel - Hymn. A New Storytelling Paradigm". Proceedings of CAST '01. pp. 345-348.

ISO. (2002). "Information technoloy – Multimedia content description interface – Part 8: Extraction and use of MPEG-7 descriptions". Technical Report ISO/IEC TR 15938-8: 2002(E).

ISO (2003). "Information Technology – Multimedia Content Description Interface – part 5: Multimedia description schemes". Technical Report ISO/IEC TR 15938-5:2003.

Jäger, L. and Stanitzek, G. (eds.) (2002). "Transkribieren - Medien/Lektüre". Wilhelm Fink Verlag, Munich (in German).

Kosch, H. (2003). Distributed Multimedia Database Technologies Supported by MPEG-7 and MPEG-21. Auerbach Publication.

Kraak, M.-J. and Ormeling, F. (2003). Cartography. Pearson Education Limited, England.

Lambert, J. (ed.) (2003). "Digital Storytelling Cookbook and Travelling Companion". Digital Diner Press, 4.0 edition, (Excerpt).

Lave, J. and Wenger E. (1991). Situated Learning: Legimate Peripheral Participation. Cambridge University Press, Cambridge, UK.

Lynch, K. (1960). The Image of the City. The MIT Press.

Martínez, J.M., Gonzández, C., García, C. and de Ramón, J. (2002). "Towards Universal Access to Content using MPEG-7". Multimedia '02. December 1-6, Juan-les-Pins, France.

Nonaka, I. and Takeuchi, H. (1995). "The Knowledge-creating Company". In: Oxford University Press, Oxford.

O'Reilly, T. (2005). "What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software". http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/ what-is-web-20.html {3.7.2006}

Phillips, M. A. and Huntley, C. (2001). "Dramatica—A New Theory Of Story". Screenplay Systems Inc., 4th edition, 2.

Polanyi, M (1985). "Implizites Wissen". Suhrkamp, Frankfurt/Main (in German).

Propp, V. (1958). "Morphology of the Folktale". International Journal of American Linguistics, 24(4, Part II).

Roth, G. and Kleiner, A. (1999). "Car Launch: The Human Side of Managing Change". Oxford University Press, New York.

Royrvik, E.A. and Bygdas, A.L. (2002). "Knowledge Hyperstories — The Use of ICT Enhanced Storytelling in Organizations". 3rd European Conference on Organizational Knowledge, Learning and Capabilities. Athens, Greece. http://www.alba.edu.gr/OKLC2002/Proceedings/pdf files/ID260.pdf {2.10.2006}.

Schneider, O., Braun, N. and Habinger G. (2003). "Storylining suspense: An authoring environment for structuring non-linear interactive narratives". In WSCG, http://wscg.zcu.cz/wscg2003/Papers_2003/I53.pdf {25.7.2006}.

Sharda, N. (2005). "Movement Oriented Design: A New Paradigm for Multimedia Design". International Journal of Lateral Computing (IJLC), 1(1):7–14, 2005.

Snow, C. P. (1959). "The Two Cultures". Cambridge University Press, Cambridge.

Spaniol, M., Klama, R., Sharda, N. and Jarke, M. (2006). "Web-Based Learning with Non-Linear Multimedia Stories", 5th International Conference on Web-based Learning (ICWL 2006), July 19-21. Penang, Malaysia.

Tulving, E. (1978). "Episodic and semantic memory". In E. Tulving and W. Donaldson (Eds.), Organization of Memory, New York: Academic Press, pp. 381-403.

Ullman, M. T. (2004). "Contributions of memory circuits to language: the declarative/procedural model". Cognition, 92:231 – 270.

Wenger, E. (1998). "Communities of Practice: Learning, Meaning, and Identity". Cambridge University Press, Cambridge, UK.

This work was previously published in Knowledge Networks: The Social Software Perspective, edited by M. Lytras, R. Tennyson, & P. Ordonez de Pablos, pp. 46-60, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.20
Towards an Integrated Model of Knowledge Sharing in Software Development:
Insights from a Case Study

Karlheinz Kautz
Copenhagen Business School, Denmark

Annemette Kjærgaard
Copenhagen Business School, Denmark

ABSTRACT

This article adds to the discussion on knowledge management (KM) by focusing on the process of knowledge sharing as a vital part of KM. The article focuses on the relationship between knowledge, learning, communication, and participation in action, and the role of social interaction and technical media in the knowledge sharing process. We develop an initial theoretical framework of knowledge sharing on the basis of a literature study. Drawing on an empirical study of knowledge sharing in a software development company, we discuss what supports and what hinders knowledge sharing in software development. Finally, we use this knowledge to improve the theoretical framework.

INTRODUCTION

The KM literature is extensive, but the discussion on how to manage knowledge in organisations is far from over, and new proposals as well as lessons learned are continually being suggested. However, the published literature, especially in the information systems field, is largely grounded in a view that considers knowledge as an objective commodity which can be collected, represented symbolically and processed like information (Dahlbom & Mathiassen, 1993; Tsoukas, 1998). The literature consequently shows a certain preoccupation with information technology (IT) and technical solutions while it reflects a limited view of individual and organisational knowledge-related processes (Swan, Scarbrough, & Preston,

1999). The practice of KM is frequently reduced to the implementation of new IT-based systems, and important organisational aspects, in particular human and social issues, are overlooked. There are, however, exceptions in the literature which reviews KM success and critical success factors (Jennex & Olfman, 2005, 2006). Like Kautz and Thaysen (2001), those who emphasise the important but not privileged role of IT provide a balanced discussion of technical issues related to KM.

This article takes this debate into account and is based on a broader perspective of knowledge and KM. Our focus is on understanding especially the process of knowledge sharing as a vital part of KM and on the relationship between knowledge, communication, and participation in action through either social interaction or technical media in the knowledge sharing process.

By studying the knowledge sharing process in a Danish software development company, we provide an insight into how developers draw from organisational memory (Walsh & Ungson, 1991) to share knowledge in a learning context. We discuss the use of social interaction and technical media in the communication process and provide conclusions on how different forms of knowledge are shared through the two types of media.

Our focus is primarily on the role of people in the knowledge sharing process, but we also include empirical findings on how people use technology to share knowledge.

The article is structured as follows. The next section introduces the concepts of knowledge, learning and communication which inform our understanding of knowledge sharing. The third section presents our research approach and setting. Our empirical findings are described in the fourth section and discussed in the fifth section. Finally in the sixth section, we present our conclusions and the challenges for future research.

THEORETICAL BACKGROUND

We begin by exploring the concepts on which we build our initial theoretical understanding in order to present how we utilise them in this study. Knowledge sharing is a bilateral process in which knowledge is exchanged between individuals and groups (Comas & Sieber, 2001). Knowledge is the outcome of a complex process, a part of which is the gathering and processing of information. This has been described by Kolb (1984) and others as a learning process. Learning is significant for the attainment of knowledge, and thus also for the sharing of knowledge. Information is communicated among people with the aid of a shared language, body language, and actions (Fiske, 1990; Nielsen, 1994) and participation in action and practice builds the foundation for learning (Wenger, 1998). This happens through social interaction and in some cases with the aid of technical media (Thompson & Walsham, 2001). Communication and participation in action are thus also significant for the sharing of knowledge. In the following we revisit the concepts of knowledge, learning communication, and participation in action and their relationship and importance for knowledge sharing in more detail.

Knowledge

Many definitions of knowledge have been presented in the literature. Although they differ in scope and orientation, they seem to agree upon the fact that knowledge is a complex multifaceted concept which can be understood from different perspectives (Cook & Brown, 1999; Kautz & Thaysen, 2001). From a hermeneutic perspective, knowledge is not a commodity which can be collected under controlled conditions and bought or sold on a market (Dahlbom & Mathiassen, 1993). On the contrary, it is subjective enlightenment, a personal property which is grounded in human cognition of things and relations in the world

(Nielsen, 1994) and it affords actions (Cook & Brown, 1999).

Nonaka (1994) and Nonaka and Takeuchi (1995) build their model of the dynamics of knowledge creation with its conversion processes socialisation, externalisation, internalisation, and combination, which is a prerequisite for knowledge sharing according to Polanyi (1966), in particular under his concept of tacit knowledge. However, the commonly used sharp distinction between explicit knowledge which can be captured and codified into manuals, procedures, and rules and is easy to disseminate, and tacit knowledge, which cannot be easily articulated and thus only exists in people's hands and minds, and manifests itself through their action, has not been made by Polanyi (Stenmark, 2000). As Stenmark argues, Polanyi (1966) sees explicit and tacit knowledge as intrinsically interrelated and mutually constituted, and views tacit knowledge as the backdrop against which all understanding is distinguished. All knowledge thus has a tacit dimension, and tacit knowledge is the cultural, emotional, and cognitive background of which humans are only marginally aware. This is supported by Tsoukas (1996), who argues that explicit and tacit knowledge are inseparably linked, and that they cannot be treated as two separate types of knowledge.

Against this background, the mechanical conversion model as a background for knowledge creation and sharing appears to be an inadequate description of the underlying processes. We follow Cook and Brown (1999), who apply the concepts of explicit and tacit knowledge in the form originally intended by Polyani (1966). Acknowledging the codifiable nature of explicit knowledge which can be transmitted through formal, systematic language, and tacit knowledge, which is rooted in action and difficult to formalise and communicate, they do not promote one distinct kind of knowledge as an ideal for knowledge. Instead they emphasise the importance and necessity of different forms of knowledge. They distinguish between explicit and tacit knowledge, but also between individual and group knowledge, thereby creating four basic knowledge forms: (1) explicit individual, (2) explicit group, (3) tacit individual, and (4) tacit group knowledge. They see these four forms as distinct forms of knowledge all with equal standing. All four forms are relevant to describe and understand knowledge and knowledge sharing in organisations. In addition to the four knowledge forms, Cook and Brown introduce the concept of knowing, which is the part of an action or practice which deals with the way knowledge is used in interaction with the social and physical world. Sharing their broader perspective on knowledge, our theoretical framework includes three dimensions of knowledge based on Cook and Brown's (1999) work: explicit/tacit knowledge, individual/group knowledge and knowledge/knowing.

We thus understand explicit knowledge as knowledge which is codifiable and can be expressed directly through language. This means that explicit knowledge is structured and ordered such that it can be described and discussed through speech, scripts, drawings, and other signs and symbols. Tacit knowledge is knowledge which cannot directly be codified, but which can (only) be expressed indirectly through language and action. Tacit knowledge is a "feel" or sense for something or "to be able to do something without being able to explain why" (Brown & Gray, 1995, p. 78). Tacit knowledge is not hidden or inaccessible—tacit knowledge simply cannot be codified or expressed directly.

Individual knowledge is knowledge held by an individual that is applied in the individual's actions, while group knowledge is knowledge held by a group and applied in the group's actions. Group knowledge is created through the cooperation of the group's members and is part of the group's practice (Brown & Gray, 1995). Group knowledge includes knowledge about how the group works, its social rules, the group's memory about earlier actions, and knowledge about the group's tasks.

Not necessarily all members of the group possess the whole group's knowledge; it is the group as a whole that possesses the group's knowledge, and it is the group as a whole that applies this knowledge (Cook & Brown, 1999).

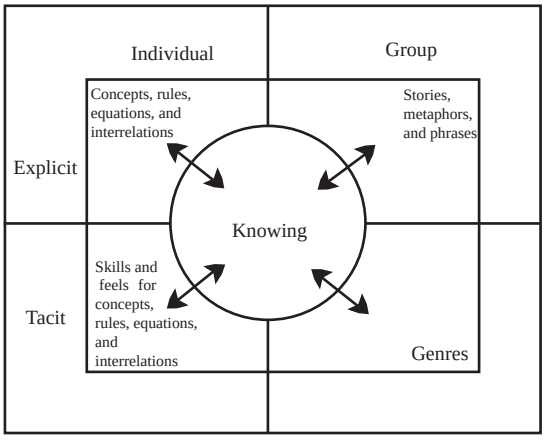
Each of the four types of knowledge presented by Cook and Brown (1999) is associated with a set of knowledge forms (see Figure 1). Explicit individual knowledge is expressed in concepts, rules, equations, and interrelations. This is, for example, knowledge in the form of concepts about the design of a product or a specific procedure to follow. Tacit individual knowledge can be expressed in a person's skills in applying tools or routines for the performance of a particular task (Comas & Sieber, 2001). Explicit group knowledge consists of shared stories about previous successes or failures and of metaphors used for establishing a common understanding of a problem or a task. And finally, tacit group knowledge is related to organisational culture in the form of genres which guide the thoughts and actions of organisational members.

The four types of knowledge can be seen as constituting organisational memory in different forms. According to Walsh and Ungson (1991), organisational memory is both an individual and organisational level construct. Although the definition of organisational memory is far from uniform, it is generally agreed upon that organisational memory consists of mental and structural artefacts which have a consequential affect on performance (Walsh & Ungson, 1991). In its most basic sense, organisational memory refers to stored information from an organisation's history which can be brought to bear on present decisions.

We have knowledge to carry out an action, but at the same time we can have knowledge independently of whether we carry out the action or not. Knowledge is something we possess irrespective of action. Thus knowledge is a tool which can be used in action, but which in itself is not action. The four forms of knowledge created by the dimensions of explicit—tacit and individual—group are thus not sufficient to describe the knowledge which is expressed through practice (Cook & Brown, 1999).

Every time someone carries out an action, a combination of many different types of knowledge is used. Each type of knowledge contributes to the action in a way in which the other types cannot. The

Figure 1. Types of knowledge related to knowing based on Cook & Brown (1999)



part of the action which focuses on the application and combination of explicit and tacit knowledge in interaction with the social and physical world is referred to as "knowing" by Cook and Brown (1999). As they put it, "Knowing is to interact with and honour the world using knowledge as a tool" (p. 389). Knowing has a focus on how we use our knowledge in practice. Knowing is not something we have, but something we do, and therefore a part of the actual action.

Figure 1 shows the different forms of knowledge related to the process of knowing.

Having argued for a three-dimensional understanding of knowledge, we now focus on the concept of learning and how this contributes to our understanding of the knowledge sharing process.

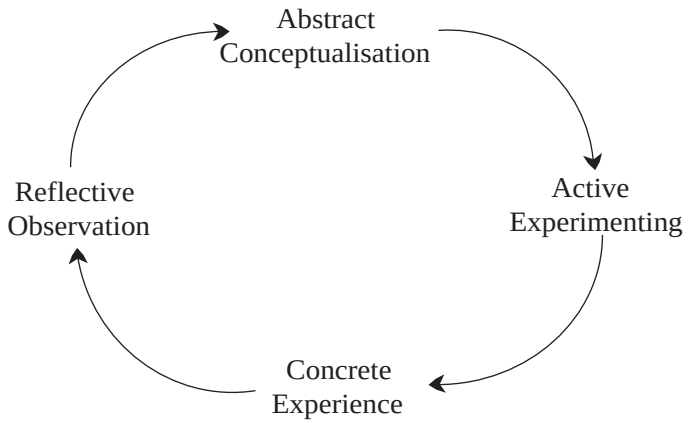
Learning: Knowing as a Learning Process

Learning is the process in which we acquire knowledge based on the communication of information and the participation in action. The learning process is thus an antecedent of having knowledge and is important for the sharing of knowledge.

Kolb (1984) has developed a theory of learning according to which learning is based on experience, and takes place through learning cycles. The learning process is a cognitive process in which individuals process new information or actions on the background of existing information or actions. In this process—depicted in Figure 2—information called abstract conceptualisation is obtained though speech and script, or indirectly through action and practice, when it is called concrete experience; thereafter information is processed actively through active experimenting or reflectively through reflection and/or observation. This means that abstract concepts and concrete experience are tested actively in practice or are reflected upon. The result of this process is new knowledge.

This learning theory can be related to Cook and Brown's (1999) understanding of knowledge and knowing (Comas & Sieber, 2001). A mutual interplay exists between knowledge and knowing: knowing means that we use our knowledge in new ways, or that we make use of our knowledge in new ways through practice, and thus discover new interrelations or gain a new understanding of our knowledge. Kolb's (1984) learning cycle, which is based on experiential learning, complements

Figure 2. Kolb's (1984) learning cycle



Cook and Brown's comprehension of knowing, as it can explain those aspects of knowing which deal with the creation of new knowledge. Comas and Sieber postulate that the four different stages of the learning process indirectly reflect the presence of different states of knowing, and argue that moving through the cycle contributes to what Cook and Brown have called productive inquiry.

There is, however, a difference between the four types of knowledge and how they relate to learning. The learning process depends on the types of knowledge involved, but also on different ways of learning (Nielsen & Kvale, 1999). This is a part of our theoretical framework and it will be discussed next.

The concept of scholastic learning describes the (part of) learning which takes place through verbal and textual instruction detached from practice (Nielsen & Kvale, 1999). This comprises the communication of information which represents explicit knowledge, and as such, concepts, rules, equations, and interrelations are acquired through scholastic learning. This involves learning through concrete institutionalised education, for example, through courses, seminars, or literature studies, but also through verbal and textual instruction among colleagues at the workplace.

Learning of tacit knowledge occurs through active participation in practice as well as general learning in day-to-day life. Wenger (1998) presents a social theory of learning in which participation in practice forms the basis for learning. When people work closely together, share a practice, a vocabulary to talk about the practice, and an understanding about the practice, tacit knowledge in the form of skills, proficiency, routines, and shared genres is more easily shared (Wenger, 1998; Wenger & Snyder, 2000). Practice learning takes place in communities of practice which are particular areas of activity or bodies of knowledge which a community has organised itself around. It is a joint enterprise inasmuch as it is understood and continually renegotiated by its members who are linked to each other through their involvement in common activities (Wenger, 1998).

Over time, a community of practice builds up an agreed set of communal resources, a shared repertoire which consists of tangible as well as intangible aspects such as procedures, politics, rituals, and values (Wenger, 1998; Wenger, McDermott, & Snyder, 2002).

While scholastic learning is primarily about receiving, processing, and "absorbing" information, practice learning is primarily a question of becoming part of a community (Brown & Gray, 1995).

Individual learning can occur through scholastic as well as practice learning, and it is an individual's learning of explicit and tacit knowledge. Individual learning denotes the acquisition of knowledge which an individual (him/herself) uses in and for his or her actions. An example is knowledge in the form of concepts about the design of a product, or skills in the application of tools or routines for the performance of particular tasks (Comas & Sieber, 2001).

Group learning is learning of tacit and explicit knowledge on a group level. Group learning can also occur through scholastic as well as practice learning. Through group learning, the group acquires knowledge which it uses for its actions. This comprises knowledge in the form of rules for how a task is to be solved, shared stories about successes and failures, routines for how the group distributes tasks, and genres for particular meetings and documents.

Communities of practice (Lave & Wenger, 1991; Wenger 1998) are a prominent example of group learning, being groups of individuals who work together over a period of time. The individuals in these groups may carry out the same job or cooperate on a shared assignment, but they do not have to be a formal or identifiable group (Brown & Gray, 1995). Individuals in communities of practice are bound together through a shared practice and understanding of this practice, and

they develop shared knowledge about the practice (Brown & Gray, 1995).

Communication and Active Participation in Action: Knowledge Sharing Through Social Interaction or Technical Media

Knowledge sharing is a process in which knowledge is exchanged between individuals and groups (Comas & Sieber, 2001). The access to and exchange of information is necessary for the acquisition of knowledge (Kautz & Thaysen, 2001), and communication is therefore important for the sharing of knowledge.

We take as a starting point Fiske (1990), who argues that communication can basically be understood as the exchange of information. Fiske relies on Lasswell's (Severin & Tankard, 1997)—Who (says) What (to) Whom (in) What Channel (with) What Effect—and Shannon and Weaver's (1949) rather general and simplistic models of communication, but emphasises that the communication process takes place with the aid of a shared oral and written language, body language, or action. In this model a sender or communicator chooses, combines, and presents data in such a way that it represents the information which he or she wants to communicate. A receiver recognises and takes information in by interpreting the transmitted data on the background of existing knowledge, other information, context, culture, and the rules and pragmatics of the language.

Communication can take place as a dialogue, meaning that the roles of sender and receiver change continuously during the communication, or as a monologue, in which case the roles of the sender and the receiver are fixed throughout the communication session. The roles of sender and receiver of information in the communication process can be held by individuals and by formal or informal groups. Formal groups are groups which are linked to business processes or organisational units and which have been defined by an organisation's management (Brown & Gray, 1995). Informal groups are groups which are connected with informal business processes or built by people who congregate outside or across the formal organisational units.

The differences between how individuals or groups exchange knowledge have an impact upon how and which information can be communicated. Information can not only be found in speech, but is also expressed through body language and action. An individual can obtain information which is embedded in actions and body language by participating in someone else's work. This means that information has different formats, is more or less structured, and is stored in different ways, which all have an influence on how information can be communicated.

Knowledge sharing takes place through inter-subjective and/or technology-facilitated communication (Thompson & Walsham, 2001). Inter-subjective communication takes the form of social interaction, while technology-facilitated communication comes about through technical media. There is a difference between which type of information can be most suitably exchanged and provided through social interaction and technical media (Thompson & Walsham, 2001).

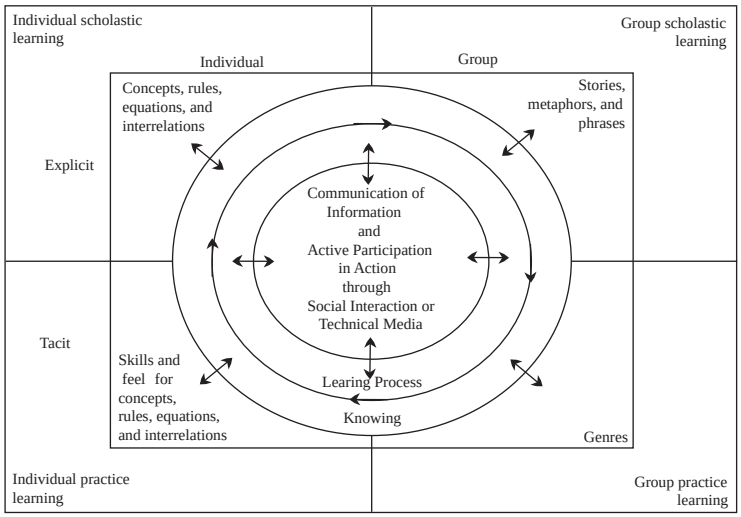
Social interaction as the exchange of information face-to-face between people comprises, for example, formal or informal meetings, verbal presentations, and teaching sessions. In social interaction, information is communicated through speech, body language, or actions; hence, social interaction enables the exchange of information linked to both explicit and tacit knowledge (Nielsen & Kvale, 1999). Explicit knowledge can be codified, and it can thus be expressed verbally (or textually) in a shared language. Tacit knowledge cannot be codified and it can therefore not be expressed verbally or textually; body language and actions are thus important for the learning of tacit knowledge (Nielsen & Kvale, 1999). This

indicates that social interaction is suitable for the communication of complex, less structured, and nonformalised information because it provides different possibilities for communicating a message. This happens through the application of a combination of a shared language and speech, body language, and actions which can supplement each other. Social interaction in itself cannot store information, and thus the storing of the exchanged information through social interaction is dependant on the individuals' or the group's capability to "store" across time and place.

Technical media are technologies, especially information and communication technologies, which support the exchange of information in the whole or in parts of the communication process. These can, for example, be telephones, databases, electronic documents, or e-mails. Technologies can "transport" communicated information between the involved parties and simultaneously support the storage of data (Thompson & Walsham, 2001). Technology thereby provides the possibility of making information available across time and space. Technology can also contribute to changes in the format or structure of information in the form of categorisation of documents, composition of document indices, or conversion of analogue sound to a data file. This improves the communication in relation to a situation where technology is not applied.

Technical media are suitable for the communication of codified information (Thompson & Walsham, 2001) as most technical media are based on verbal and textual communication. Some technologies, such as video, can also reproduce information expressed through body language and action. Technical media are suitable for well structured, well defined, and formalised information such as technical specifications or measurement results. Technical media primarily support the communication of information related to explicit knowledge, and they can only indirectly, and to a limited extent, support communication of information related to tacit knowledge (Thompson & Walsham, 2001). Different strategies are used to access that information depending on the type of

Figure 3. An integrated model of knowledge, learning, communication and participation in action



knowledge and the background of those who use the medium (Smolik, Kremer, & Kolbe, 2005).

An Initial Theoretical Model for Knowledge Sharing

In summary, in this section we have presented our understanding of knowledge, learning, communication, and participation in action as constituting knowledge sharing via different types of media. In Figure 3 we have integrated these concepts as a step towards a unified model of knowledge sharing.

In the model, learning occurs through scholastic or practice learning, and the learning process is the process which determines how individuals and groups acquire different types of knowledge through the communication of information and participation in practice. The model thus shows how knowledge, organisational memory, and learning are related, and how they influence each other as prerequisites for knowledge sharing. We will use the framework to analyse the relationship between ways of learning and types of knowledge as prerequisites for knowledge sharing in our case organisation, and take it up again later and discuss it in relation to our findings, which resulted in a refinement of the theoretical framework.

We use this model to discuss our empirical findings. In the next section we present the research approach and setting of our empirical study of knowledge sharing in software development as a prelude to the presentation of the findings in the *KNOWLEDGE SHARING IN THE CASE ORGANISATION* section.

RESEARCH APPROACH AND SETTING

The ontological and epistemological assumptions of our work are informed by the interpretive paradigm. The objective of our research was to study and understand the organisational members' knowledge sharing processes within their work and organisational context. In line with Andersen (1999), we chose a case study approach in order to study the process of knowledge sharing in its organisational context. To ensure the validity of our work we largely followed Klein and Myers's (1999) seven principles for interpretive field research and Eisenhardt's (1989) seven steps for case study research.

Given the resources available to our study, we concentrated on a single case study of knowledge sharing in a large Danish IT company. This approach might be criticised for only generating a local empirical theory which might not be generalisable, but as argued by Hughes and Jones (2003), it contributes to the existing body of knowledge by providing a detailed account of empirical findings. By limiting the study to a single organisation, we were able to examine the case in more detail to understand more thoroughly the interrelationships of separate data, which was our research objective.

The research setting was a Danish subsidiary of a global provider of telecommunication systems. At the time of the study (in 2002), the Danish subsidiary was 23 years old and it had been acquired by the American parent company 6 years earlier. The company's customers are fixed network, mobile network and cable operators who reside in more than 80 countries.

The Danish subsidiary concentrates on the development of new products. The company's products consist of hardware and embedded software which provide the product functionality and management software to access, calibrate, and monitor the products' functions. This is mirrored in the organisation of the company's research and development (R&D) department, which consists of three divisions (access, calibration, and monitoring) which perform the detailed specification and actual development tasks. In addition, the company has a product management department; a product planning and specification department; and a finance department. The company has 420

employees of whom 180 are directly involved in product development.

In agreement with company management and also because of resource restrictions, we agreed to focus our investigation of the knowledge sharing process on one organisational unit—the division of embedded software. The embedded software division consists of some administrative staff, a manager, three project leaders, two team leaders, and 27 staff members. With its position between different departments and divisions, embedded software appeared to be most appropriate and representative for our purpose in light of the resources available for the investigation.

The company was interested in participating in our research study due to increased competition in the market, they had decided to focus on employees' knowledge, learning, and communication both within and across the organisational units with the aim of increasing productivity and quality in the product development process.

The product development process is organised in programmes consisting of projects. The company has about three large and 5-10 minor programmes running at any one time, consisting of multiple projects of varying length from 6 months to 2 years. The projects are typically carried out in and by the different divisions. The company's development process can be characterised as product-oriented, sequential, and document focused. The process consists of an analysis and design phase, a development phase, a test phase, and a product installation and introduction phase.

A planning team consisting of experienced employees from different departments is responsible for the overall definition of a programme. A so-called core group of project leaders and technical experts performs a feasibility study and produces product and document reviews at the end of each development phase. Finally, a core team which consists of the project leaders of the different divisions in R&D is responsible for conducting and completing the actual projects and products. The development work is performed by the individual divisions and their developers and is coordinated by the project leaders in and across the three divisions and beyond.

The developers in the division were our primary unit of investigation. Within the category individuals and groups within the division under investigation, we differentiated between developers in the division, developers who worked on the same tasks, developers who shared a two-staff office, project groups, subproject groups, informal groups, and experience (exchange) groups. Individuals and groups related to other units were roughly distinguished into individuals in other divisions and departments affiliated with the same project or product, other departments and divisions in the Danish subsidiary, or individuals and groups in other countries.

We collected data through 13 semi-structured interviews with eight developers, our primary group of interest, and one project leader in the unit under investigation. We also interviewed four employees from other divisions and departments. We also performed one group interview with management personnel and two one-workday-long observations where we followed a member of management and a developer. Finally, we have included secondary material in the form of the different kinds of documents used in the organisation.

All interviews were taped, and a summary listing relevant issues from each interview was written. During and after the interviews we produced rich pictures inspired by the Soft Systems Methodology (Checkland & Scholes, 1990) to depict our understanding of the situation with regard to knowledge sharing in the organisation. These rich pictures constructed the basis for the data analysis and were so useful that we only had to return to the tape recording for further consultation in one case.

To analyse the data we used a combination of common sense interpretation, inspired by the Soft Systems Methodology, and theoretical interpretation which related the data to our theoretical

framework (Andersen, 1999). For all interviews and observations we identified and coded sensible units of expressions. These statements were then linked to one or more categories which we deduced from our theoretical framework, and by comparing all statements within each category and across categories, a number of recurring subjects emerged, and over 100 issues critical for knowledge sharing in the case organisation were identified. The three main categories, which will also be used in the subsequent in-depth description, were (1) knowledge sharing through social interaction, (2) knowledge sharing through technical media, and (3) (the) different kinds of knowledge (which are) shared.

This allowed for an in-depth understanding and discussion of knowledge sharing in the case organisation, revealing what kind of interaction and which media provide possibilities for sharing different kinds of knowledge. It also revealed what impact the identified issues had on the sharing of knowledge, and it ultimately led to a refinement of our theoretical framework.

KNOWLEDGE SHARING IN THE CASE ORGANISATION

We explore the case study to identify how the software developers draw upon organisational memory to share knowledge. As part of our research question, we investigate how knowledge sharing in software development unfolds in social interaction and via technical media. Our findings provide some interesting insights on these matters and present a more detailed account of the kinds of knowledge which are shared. Another interesting finding concerns how and when the developers rely on formal or informal contacts in the knowledge sharing process. Finally, the findings show a difference in the unfolding of the knowledge sharing process according to individual or group action.

When we refer to knowledge sharing as unfolding through social interaction, we relate to the subjects' knowledge sharing through formal interaction in formal meetings, informal interaction in informal meetings; seminars and courses; presentations; exchange of experience groups; personal networks; and in the offices shared by two or more people.

When we refer to knowledge sharing as unfolding through technical media, we cover the subjects' knowledge sharing by the use of the organisation's document handling system (DHS), the file server, the error reporting system, the version control system, the project management system, e-mail, electronic discussion groups, Internet, intranet, video, whiteboards, paper documents, and literature.

When we refer to kinds of knowledge, we cover the subjects' shared knowledge of technical specifications (particularly requirement specifications and design documents); general project information and technical standards; information about who knows what about previous projects or old products; and information about technical problems and improvement of products.

We then provide a more detailed analysis of how knowledge sharing unfolded between individuals and in and between groups, and what kinds of knowledge are shared. We identify critical issues related to each of the categories and provide some suggestions regarding how some of these issues can be solved. As limitations of space preclude a full description of our analysis, we have chosen two subjects from each of the categories to illustrate our findings.

Knowledge Sharing through Social Interaction: Formal Meetings and Personal Networks

Social interaction in the case study took place through formal meetings and personal networks which are now discussed.

Formal Meetings

The members of the case study organisation participate in various formal meetings dealing with the communication of the status of projects and problems through verbal and textual communication. These formal meetings provide the opportunity for the acquisition of explicit individual and group knowledge. Explicit individual knowledge is shared in the form of coordination, communication of status, and discussions of tasks. Explicit group knowledge is shared in the form of (communication of) management attitudes and opinions; definition of rules; and (communication of) stories about good and bad projects. The formal meetings are also used as "pathfinders" to those developers who hold individual knowledge about, for example, concepts and rules, or who have specific skills. In this way the formal meetings provide the individuals with the possibility of obtaining knowledge about where they can obtain further knowledge. Active participation in formal meetings also provides the opportunity for acquisition of knowledge about the genres which are related to formal meetings and the acquisition of individual skills about how meetings are run.

The following issues were identified in relation to the formal meetings:

- **The information communicated through project meetings is arbitrary.** This shows that the developers are not sure that the information they need, or which would be helpful to them, is communicated through meetings. It contributes to the project meetings having less value for the developers, and it indicates that the developers are not sure about the genres which are related to formal meetings, or that they lack skills to utilise formal meetings. The study does not provide any evidence that the developers are trained to use formal meetings. The developers therefore communicate to a higher degree through informal meetings which are "held" as the need for particular information arises, or through personal networks. Information about the same subject is thus communicated through many different forms of social interaction and through different technical media. This in itself does not have to be a problem as a certain amount of redundancy increases the chance that everyone will receive the desired or necessary information, but it is problematic when it confuses the developers.
- **There are no project meetings between divisions, and coordination across divisions rarely happens in formal meetings.** These issues indicate that communication between the divisions is generally poor, and that it is limited to taking place through the project leaders' personal networks. This reinforces the organisational divide between the developers in the different divisions and creates a barrier which does not support communication or insights into other developers' work. The communication and the exchange, which happen across the departments and is necessary for the developers, instead comes about through personal networks or written information. The communication between divisions is much more difficult than within a division because one has to be much more precise, as the developers cannot assume that the receiver of their information has the same background knowledge.

Personal Networks

Personal networks are personal relationships between the developers. These are used to communicate information and to instruct each other. Communication of information through personal networks is verbal and textual and provides the opportunity for scholastic learning of explicit knowledge. This is true for both individual and explicit group knowledge because personal networks are directed towards communication of concrete

problems, discussions of different possibilities for solutions, and informal education of the individual developer. This knowledge takes the form of concepts and rules related to projects, products, and other technical matters which the developers need for the performance of their work. Good and bad experiences are communicated for this purpose, and phrases are used which are defined by the groups in the organisation. This happens, for example, through informal meetings, which are the links for, and which make the networks visible. The developers communicate information through informal meetings, and at the same time they can identify other developers' personal networks and thereby build and extend their own personal networks.

The developers become actively involved and participate in each others' practice through personal networks, and in so doing they have the opportunity to acquire the genres which are related to personal networks. In addition, the networks give the developers the opportunity to acquire skills in how personal networks can be used, and skills related to the topics which are communicated. This happens, for example, through the two-staff offices, which facilitate the developers' participation in the same practice. It is common practice in the company that an experienced developer shares an office with a less experienced developer to help him or her to achieve skills and learn genres. This includes helping the less experienced ones by pointing them in the right direction when it comes to searching for information. This occurs in communication about how things work, and where what information can be found, as well as in the form of stories and rules related to work in the divisions.

The following issues were identified in relation to personal networks:

- **The poorer the formal meetings and the technical specifications, the more the developers use their personal networks.** This shows that personal networks are pivotal for communication in the division because many other forms of social interaction and technical media are not adequate and detailed enough for the developers to communicate all the desired or necessary information.
- **The developers gain a "feel" through the personal network for what is important and whom they should go to in order to prioritise resources and tasks.** These issues show the importance of the personal networks; they bind other forms of social interaction and technical media together, and they satisfy the need for communication which is not covered through other social interactions and technical media.
- **Developers depend on personal networks, which are strongest within the division, but more experienced developers have networks which span divisions.** This issue shows that the developers are dependant on their personal networks to be able to perform the work for which they need information from previous projects. This dependency on the personal networks and on the most experienced developers can be problematic when the developers, who have comprehensive knowledge of previous projects and old products, leave the organisation and take their knowledge with them.
- **Personal networks save time**. This shows that personal networks are faster to use than searching in documents. The developers can find what they are looking for faster through their personal networks than through other forms of social interaction or technical media.
- **A good personal network takes time to build.** These issues show that personal networks are created through the developers' daily work, and that a good network takes many years to build up. This is because the developers work on the same projects and

with the same colleagues over a long period of time. This contributes to those developers who have broad and comprehensive networks having a lot of information at their disposal, while others with lesser networks have less information. This indicates that not all developers have comprehensive personal networks, which results in less communication of information and less opportunity for the acquisition of new knowledge for these developers.

Some Conclusions on Knowledge Sharing through Social Interaction

The analysis of the examples of social interaction shows that social interaction through verbal and textual communication provides the possibility for scholastic learning of explicit knowledge. Beyond this, the active participation in action in the form of social interaction provides the opportunity to acquire those genres which are related to social interaction and the acquisition of the individual skills for how social interaction can be applied, and the skills related to the topics and themes which are communicated. The investigation shows that the individuals and groups in the investigated division communicate through social interaction, and social interaction thus provides the opportunity to share knowledge in the division.

This finding is interesting, as the analysis also shows that social interaction is not facilitated in a structured way, and that formal forms of social interaction in general are not prioritised. This creates problems, as the developers are uncertain about which information is communicated through which type of social interaction, and the findings show that this results in a high degree of arbitrariness with respect to which information is communicated through which form of social interaction, for example, in courses, presentations, experience groups, or personal networks.

Knowledge Sharing Through Technical Media: The Document Handling System and the File Server

In our case study company, documents used to share knowledge are stored on two kinds of technical media: the DHS and the file server.

The Document Handling System

The DHS provides the opportunity for textual communication. By far the largest part of all technical specifications, analysis documents, and project-related documents is stored in the DHS. This comprises feature plans, functional and object models, and iteration plans, but also how-to documents, technical standards, and presentations. The documents are used by the individual developers in their work as they describe what and how something should be or has been done. The documents are also directed towards groups as they provide the framework and the scope for the groups' collective work. The verbal and textual communication through the DHS offers the opportunity for individuals and groups to acquire explicit knowledge.

Active participation in the form of interaction with and use of the DHS provides the possibility for the developers to gain skills in use of the system to acquire the genres related to it. This comprises which documents are stored in the system, how the documents are structured and categorised, how search functions are applied, and how keywords and indices are linked to documents.

- **Not all documents are stored in the DHS, and as a consequence not all necessary documents are accessible to all developers.** This issue is related to the fact that the DHS is regarded as being for documents which are directly related to projects or the product development process. Other documents

which do not comprise formal or project-defining information, such as descriptions of "private" ideas for solutions, experience or general descriptions of problems, are often stored on other media, among others the file server. Only what the individual developer thinks has to be version-controlled is stored in the DHS. The developers have to search for information in many places and to supplement information from the DHS with information from, for example, the file server or their personal networks. This indicates that developers need to have a good overview of the information and have to make an extra effort to gather the quantity and the quality of information they need for their work.

- **The DHS is slow and hard to use, the search functions are poor, and distribution lists are not used to the necessary degree; it is difficult to find the right documents; there is a lack of abbreviations and designations, and the information structure in the system is strictly hierarchical.** These issues indicate that the actual functionality and the information structure in the DHS hinder the developers' use of the DHS and the communication of information. This limits the developers' chances of sighting all documents and finding all relevant information for a particular task. The strict information structure does not mirror the individuals' and the groups' problem-oriented way of working. Communication of information in the DHS is based on the partition between projects and divisions instead of being based on the structure of the working processes, problems, and solutions.
- **Not all departments are equally engaged in using the system.** This issue reinforces and even increases the problems related to the developers' limited possibilities of accessing all documents and finding all relevant information for a particular task. Although the DHS is the official place for storing and distributing documents, some departments use other technical media, for example databases for technical specifications.
- **Many developers use the system only to the extent to which it has an impact on their communication with other divisions, or when demanded as part of the formal work process.** This indicates that some divisions provide their documents to developers in other divisions only to a very limited extent, and that the division only has an insight into other divisions' documents to a limited extent. The lack of insight into each others' work makes the communication of information across divisions difficult and hinders the sharing and exploitation of explicit knowledge. In the case where the use of the DHS is bypassed to the advantage of other technical media, this also has an influence on the acquisition of tacit knowledge related to the use of the DHS, especially the individual skills related to using the DHS for the storage of technical specifications. This is also true of the acquisition of genres related to the DHS. Information about genres is communicated though the documents in the DHS. When the same information is communicated through different forms of social interaction and technical media, meaning that different genres are used for the same themes, this can lead to a situation where the same information gains different meanings.

The File Server

The file server is primarily used for informal documents such as technical notes, proposals for technical designs, intermediate and temporary versions of documents, standards, product de-

scriptions, third party documents, and so forth, which are not directly or ultimately related to the projects or the product development process. Apart from these documents the file server also contains binary files in the form of programs and video recordings. This shows that the file server is used for textual and verbal—in the form of video recordings—communication, which provides the opportunity for acquisition of explicit knowledge related to the information that is stored on the server. Interaction with and use of the file server provides the opportunity for the developers to acquire skills in the use of the file server and genres related to it. These are implicitly determined by the developers and deal, for example, with which documents and files are to be stored on the server; how they are and should be structured; and which search functions can and should be applied.

The following issues were identified in relation to the file server:

- **The file server is used to store files which should be stored in the DHS.** This problem is related to the fact that the DHS is regarded as being for documents which are directly linked to the projects or to the product development process. Nevertheless, some older documents have not been transferred to the DHS and are stored on the server. This includes, for example, own ideas and experiences; technical notes; proposals for technical designs; designs; intermediate and temporary versions of documents; standards; product descriptions; third party documents; and so forth, which are not directly or ultimately related to the projects or the product under development. This indicates that information which relates to the same subject is stored and communicated through different technical media: for example, own experience and proposals for technical designs are stored on the file server, while technical specifications related to the same product are stored in and communicated through the DHS. This results in a situation where the developers have to gather information from different places, depending on whether the information is categorised as formal or informal, or whether the information relates to newer or older products. This means that the developers must have a good overview of the information and extra time to gather the necessary quantity and quality of information. As information has an impact on the learning process and ultimately the acquisition of knowledge, the use of the file server hinders the sharing of explicit knowledge, as it functions as a ragbag and as it provides the opportunity to store information in places other than the DHS.
- **The developers' and the divisions' use of the file server is very different.** Some developers never use it; others use it often or store and find much useful information there. The divisions also structure files differently on the server, and this is an obstacle to the sharing of knowledge. The different use of the file server makes the communication of information through this technical medium look arbitrary and dependant on individuals. The different use of the file server holds the risk that the developers do not communicate the same information, and thus that the information accessible to the other developers becomes different. There is a risk that those developers who do not use or do not have knowledge of the information on the file server do not acquire knowledge of the same concepts or stories. This then decreases their later capabilities to act. The developers' different uses of the file server lead to differences in the extent to which the developers acquire skills related to the file server and knowledge of the file server as a genre. This includes skills related to

the use of the file server to store and work with technical standards or the handling of video recordings or other binary files on the server. It also includes the genres related to the file server, for example the form in which information in a technical standard is communicated through the medium.

- **It is difficult to find information on the file server; the developers have to know where the information is stored to find it.** The problem with access to information on the file server is emphasised by this issue. This is due to the fact that the file server has limited search functions which only cover searching based on file names, and that the directory structures and naming are very "anarchistic." This means that the developers have to have considerable knowledge of the file server and its contents to be able to search for information there. Information about the file server is communicated through personal networks, but the personal networks take a long time to build up and seldom cross division boundaries. It is thus difficult for new employees and developers in other divisions to find relevant information on the server.

Some Conclusions on Knowledge Sharing Through Technical Media

All these issues deal with access to information. As information is the basis of the developers' acquisition of knowledge, access to and gathering of information are necessary for the acquisition and sharing of knowledge. The analysis shows that the issues identified in relation to the developers' use of technical media to communicate have a negative influence on the sharing of knowledge between individuals and groups in the company division which was investigated.

Sharing Different Kinds of Knowledge: Technical Specifications and Information about "Who Knows What"

Finally, we shift focus from communication of information and participation in action through social interaction or technical media and emphasise what kinds of knowledge are actually shared, and we concentrate here on technical specifications and on information about "who knows what."

Technical Specifications

Technical specifications comprise feature plans; device transport and management specifications; overview and detailed function and object specifications; and hardware specifications. They are the basis for the acquisition of explicit knowledge, and the developers actually use specific technical specifications for product development. The technical specifications are directed towards the individual developer's work and at the same time define the rules and phrases for the collective, joint, and shared development work. They thereby form the basis for the sharing of explicit knowledge between individuals as well as groups.

The developers, who have been in the division for a long time, can understand and use technical specifications better than others. This indicates that through their daily work, the developers acquire skills and genres in utilising technical specifications. Active participation in action provides the opportunity for the acquisition of tacit knowledge. This means that the developers' use of technical specifications contributes to their skills in utilising technical specifications. Their skills related to the topics that the technical specifications deal with, and provide possibilites to the acquisition of genres that are related to technical specifications.

However, not all developers understand the documents equally well. When they do not under-

stand the documents they contact the developers who originally produced the documents and ask them what they were trying to describe and express. This shows that the developers need certain skills and knowledge of the applied genres to be able to use the information which is provided in the technical specification.

The following issues were identified in relation to the use of technical specifications:

- **The developers find it difficult to gain an overview of what has been agreed on, and which of the changes in the specifications have been carried out; changes are sometimes carried out before they are agreed upon and approved; not all changes are documented and reviewed.** These problems show that communication about changes or updates to specifications and documentation is sometimes difficult or defective. Acquisition of knowledge is based on the information which the receiver obtains. Access to and exchange of information is thus necessary for the acquisition of knowledge. The developers' different and nonupdated information basis therefore has an impact on the explicit knowledge they acquire, and ultimately on their possibilities of action.
- **The change management tool is not used to receive information on updated technical specifications; the document handling system is sometimes bypassed and e-mail is used instead.** These issues show that the communication of these documents and information on updated technical specifications takes place through many "routes" and different technical media. This indicates that the developers are not secure in and certain about the genres which are connected to these technical media, for example, the specifications are communicated through the DHS, but updates to these documents are communicated through the error reporting system, while both types of information should be communicated through the DHS.
- **Sometimes feature plans are not finished when a project starts, which results in guessing and rough estimates; there is no possibility of tracing all features through the whole product development process including the final test of the product.** These issues indicate that the quality of the information which the developers use in their work is insufficient. As noted earlier, access to and exchange of information is necessary for the acquisition of knowledge, and when information is not available, acquisition of knowledge is impeded or hindered.
- **The hardware specifications are difficult for the developers in the division to understand.** The hardware division uses different standards and a different version of the control system; hardware specifications are formulated in a 'slangy' way and are not written in pedagogical form; the schedule seldom leaves the developers an opportunity to dig deeply into the hardware specifications.

These issues also indicate that insufficient information, and information which is difficult to understand, as well as lack of time, hinder the sharing of explicit knowledge and the developers' possibilities of action.

Information on "Who Knows What"

Information on "who knows what" includes information about which employees and colleagues have which skills, and knowledge about projects, products, and technologies. As this information is directed towards the individual developer's seeking of information and simultaneously defines rules for who the developer should turn to, this information is the basis for explicit individual and group knowledge.

Active participation in action provides the possibility of acquisition of tacit knowledge. This means that the developers' utilisation of personalised knowledge contributes to the skills in utilising these skills and in skills to actually find the colleagues in question, and to the acquisition of the genres which are related to the skills and to these people. The study shows that the developers use a lot of time finding out who knows what, and that they do not always succeed. The developers find that knowledge about who knows what comes with time, when they become familiar with other developers.

The following issues were identified in relation to knowledge about "who knows what":

- **Many developers lack insight into who has what information and skills; the developers often only know the skills of those they work or have worked with in projects.** These issues show that many developers have limited knowledge of their associates. However, for development of the technical skills it is important to know whom to ask in the organisation, while formal education is less important. The lack of knowledge of colleagues can thus have a negative impact on the access to and communication of information, and ultimately on knowledge sharing, as the developers do not have an overview of what information can be found where, and this is even more problematic given that it is most important to know who to ask as part of the development of the skills which the developers require to perform their tasks.
- **The project leaders know who has knowledge; the division's "catalogue of competences" has become too voluminous to be updated; it is difficult to find information about who works in which project.** These issues show that the project leaders are prominent as those who know who has what knowledge. The overview over who has what information and which skills, and therefore the access to other people's information and capabilities, depends on a relatively small number of people. The division's catalogue of expertise is no longer updated, and therefore is of very limited help in this respect. The developers instead use their personal networks to obtain this information. The difficulties in localising information and skills of other developers are reinforced through the fact that the developers have difficulty finding overviews of who works in which project.
- **The developers do not know what other developers and colleagues need to know.** At the same time as they do not find desired or necessary information, the developers do not know what others need to know. This hinders the distribution of information to developers who might benefit from it.

Some Conclusions on Sharing of Different Kinds of Knowledge

In summary, the analysis of technical specifications and information about "who knows what" as exemplars of different kinds of knowledge shows that the identified issues can be detrimental to the sharing of knowledge in the case organisation, as this knowledge is difficult to find, and in part also hard to understand. The developers instead use their personal networks to acquire the knowledge they need. However, as personal networks take time to build, this solution works best for those employees who have been with the company for long time.

DISCUSSION

The study has shown that the integrated model can explain the relationship between the communication of information, participation in action, the learning process, and knowing the different

types of knowledge. It thus presents an appropriate basis for a detailed understanding of knowledge sharing. The empirical data provide examples of how different forms of knowledge are shared in social interaction and by using different media.

Our study shows that when there are problems with the communication of information such as missing information, poor access to information, or poor quality of information, this determines the extent to which explicit knowledge can be acquired. Similarly, if active participation and learning in practice are hampered or function poorly, this determines the extent to which tacit knowledge can be acquired.

Our work is based on the assumption that a number of different dimensions and perspectives have to be taken into account in order to create a comprehensive understanding of the knowledge sharing process.

Other authors such as Cook and Brown (1999); Comas and Sieber (2001); Kolb (1984); and Nielsen and Kvale (1999) only deal with the relationship between two dimensions such as communication of information and knowledge, or knowledge and learning, or they consider and describe the dimensions as different perspectives on knowledge sharing. We take all these different dimensions and perspectives into account.

Cook and Brown (1999) identify types of knowledge and knowing which can exist in an organisation and the relationship between them. They consider knowing primarily as productive inquiry. They do neither relate them to the concept of learning or to the importance of communication of information. Comas and Sieber (2001) discuss managing knowing by describing the relationship between experiential learning and knowing. They argue that managing knowing can be described as an experiential process, and they thus emphasis knowing primarily as a learning process. They do not, however, draw on the importance of what precedes this process, namely communication of information, and the actual situation in which learning takes place. Kolb's (1984) learning cycle also concentrates primarily on the learning process with a focus on individual learning which ignores the group dimension of learning and knowledge. Kolb also emphasises how information is gathered and processed in the learning process, but does not deal with what happens before in the form of communication of information. He thus ignores the problems which can arise when obtaining information.

Nielsen and Kvale (1999) identify scholastic learning and practice learning as well as the explicit knowledge and tacit knowledge to which these ways of learning contribute, but they do not clearly deal with the individual and group dimensions of knowledge and learning. We take all these issues into account.

Finally, our understanding of communication is distinct from the original models (Fiske, 1990) by being explicitly directed towards our focus on knowledge and learning processes, and by viewing communication not as a technical, decontextualised process, but as a social process which takes place in a context. We thus emphasise the concepts of social interaction and technical media instead of the technical term *channel,* and we do not use the terms *sender* and *receiver,* but stress the concepts *individual* and *group* in relation to communication.

In summary, our theoretical framework thus contributes to research within the field of knowledge sharing by integrating different areas related to knowledge sharing.

The theoretical framework does however not clearly show how communication in the form of verbal and textual communication and active participation in practice contribute to the acquisition of knowledge through scholastic and practice learning. Kolb's (1984) learning cycle can contribute to many types of knowledge. His learning cycle and the different types of knowledge focus on the individual's learning and the related cognitive process which, Kolb argues, can be dominated by a particular way of learning and type of knowledge. How far multiple learn-

ing cycles can take place simultaneously in the same situation is not evident from Kolb's theory. Comas and Sieber (2001) describe the relationship between experience or experiential learning and knowing, but they do not take a position on either, whether or if so, how many learning cycles take place at one time. Cook and Brown (1999), however, argue with respect to the relationship between tacit and explicit knowledge that these are separate types of knowledge, but that one is or can be used as support to acquire the other. We now provide examples from our study and discuss how different types of knowledge contribute to the acquisition of other types of knowledge, how this can be conceptualised as multiple learning cycles, and the consequences this has for our framework.

The Relationship Between Individual and Group Knowledge

A number of developers in our study explained that they discussed technical problems and solutions with other developers using, for example, personal networks and experience groups. We found that the developers had a shared repertoire of phrases for particular types of technical specifications. This indicates that the developers' individual explicit knowledge about, for example, how technological solutions are specified contributes to the group's use of phrases concerning technical specifications, and thus to explicit group knowledge, about specific types of technical specifications.

We also found that the developers primarily store formal documents, that is, documents which are a defined part of the development process or which are somehow related to projects and products, in the DHS. These documents are placed in a strict hierarchical structure which is ordered according to departments, products, and/or projects. This indicates that the developers' skills—tacit individual knowledge of skills and "feel" for which documents should be stored in the DHS, and where—contributes to the genre, tacit group knowledge with respect to which documents are communicated through the system and how they are communicated.

The study also showed that new developers are trained in use of the system through courses and apprenticeship learning. This indicates that the genre tacit group knowledge related to the DHS contributes to the development of the individual developer's skills in relation to the use of these genres. Figure 4 depicts these relationships.

The Relationship Between Explicit and Tacit Knowledge

The relationship between explicit and tacit knowledge is depicted in Figure 5. A number of the

Figure 4. The relationship between individual and group knowledge

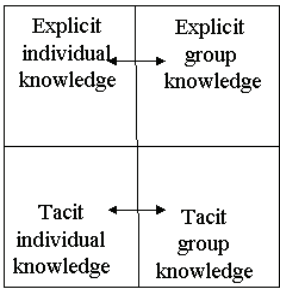
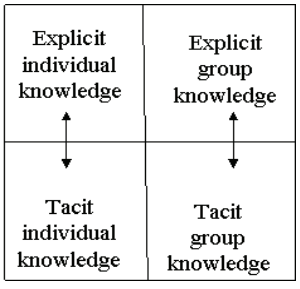


Figure 5. The relationship between explicit and tacit knowledge



developers said that on the basis of books, presentations, and courses they use their understanding of concepts and interrelations to experiment in practice and thereby acquire new skills. This indicates that the individual developers apply explicit knowledge in practice—practice learning which contributes to the acquisition of tacit knowledge in the form of skills. The study also shows that developers who are experts in a particular field tell other developers about how a specific task should be carried out. This can happen through presentations, experience groups, and personal networks, thereby giving other developers the opportunity to gain explicit knowledge about the action related to solving the task in question. This indicates that the developers reflect upon their tacit knowledge and explicate related actions and activities, which they communicate to other developers through scholastic learning.

The study also shows that the project management system which was previously used to plan projects is now only used for general project planning and time registration. According to the respondents it ought to be possible to extract estimates for new projects directly from the system, but because the projects are dissimilar, these estimates are hard to use. We also found that some developers had ceased to register time in the system. This indicates that stories—explicit group knowledge—about the deficiencies of the project management system make the developers interpret the information which is communicated through the system in an unfavourable way. They have thus developed a concept of the system which has become tacit group knowledge. The study also shows that the developers reflect over their use of the document handling system as a form of communication in the company which only functions poorly. This contributes to stories among the developers about how their work is made cumbersome by the system, and that necessary documents are hard to find. This shows that the groups reflect on their tacit knowledge and explicate actions related to this knowledge which contribute to the acquisition of stories and phrases in the groups related to the use of genres.

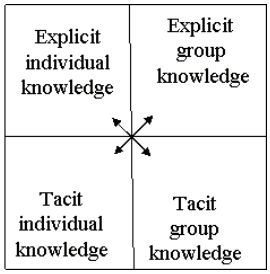
The Relationships between Tacit Individual and Explicit Group Knowledge and Explicit Individual and Tacit Group Knowledge

The relationships between tacit individual and explicit group knowledge and explicit individual and tacit group knowledge are depicted in Figure 6.

We observed how one developer, through use of his skills, developed a browser tool which other developers found useful. It became part of the development toolbox and contributed to the stories and phrases which the members of this group of developers told about useful tools. This shows that developers, through their skills and thus their individual tacit knowledge and its explication in related actions, contribute to the group's stories and phrases, and thus their explicit group knowledge.

The study also provided examples of how success stories about how to solve a specific task were sometimes transformed into how-to documents, for example how to program in C, which other developers could use to acquire new skills. This indicates that a group's stories enter into

Figure 6. The relationships between tacit individual and explicit group knowledge and explicit individual and tacit group knowledge



individual practice learning which contributes to the development of the individual group member's skills, in this case C programming.

Our findings show that the developers' understanding of the importance of standards for the development work makes them perceive what is communicated as a standard as being specifically important. This indicates that concepts or rules related to the use of a particular form of communication contribute to the development of genres. The study also shows that when developers observe others communicating, they gain knowledge about how, for example, one conducts a presentation or a formal meeting. This indicates that the developer's acquaintance and application of genres contributes to the developers' individual understandings of the concepts, rules, and interrelations which are related to genres.

Knowing as Multiple Learning Cycles

We have shown above how different types of knowledge contribute to the learning of different types of knowledge by drawing from our case study to show how the developers have knowledge which they apply to acquire new knowledge. Figure 7 summarises the relationship between the different types of knowledge and indicates a cyclic connection between them.

The various and different learning situations can also be understood as multiple cycles which can take place simultaneously—in other words, the developers constantly use knowledge to learn in order to create new knowledge. How this relates to the different types of knowledge is shown in Figure 8.

Figure 7. The relationships between different types of knowledge

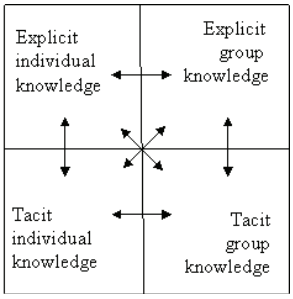
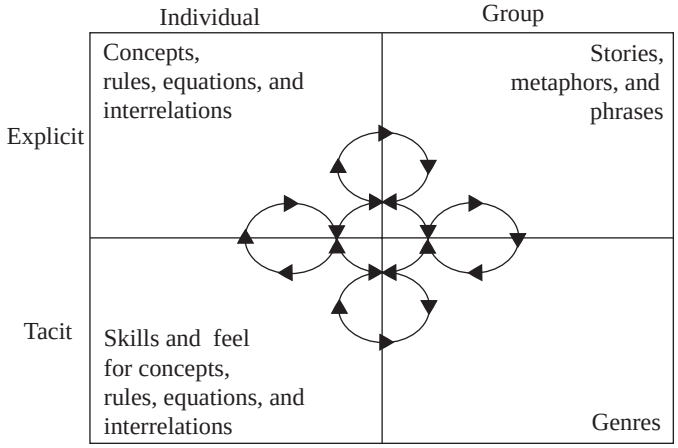


Figure 8. The relationships of different types of knowledge as multiple learning cycles

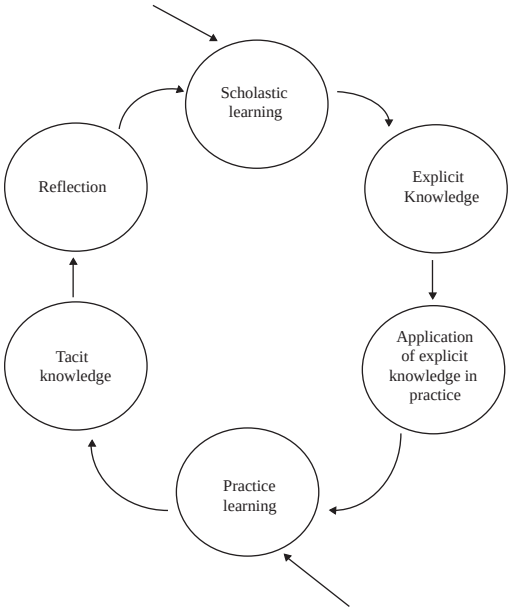


According to Cook and Brown (1999), learning is a process in which an individual or a group processes new information on the basis of existing knowledge which can then be used as a basis for action. The result of the learning process is new knowledge. Following Kolb's (1984) understanding of the learning process and the relationship between different ways of learning and different types of knowledge as expressed in our initial theoretical framework, and relating and comparing these to the relationship of different types of knowledge illustrated as multiple learning cycles and summarised in the preceding paragraph, the following becomes evident:

Tacit knowledge is acquired through practice learning, but when, for example, the developers explain how a task is solved, they reflect upon tacit knowledge. The situation where a developer explains to other developers what he or she has done is scholastic learning. It is not the tacit knowledge itself which becomes explicit through scholastic learning, but the developers acquire new explicit knowledge. This is not the same as being able to solve a particular task. If other developers want to achieve the same tacit knowledge, this can only happen through practice, that is, they themselves must apply the explicit knowledge in practice and participate actively in carrying out the task.

Explicit knowledge can contribute to the acquisition of tacit knowledge through application in practice and through participation in practice learning. At the same time, tacit knowledge contributes to the acquisition of explicit knowledge through reflection and participation in scholastic learning. Tacit knowledge and explicit knowledge thus supplement each other through reflection and application in practice (see Figure 9). This can be understood as a detailed description of knowing as a learning process. Taking scholastic learning as a starting

Figure 9. Learning of explicit and tacit knowledge through scholastic and practice learning



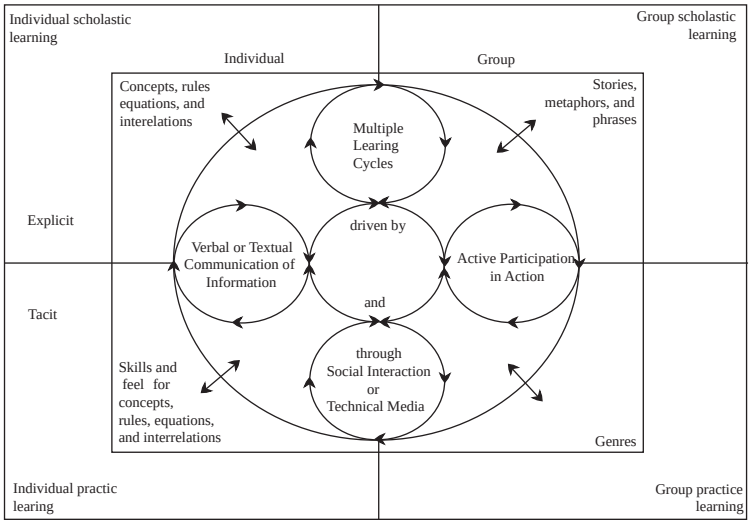
point, explicit knowledge will always precede an eventual acquisition of tacit knowledge. Taking practice learning as a starting point, tacit knowledge will always precede an eventual acquisition of explicit knowledge. It is a question of whether one starts with scholastic or with practice learning.

The Consolidated Model

The overall relationship between participation in action, communication, learning, and knowledge as the starting point for our theoretical framework was illustrated in Figure 3. On the basis of the study, we have brought together and integrated the theoretical framework's models for knowledge, learning, communication and participation in action, and amended them with the evidence for multiple learning cycles explained previously. As a result, the theoretical framework and the study's outcome can be presented in a model of sharing knowledge, where communication consists of verbal and textual communication and active participation in action. Learning comprises learning through scholastic learning, practice learning and multiple learning cycles, and knowledge consists of explicit and tacit individual and explicit and tacit group knowledge.

The model for the sharing of knowledge shows that verbal and textual communication of information through social interaction and/or technical media contribute to individual and group scholastic learning, which thereby contribute to explicit individual and group knowledge. The model also shows that participation in action through social interaction and/or technical media contributes to individual and group practice learning, which thereby contribute to tacit individual and group knowledge. The model for the sharing of knowledge shows in addition how multiple learning cycles and knowing contribute to the acquisition of knowledge.

Figure 10. The consolidated model



CONCLUSIONS AND FUTURE RESEARCH

In the research presented here, we have applied Cook and Brown's (1999) multiple forms of knowledge to an empirical example in software development, emphasising how these forms of knowledge are shared in communication via social interaction or different types of technical media.

Our work resulted in a model of knowledge sharing which can be used to understand and ultimately improve knowledge sharing in practice. Although some concrete advice for improving knowledge sharing could be derived from our analysis, the theoretical framework and our study do not describe how the results of the analysis and identified areas for improvements can be transformed into concrete organisational or technical improvements, or which strategies organisations should use to improve the sharing of knowledge. Furthermore, we do not include concrete guidelines for the design of information technology and technical media.

Problems related to the technical media lead to limited access to and quality of information. The developers in the case organisation thus use or supplement them with social interaction or other technical media which enable them to gain access to the information or actions which are a prerequisite for their learning processes, and the acquisition of knowledge which is necessary for them to carry out their tasks. How an optimal balance between social interaction and technical media might look has to be decided from case to case, and is again not part of our framework, but it could be constructed with the aid of Hansen, Nohria, and Tierney's (1999) work on codification and personalisation strategies. In this context it is necessary for future research to investigate the relationship between the different types of knowledge in more detail in order to clarify how this relationship can be used to improve the sharing of knowledge.

Finally, the study shows that personal networks play a specific role in the sharing of knowledge, as they link other forms of communication together and compensate for the information which is not communicated through these other forms of communication. The framework does not explain all phenomena related to personal networks with regard to knowledge sharing, and future research based on the framework should thus aim to extend the framework in this respect.

ACKNOWLEDGMENT

With acknowledgments to Dorte Boejstrup and Mads E. Bock, who worked as research assistants on this project during their master's theses.

REFERENCES

Andersen, I. (1999). *The apparent reality—On knowledge production in the social sciences* (3rd ed.) [in Danish]. Copenhagen, Denmark: Samfundslitteratur.

Brown, J. S., & Gray, E. S. (1995). *The people are the company.* Retrieved January 11, 2005, from http://www.fastcompany.com/magazine/01/people.html

Checkland, P., & Scholes, J. (1990). *Soft systems methodology in action.* Toronto, Canada: John Wiley and Sons.

Comas, J., & Sieber, S. (2001). Connecting knowledge management and experiential learning to gain new insights and research perspectives. In *Proceeding of the ECIS 2001,* Bled, Slowenia.

Cook, S. D. N., & Brown, J. S. (1999). Bridging epistemologies: The generative dance between organisational knowledge and organisational knowing. *Organisational Science, 4,* 381-400.

Dahlbom, B., & Mathiassen, L. (1993). *Computers in context—The philosophy and practice of systems design*. Cambridge, UK: Blackwell.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532-550.

Fiske, J. (1990). *Introduction to communication studies* (2nd ed.). London: Routledge.

Hansen, M. T., Nohria, N., & Tierney, T. (1999). What is your strategy for managing knowledge? *Harvard Business Review, 77*(2), 106-116.

Hughes, J., & Jones, S. (2003). Reflections on the use of grounded theory in interpretive information systems research. In *Proceedings of the ECIS 2003 Conference,* Naples, Italy.

Jennex, M. E., & Olfman, L. A. (2005). Assessing knowledge management success. *International Journal of Knowledge Management, 1*(2), 33-49.

Jennex, M. E., & Olfman, L. A. (2006). A Model of knowledge management success. *International Journal of Knowledge Management, 2*(3), 51-68.

Kautz, K., & Thaysen, K. (2001). Knowledge, learning and IT Support in a small software company. *Journal of Knowledge Management, 5*(4), 349-357.

Klein, H., & Myers, M. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly, 23*(1), 67-93.

Kolb, D. A. (1984). *Experiential learning—Experience as source of learning and development.* New Jersey: Prentice Hall.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation.* Cambridge, UK: Cambridge University Press.

Nielsen, J. (1994). Ways to Knowledge [in Danish]. In M. Brorup, L. Hauge, & U.L. Thomsen (Eds.). *Pieces of psychology* (pp. 65-86). Copenhagen, Denmark: Gyldendah.

Nielsen, K., & Kvale, S. (1999). Master-apprenticeship learning: Learning as social practice [in Danish]. Copenhagen, Denmark: Hans Reitzel.

Nonaka, I. (1994). A dynamic theory of organisational knowledge creation. *Organisation Science, 5*(1), 14-37.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company.* Oxford, UK: Oxford University Press.

Polanyi, M. (1966). *The tacit dimension.* London: Routledge.

Severin, W., & Tankard, J. (1997). *Communication theories* (4th ed.). New York: Longman.

Shannon, C., & Weaver, W. (1949). *The mathematical theory of communication.* Urbana: University of Illinois Press.

Smolnik, S., Kremer, S., & Kolbe, L. (2005). Continum of context explication: Knowledge discovery through process-oriented portals. *International Journal of Knowledge Management, 1*(1), 27-46.

Stenmark, D. (2000). Leveraging tacit organisational knowledge. *Journal of Management Information Systems, 17*(3), 9-24.

Swan, J., Scarbrough, H., & Preston, J. (1999). Knowledge management—The next fad to forget people. In *Proceedings of the ECIS 1999,* Copenhagen, Denmark.

Thompson, M., & Walsham, G. (2001). Learning to value the Bardi tradition: Culture, communication, and organisational knowledge. In *Proceeding of the ECIS 2001,* Bled, Slowenia.

Tsoukas, H. (1996). The firm as a distributed knowledge system: A constructionist approach [Winter special issue]. *Strategic Management Journal, 7,* 11-25.

Tsoukas, H. (1998). The word and the world: A critique of representationalism in management research. *International Journal of Public Administration, 21*(5), 781-817.

Walsh, J. P., & Ungson, G. R. (1991). Organisational memory. *Academy of Management Review, 16,* 57-91.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity.* Cambridge, UK: Cambridge University Press.

Wenger, E. C., McDermott, R., & Snyder, W. M. (2002). *Communities of practice—A guide to managing knowledge.* Boston: Harvard Business School.

Wenger, E. C., & Snyder, W. M. (2000). Communities of practice: The organisational frontier. *Harvard Business Review, 78,* 139-145.

This work was previously published in the International Journal of Knowledge Management, edited by M. Jennex, Volume 3, Issue 2, pp. 91-117, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 4.21
Digital Library Structure and Software

Cavan McCarthy
Louisiana State University, USA

INTRODUCTION

Digital libraries (DL) can be characterized as the "high end" of the Internet, digital systems which offer significant quantities of organized, selected materials of the type traditionally found in libraries, such as books, journal articles, photographs and similar documents (Schwartz, 2000). They normally offer quality resources based on the collections of well-known institutions, such as major libraries, archives, historical and cultural associations (Love & Feather, 1998). The field of digital libraries is now firmly established as an area of study, with textbooks (Arms, 2000; Chowdhury & Chowdhury, 2003; Lesk, 1997); electronic journals from the US (D-Lib Magazine: http://www.dlib.org/) and the UK (Ariadne: http://www.ariadne.ac.uk/); even encyclopedia articles (McCarthy, 2004).

BACKGROUND

Digital libraries require appropriate presentation and careful logical organization to make them easily accessible, but arrangements typical of Web systems are inadequate for them. The classic Web structure, where random links can be created between any pair of pages, is not appropriate to highly organized data. The other classic arrangement is the tree or directory structure often found in computerized systems, where the user starts from a "trunk" or "root directory" and goes to a branch, then a subdivision of that branch. This is effective for individual images, but is inadequate for navigating sequential pages, as in a digital library system presenting lengthy texts. Before discussing the different software solutions available, it is useful to review the principle types of digital material currently offered by digital libraries.

DIGITAL LIBRARY MATERIALS

At this time digital library resources can be divided into three categories: images, texts and other resources:

Images

Image access is used for individual visual resources, such as photographs, posters, drawings, etc. The classic procedure uses a series of three types of image. Scanning produces a high-quality archive image, which is then used to generate an access image, for general public use. Finally, a small thumbnail image is produced, for quick reference (Boss, 2001; Lee, 2001). In more detail:

Archive Image

A high-quality image, scanned directly from the original, destined for long-term preservation. Normally an uncompressed TIF (Tagged Image File Format) image is used here; TIFs offer the highest quality images and a resolution of 600 dpi (dots per inch) is standard. As scanning is an expensive operation, which exposes original materials to possible damage, the archive image will be carefully preserved. It must always exist at the system level, but is not necessarily available to the end-user. TIF files occupy significant server space and imply lengthy download times. Another factor is that some DL will want to sell their own hard-copy prints of quality images.

Access Image or Working Image

A quality image, adequate for consultation and serious study by digital library users. This is normally a high-quality JPG (Joint Photographic Experts Group) image, generated from the archival TIF. JPG files are widely used on the Internet and offer quality spatial and color reproduction and a high compression ratio. For DL purposes JPG images will often be generated at a resolution of 300 dpi; a size of 640x480 pixels is also common.

Thumbnail Image

A small reference image, which gives the user a general idea of the Access image, before downloading that image. Typically a medium to low quality JPG, generated from the Access image, but about one-tenth of its size, and commonly produced at a resolution of 72 dpi. GIF format (Graphic Interchange Format) can also be used for thumbnails (Arizona, 2000; Western, 2003).

Text

Multi-page text documents, such as books, or journal articles require special procedures. Numerous options are possible and the principle alternatives for input, simple text presentations and pagination will be examined in turn; the earliest procedures will be discussed first.

Text Input

Manual keyboarding was originally adopted by Project Gutenberg, the first significant text-oriented digital library (http://promo.net/pg/), founded in 1971. This is a laborious process which severely limits productivity, and is now rarely used.

OCR (Optical Character Recognition) software is now routinely used to scan text into digital libraries. OmniPage Pro (http://www.scansoft.com/omnipage/) or the Russian software ABBYY (http://www.abbyy.com/) are frequently cited in the digital library context. OCR text requires careful revision, because even 99.99% accuracy means that there will be one mistake every couple of pages, but only a person fully conversant with the literature will be able to identify errors at this level. Many digital library texts are older books whose ornate type faces or soiled pages can generate additional OCR errors.

Simple Text Presentations

"Plain-vanilla" ASCII texts were the original basis of Project Gutenberg (http://promo.net/pg/). These TXT files can be used by a variety of computing platforms, but are suited to a word processing, rather than an Internet environment.

HTML text is now firmly established for textual work in digital libraries, including Project Gutenberg. All Internet users are familiar with HTML, files need not be much larger than TXT files, it is easy to present crisp black text on white background, text size can be quickly adjusted on computer screens, HTML can be indexed, readers can easily take extracts from the text, and, within the constraints of ethics and plagiarism, manipulate them and insert them in other documents.

The entire text, without page divisions, is normally delivered by Project Gutenberg as an ASCII or HTML file. Entire chapters or lengthy blocks of text are supplied by other digital libraries, such as Bartleby.com (http://www.bartleby.com/), another pioneering system, which has been offering free copies of classic books since 1963.

Disadvantages of HTML are that it does not necessarily communicate the original text in an integral manner: the "look and feel" may be different, transcription errors may occur and it may not accurately reproduce a combination of text and images.

Paginated Text

Pagination is typical of the traditional book, and most sophisticated digital library software now creates easily-navigated page-by-page sequences. The reader wants above all to go to the next page, also to go back a page when necessary, using appropriate icons or buttons.

JPG images reproduce individual pages exactly, without scanning or transcription errors, making them suitable for research. JPGs are widely used on the Internet and are easy to generate, download, and print; they are also appropriate for mixed text and images. But small size text, older typefaces and brown, mottled or foxed paper may reduce legibility, while readers cannot easily take extracts from a JPG document.

Adobe Acrobat's PDF (Portable Document File) format offers advantages similar to JPGs, such as quality reproduction of the original in relatively small files, excellent enlargement of the text and easy combination of text with images. Content producers can adjust the security settings within Acrobat to constrain the end user's interaction with the document. The digital library needs to purchase Adobe Acrobat (http://www.adobe.com/products/acrobat/main.html) to create PDFs, but end-users can easily download the free Adobe Reader.

Digital libraries offering access to texts therefore have two principle possibilities, text, typically delivered in HTML, or page images, which typically use JPGs. In fact, these are not alternatives, they are complementary approaches. The image offers a reliable reproduction of the text, while HTML may be easier for the end-user. Intellectual property and copyright issues may also impact the decision to present either HTML text or images. Even if only images are made available to the end-user, it will be necessary to create a text version in order to generate an index. Word-by-word indexing is a major advantage offered by digital libraries, which cannot be matched by traditional libraries.

For examples of sophisticated paginated access, browse the 8,500-volume Making of America collection from the University of Michigan at http://www.hti.umich.edu/cgi/t/text/text-idx?tpl=browse.tpl&c=moa&cc=moa. This uses DLXS software, discussed in more detail below. The Digital Quaker Collection (http://esr.earlham.edu/dqc/) contains full text and page images of over 500 Quaker works from the 17th and 18th centuries, generated using eXist software (see below for details).

The three-level imaging sequence, Archive, Access, and Thumbnail, described in Image ac-

cess above, is of lesser relevance to text systems. It is still possible to use TIF format to produce the Archive copy, because this creates a high quality preservation file in a non-commercial format. But the TIF will rarely be available to the reader in a text-based system. TIF files are slow to download and text, even with images, can be reproduced adequately using JPGs or PDFs. Thumbnail images of a series of book pages will normally appear identical to the reader, thus adding little to the digital library, and are therefore often omitted.

Location within physical books is not a problem; it is easy to see when half the text has been read, or to flip back to the table of contents. In digital books readers need a clear indication of location in relation to the text as a whole, via page numbering or by a link to a table of contents or chapter list. If the text has notes or references, it will be necessary to link to them, then jump back to the main text. It will also be necessary to link to a full bibliographic citation.

Other Resources

Collections of maps and aerial photographs need to offer spatially-oriented access, so that users can pan or zoom. For an example, see the collection of 50,000 old aerial photographs of the State of Georgia, USA, at http://dbs.galib.uga.edu/gaph/html/. Collections of 3-dimensional objects may need to be viewed from various angles. See http://www.lunaimaging.com/insight/featuretour/multiview.html for an example using Luna Imaging software (discussed in more detail below). Sound and moving images are still relatively rare in digital libraries, for examples see History and Politics Out Loud: a searchable archive of politically significant audio materials from NorthWestern University, USA (http://www.hpol.org/) or the Internet Archive at http://www.archive.org/movies/index.html.

Metadata

Digital resources need to be adequately described. Historical photographs, for example, are only useful if location, date, subject, etc., have been identified and input to the system. This additional information is known as metadata and must be added by trained professionals. There are various types of metadata:

- **Descriptive metadata:** creator, title, subject, etc.
- **Structural metadata:** information on the internal organization of the digital resource, such as the chapters in a book.
- **Administrative metadata:** management information, including technical information, such as scanning resolution, hardware and software used, also rights management information and restrictions on use.

Access

Initial access to a digital library is through a gateway, which needs to be attractive and easy to use. Many digital libraries, especially those sponsored by educational institutions, offer free and immediate access to their materials. Commercial digital libraries, such as e-journal and paid e-book collections, embed security procedures, such as IP recognition or password control, into their gateways. Once in the digital library, the user is normally offered a choice between browse and search access.

- **Browse access** or directory-based systems take the user to a menu of choices, such as an alphabetically organized list of authors or titles, a categorized list by subject, geographic location, specific collection, etc. Menus can be nested, subdividing broad subjects. This is the easiest access to orga-

nize (it can even be created manually), and is offered by almost all digital libraries, normally from the opening page.

- **Search access** involves software, which is used to create an index. The user queries the index via a search box, usually a prominent feature of the digital library.

See, for example, The American Memory collections of the Library of Congress, which offer both browse access (via a "Collection Finder") and search access prominently from the opening page (http://www.memory.loc.gov).

Boolean search (AND, OR, NOT) is common in digital libraries. For a sophisticated search of the documents of the Chicago Women's Liberation Union, see http://cwluherstory.master.com/texis/master/search/?q=CWLU&s=SS&cmd=Options. This uses indexing software from Master.com (http://www.master.com/texis/master/app/home.html). A help file will often be available, to familiarize the user with search techniques, which can vary considerably between systems. The State Library of New South Wales offers a good example at http://www.sl.nsw.gov.au/search/guide.cfm. The user will also want to know exactly what is being searched: is it the full text of the document? Or the text of the document, but excluding common words? Or the search may be limited to metadata elements, such as author, title and abstract. This is not always clearly stated. Whatever the parameters, the use of a computer-produced index implies a higher level of automated support than the browse or directory systems cited previously. Search access is therefore not quite as common as directory access.

Efficient retrieval implies both types of access. These are also the principal options for retrieving information from the WWW: browse access is available from directory-based systems such as that which forms the basis of Yahoo! and the Open Directory, while search is offered by search engines, such as Google.

DIGITAL LIBRARY SOFTWARE

It is possible to create small-scale digital collections by hand, using standard HTML editing or open-source software. But, due to the complexity and variety of the field, purpose-designed software is now in common use. In alphabetical order, principle options are:

CONTENTdm Digital Media Management Software Suite

A high-performance storage and retrieval software for multimedia collections which is rapidly gaining acceptance (http://contentdm.com/index.html). Developed at the University of Washington, it is offered to libraries, museums, and nonprofit archives by OCLC, the major supplier of bibliographic data to libraries (http://www.oclc.org/contentdm/default.htm). Collections relevant to regional studies include the Louisiana Digital Library (http://www.lsu.edu/diglib) and Early Las Vegas from the University of Nevada, Las Vegas (http://www.library.unlv.edu/early_las_vegas/index.html).

DLXS

Offered by the University of Michigan Digital Library eXtension Service. It is a comprehensive suite of software tools, especially suited to indexing and presenting multi-page documents (http://www.dlxs.org/). It has already been mentioned as the basis for Michigan's The Making of America collection with approximately 8,500 books and 50,000 journal articles from the antebellum period through reconstruction (http://www.hti.um ich.edu/cgi/t/text/text-idx?tpl=browse.tpl&c=moa&cc=moa). A further 3,000 books are in the Wright American Fiction collection 1851-1875 (http://www.letrs.indiana.edu/web/w/wright2/).

DSpace

A "Durable Digital Depository," designed to capture, distribute and preserve the intellectual output of universities and similar institutions (http://dspace.org/). A joint project of MIT Libraries and the Hewlett-Packard Company, the original DSpace can be searched at https://dspace.mit.edu/index.jsp; full-text is in PDF. An open-source software, DSpace can be freely downloaded from http://sourceforge.net/projects/dspace/ (Atwood, 2002; Carnevale, 2003).

eXist

An open source XML database created in Germany (http://exist.sourceforge.net or http://exist-db.org/). It was used for one of most ambitious digitization projects of recent years, The Proceedings of the Old Bailey, 1674 to 1834 (http://www.oldbaileyonline.org/). This offers access to the largest body of texts detailing the lives of non-elite people ever published, accounts of over 100,000 criminal trials held at London's central criminal court.

Greenstone Digital Library Software

An open source software suite, developed in New Zealand at the University of Waikato (http://www.greenstone.org/). It automatically creates organized collections of digitized documents with a standardized interface, with automatic full-text indexing, listing of titles, etc. (Witten & Bainbridge, 2002). It handles large collections of documents, in a variety of formats, works on a server or desktop, and exports to CD-ROM. It is used for the New Zealand Digital Library (http://www.nzdl.org/), notable for humanitarian and UN collections in a variety of languages and scripts, including Chinese and Arabic. It is distributed free under GNU Public License, and can be downloaded from http://www.greenstone.org/english/download.html. All texts in Project Gutenberg can be searched via Greenstone at http://public.ibiblio.org/gsdl/cgi-bin/library.cgi?a=p&p=about&c=gberg.

Hyperion Digital Media Archive

A tool for organizing, full-text indexing, storing and accessing of non-book holdings in a digital format (http://www.sirsi.com/Sirsiproducts/hyperion.html). It was produced by Sirsi, a Huntsville, Alabama, company which supplies automated cataloging and circulation systems to libraries of all types. An important application is the Civil Rights in Mississippi Digital Archive of resources on race relations, created by the University of Southern Mississippi (http://www.lib.usm.edu/~spcol/crda/).

Insight Image Management and Delivery System

Produced by Luna Imaging, a joint enterprise of the J. Paul Getty Trust, California, and Eastman Kodak (http://www.luna-imaging.com/). This high quality imaging software, notable for its powerful zoom capability, is used for works of art, photographs and maps. An example is the David Rumsey Historical Map Collection (http://www.davidrumsey.com/), which features over 8,000 maps and won a Webby Award in 2002.

Olive ActivePaper Archive Software

Permits historical newspapers to be scanned from the originals, or from microfilm, creating fully searchable digital collections. Readers are able to click on any article, photograph or advertisement, which is then enlarged and presented in a separate window. For a sample, see the Missouri Historical Newspapers Collection (http://newspapers.umsystem.edu/archive/Skins/Missouri/navigator.asp). OCLC, the major supplier of bibliographic data to libraries, offers Olive to libraries (http://www.oclc.org/middleeast/en/olive/about/); Olive

Software is a Denver company (http://www.olivesoftware.com/home.html).

FUTURE TRENDS

The following future developments can be forecast with confidence:

- **More standardization:** the current period is one of experimentation, but standardized solutions will rapidly become established. Cross-system searching and access will be major considerations in the future.
- **More access to audio, film and video:** resources are still thin in this area, but more sophisticated software and increased bandwidth will make much more material available.

CONCLUSION

Digital libraries will have a bright future in a world which is ever more reliant on electronic access to information to guarantee social integration, progress and the end of the digital divide. The increasing availability of special purpose software will facilitate digital library production. Open source software such as Greenstone, which is available free of charge, will be of great value for regional community digitization projects.

REFERENCES

Arizona State Library, Archives and public records. (2000). *Digitization guidelines.* Retrieved September 9, 2004, from *http://www.lib.az.us/digital/dg_a4.html*

Arms, W. Y. (2000). *Digital libraries.* (Digital Libraries and Electronic Publishing). Cambridge, MA: MIT Press.

Atwood, S. (2002). MIT's Superarchive: A digital repository will revolutionize the way research is shared and preserved. *Technology Review, 104*(12). Retrieved September 9, 2004, from *http://www.technology review.com/articles/atwood1202.asp*

Boss, R, W. (2001). Imaging for libraries and information centers. *Library Technology Reports, 37*(1), 1-59.

Carnevale, D. (2003). Six institutions will help fine-tune a popular new archiving program. *The Chronicle of Higher Education, 49*(23), 36. Retrieved September 9, 2004, from *http://chronicle.com/free/2003/01/2003013001t.htm*

Chowdhury, G. G., & Chowdhury, S. (2003). *Introduction to digital libraries.* New York: Neal-Schuman Publishers.

Lee, S. D. (2001). *Digital imaging: A practical handbook*. New York: Neal-Schuman Publishers.

Lesk, M. (1997). *Practical digital libraries: Books, bytes, and bucks.* (The Morgan Kaufmann Series in Multimedia Information and Systems). San Francisco: Morgan Kaufmann Publishers.

Love, C., & Feather, J. (1998). Special collections on the World Wide Web: A survey and evaluation. *Journal of Librarianship and Information Science, 30*(4), 215-222.

McCarthy, C. (2004). Digital libraries. In Hossein Bidgoli, *The Internet encyclopedia* (vol. 1, A-F, pp. 505-525). New York: John Wiley & Sons.

Schwartz, C. S. (2000). Digital libraries: An overview. *Journal of Academic Librarianship, 26*(6), 385-393.

Waters, D. J. (1998). What are digital libraries? *CLIR Issues* (4). Retrieved September 9, 2004, from *http://www.clir.org/pubs/issues/issues04.html#dlf*

Western states digital imaging best practices, Version 1.0. (2003). (41 p). Denver: Western States Digital Standards Group; University of Denver; Colorado Digitization Program. Retrieved September 9, 2004, from *http://www.cdpheritage.org/resource/scanning/documents/WSDIBP_v1.pdf*

Witten, I. H., & Bainbridge, D. (2002). *How to build a digital library*. San Francisco: Morgan Kaufmann Publishers.

KEY TERMS

ASCII: American Standard Code for Information Interchange-Codification system used to convert simple text to computer readable form.

Digital Library: Provides the resources, including the specialized staff, to select, structure, offer intellectual access to, interpret, distribute, preserve the integrity of, and ensure the persistence over time of collections of digital works so that they are readily and economically available for use by a defined community or set of communities (Waters, 1998).

JPEG: Joint Photographic Experts Group-Specification for reproduction of digital images, widely used on the Internet.

Metadata: Data about data,surrogates or descriptions of data, key words or codes used as systematic keys to the content of digital objects, such as Web pages or digital images.

OCR: Optical Character Recognition-Software that recognizes text and converts it to a form in which it can be processed by computer.

PDF: Portable Document Format-Method of document reproduction, generated by Adobe Acrobat Systems, notable for quality reproduction of both text and accompanying images.

TIFF: Tagged Image File Format-High quality image reproduction format, typically generates large files used for archival purposes.

This work was previously published in Encyclopedia of Developing Regional Communities with Information and Communication Technology, edited by S. Marshall, W. Taylor & X. Yu, pp. 193-198, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 4.22
Comparing Four-Selected Data Mining Software

Richard S. Segall
Arkansas State University, USA

Qingyu Zhang
Arkansas State University, USA

INTRODUCTION

This chapter discusses four-selected software for data mining that are not available as free open-source software. The four-selected software for data mining are SAS® Enterprise Miner™, Megaputer PolyAnalyst® 5.0, NeuralWare Predict® and BioDiscovery GeneSight®, each of which was provided by partnerships with our university. These software are described and compared by their existing features, characteristics, and algorithms and also applied to a large database of forest cover types with 63,377 rows and 54 attributes. Background on related literature and software are also presented. Screen shots of each of the four-selected software are presented, as are future directions and conclusions.

BACKGROUND

Historical Background

Han and Kamber (2006), Kleinberg and Tardos (2005), and Fayyad et al. (1996) each provide extensive discussions of available algorithms for data mining.

Algorithms according to StatSoft (2006b) are operations or procedures that will produce a particular outcome with a completely defined set of steps or operations. This is opposed to heuristics that according to StatSoft (2006c) are general recommendations or guides based upon theoretical reasoning or statistical evidence such as "data mining can be a useful tool if used appropriately."

The Data Intelligence Group (1995) defined data mining as the extraction of hidden predictive

information form large databases. According to The Data Intelligence Group (1995), "data mining tools scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations."

Brooks (1997) describes rules-based tools as opposed to algorithms. Witten and Frank (2005) describe how data mining algorithms work including covering algorithms, instance-based learning, and how to use the WEKA, an open source data mining software that is a machine learning workbench.

Segall (2006) presented a chapter in the previous edition of this Encyclopedia that discussed microarray databases for biotechnology that included a extensive background on microarray databases such as that defined by Schena (2003), who described a microarray as "an ordered array of microscopic elements in a planar substrate that allows the specific binding of genes or gene products." The reader is referred to Segall (2006) for a more complete discussion on microarray databases including a figure on the overview of the microarray construction process.

Piatetsky-Shapiro (2003) discussed the challenges of data mining specific to microarrays, while Grossman et al. (1998) reported about three NSF (National Science Foundation) workshops on mining large massive and distributed data, and Kargupta at al. (2005) discussed the generalities of the opportunities and challenges of data mining.

Segall and Zhang (2004, 2005) presented funded proposals for the premises of proposed research on applications of modern heuristics and data mining techniques in knowledge discovery whose results are presented as in Segall and Zhang (2006a, 2006b) in addition to this chapter.

Software Background

There is a wealth of software today for data mining such as presented in American Association for Artificial Intelligence (AAAI) (2002) and Ducatelle (2006) for teaching data mining, Nisbet (2006) for CRM (Customer Relationship Management) and software review of Deshmukah (1997). StatSoft (2006a) presents screen shots of several softwares that are used for exploratory data analysis (EDA) and various data mining techniques. Proxeon Bioinformatics (2006) manufactures bioinformatics software for proteomics the study of protein and sequence information.

Lazarevic et al. (2006) discussed a software system for spatial data analysis and modeling. Leung (2004) compares microarray data mining software.

National Center for Biotechnology Information (NCBI) (2006) provides tools for data mining including those specifically for each of the following categories of nucleotide sequence analysis, protein sequence analysis and proteomics, genome analysis, and gene expression. Lawrence Livermore National Laboratory (LLNL) (2005) describes their Center for Applied Scientific Computing (CASC) that is developing computational tools and techniques to help automate the exploration and analysis of large scientific data sets.

MAIN THRUST

Algorithms of Four-Selected Software

This chapter specifically discusses four-selected data mining software that were chosen because these software vendors have generously offered their services and software to the authors at academic rates or less for use in both the classroom and in support of the two faculty summer research grants awarded as Segall and Zhang (2004, 2005).

SAS Enterprise Miner™ is a product of SAS Institute Inc. of Cary, NC and is based on the SEMMA approach that is the process of Sampling (S), Exploring (E), Modifying (M), Modeling (M), and Assessing (A) large amounts of data. SAS

Enterprise Miner™ utilizes a workspace with a drop-and-drag of icons approach to constructing data mining models.

SAS Enterprise Miner™ utilizes algorithms for decision trees, regression, neural networks, cluster analysis, and association and sequence analysis.

PolyAnalyst® 5 is a product of Megaputer Intelligence, Inc. of Bloomington, IN and contains sixteen (16) advanced knowledge discovery algorithms as described in Table 1 that was constructed using its User Manual by Megaputer Intelligence Inc. (2004; p. 163, p. 167, p.173, p.177, p. 186, p.196, p.201, p.207, p. 214, p. 221, p.226, p. 231, p. 235, p. 240, p.263, p. 274.).

NeuralWorks Predict® is a product of NeuralWare of Carnegie, PA. This software relies on neural networks, According to NeuralWare (2003, p.1):

"One of the many features that distinguishes Predict® from other empirical modeling and neural computing tools is that is automates much of the painstaking and time-consuming process of selecting and transforming the data needed to build a neural network."

Table 1. Description of data mining algorithms for PolyAnalyst® 5

Data Mining Algorithm	Underlying Algorithms
1. Discriminate	1. (a.) Fuzzy logic for classification 1. 1. (b.) Find Laws, PolyNet Predictor, or Linear Regression
2. Find Dependencies	2. ARNAVAC [See Key Terms]
3. Summary Statistics	3. Common statistical analysis functions
4. Link Analysis (LA)	4. Categorical, textual and Boolean attributes
5. Market and Transactional Basket Analysis	5. PolyAnalyst Market Basket Analysis
6. Classify	6. Same as that for Discriminate
7. Cluster	7. Localization of Anomalies Algorithm
8. Decision Forest (DF)	8. Ensemble of voting decision trees
9. Decision Tree	9. (a.) Information Gain splitting criteria (b.) Shannon information theory and statistical significance tests.
10. Find Laws	10. Symbolic Knowledge Acquisition Technology (SKAT) [See Key Terms]
11. Nearest Neighbor	11. PAY Algorithm
12. PolyNet Predictor	12. PolyNet Predictor Neural Network
13. Stepwise Linear Regression	13. Stepwise Linear Regression
14. Link Terms (LT)	14. Combination of Text Analysis and Link Analysis algorithms
15. Text Analysis (TA)	15.Combination of Text analysis algorithms augmented by statistical techniques
16. Text Categorization (TC)	16. Text Analysis algorithm and multiple subdivision splitting of databases.

NeuralWorks Predict® has a direct interface with Microsoft Excel that allows display and execution of the Predict® commands as a drop-down column within Microsoft Excel.

GeneSight™ is a product of BioDiscovery, Inc. of El Segundo, CA that focuses on cluster analysis using two main techniques of hierarchical and partitioning both of which are discussed in Prakash and Hoff (2002) for data mining of microarray gene expressions.

Both SAS Enterprise Miner™ and PolyAnalyst® 5 offer more algorithms than either GeneSight™ or NeuralWorks Predict®. These two software have algorithms for statistical analysis, neural networks, decision trees, regression analysis, cluster analysis, self-organized maps (SOM), association (e.g. market-basket) and sequence analysis, and link analysis. GeneSight™ offers mainly cluster analysis and NeuralWorks Predict® offers mainly neural network applications using statistical analysis and prediction to support these data mining results. PolyAnalyst® 5 is the only software of these that provides link analysis algorithms for both numerical and text data.

Applications of the Four-Selected Software to Large Database

Each of the four-selected software have been applied to a large database of forest cover type that is available on the same website of the Machine Learning Repository at the University of California at Irvine by Newman et al. (1998) for which results are shown in Segall and Zhang (2006a, 2006b) for different datasets of numerical abalone fish data and discrete nominal-valued mushroom data.

The forest cover type's database consists of 63,377 records each with 54 attributes that can be used to as inputs to predictive models to support decision-making processes of natural resource managers. The 54 columns of data are composed of 10 quantitative variables, 4 binary variables for wilderness areas, and 40 binary variables of soil types. The forest cover type's classes include Spruce-Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-Fir, Krummholz, and other.

The workspace of SAS Enterprise Miner™ is different than the other software because it uses icons that are user-friendly instead of only using spreadsheets of data. The workspace in SAS Enterprise Miner™ is constructed by using a drag-and-drop process from the icons on the toolbar which again the other software discussed do not utilize.

Figure 1 shows a screen shot of cluster analysis for the forest cover type data using

SAS Enterprise Miner™. From Figure 1 it can be seen using a slice with a standard deviation measurement, height of frequency, and color of the radius that this would yield only two distinct clusters: one with normalized mean of 0 and one with normalized mean of 1. If different measure of measurement, different slice height, and different key for color were selected than a different cluster figure would have resulted.

A screen shot of PolyAnalyst® 5.0 showing the input data of the forest cover type data with attribute columns of elevation, aspect, scope, horizontal distance to hydrology, vertical distance to hydrology, horizontal distance to roadways, hillshade 9AM, hillshade Noon, etc. PolyAnalyst® 5.0 yielded a classification probability of 80.19% for the forest cover type database.

Figure 2 shows the six significant classes of clusters corresponding to the six major cover types: Spruce-Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, and Douglas-Fir. One of the results that can be seen from Figure 2 is that the most significant cluster for the aspect variable is the cover type of Douglas-Fir.

NeuralWare Predict® uses a Microsoft Excel spreadsheet interface for all of its input data and many of its outputs of computational results. Our research using NeuralWare Predict® for the forest type cover data indicates an accuracy of 70.6% with 70% of the sample for training and 30% of

Figure 1. SAS Enterprise Miner™ screen shot of cluster analysis

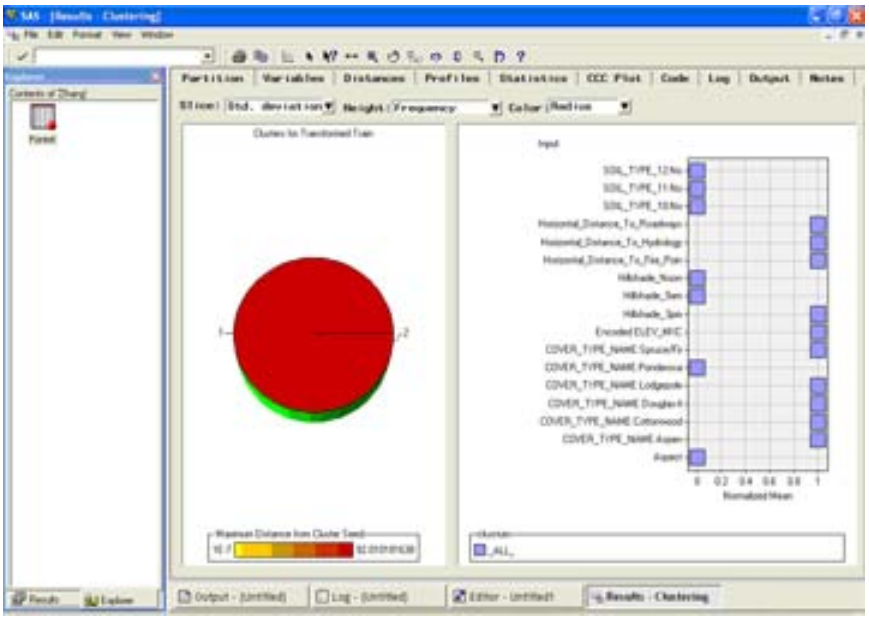


Figure 2. PolyAnalyst® 5.0 screen shot of cluster analysis

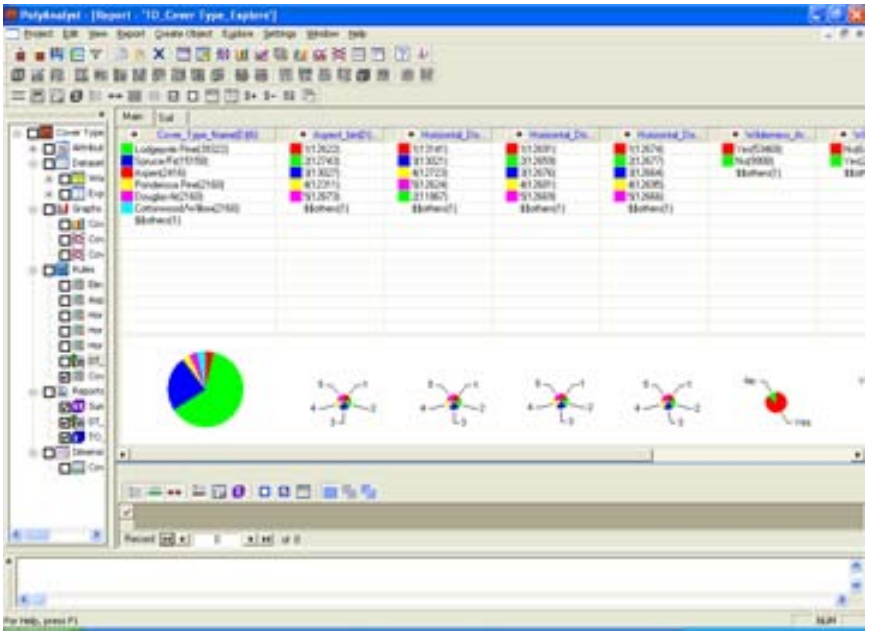


Figure 3. NeuralWare Predict® screen shot of complete neural network training (100%)

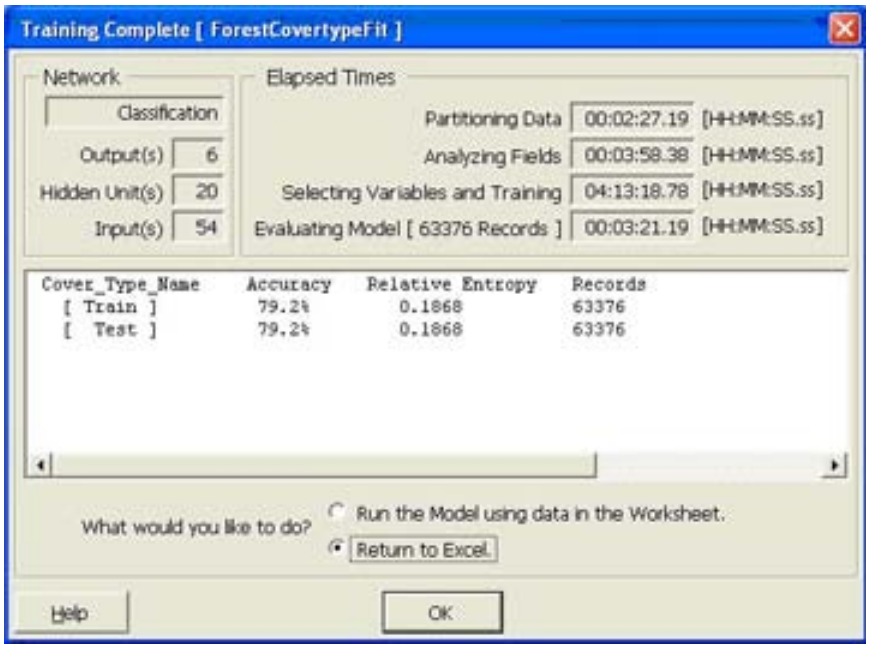
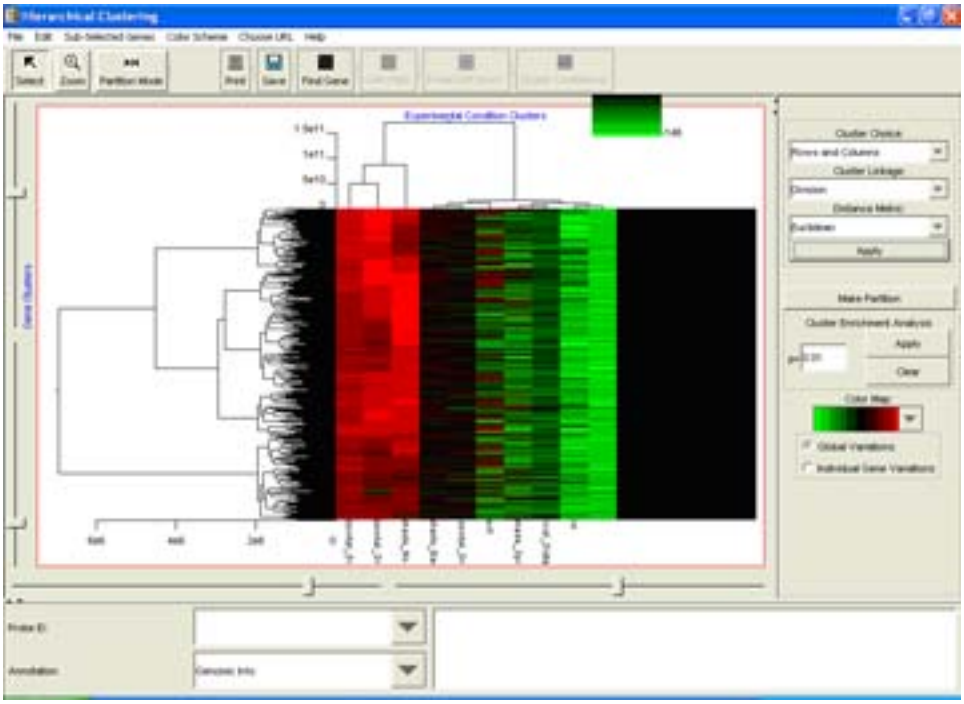


Figure 4. BioDiscovery GeneSight® screen shot of hierarchical clustering



the sample for test in 2 minutes and 26 seconds of execution time.

Figure 3 is a screen shot of NeuralWare Predict® for the forest type cover data that indicates that an improved accuracy of 79.2% can be obtained using 100% of the 63,376 records for both training and testing in 3 minutes 21 seconds of execution time in evaluating model. This was done to investigate what could be the maximum accuracy that could be obtained using NeuralWare Predict® for the same forest cover type database for comparison purposes to the other selected software.

Figure 4 is a screen shot using GeneSight® software by BioDiscovery Incorporated. That shows hierarchical clustering using the forest cover data set. As noted earlier, GeneSight® only performs statistical analysis and cluster analysis and hence no regression results for the forest cover data set can be compared with those of NeuralWare® Predict and PolyAnalyst®. It should be noted that from Figure 4 that the hierarchical clustering performed by GeneSight® for the forest cover type data set produced a multitude of clusters using the Euclidean distance metric.

FUTURE TRENDS

These four-selected software as described will be applied to a database that already has been collected of a different dimensionality. The database that has been presented in this chapter is of a forest cover type data set with 63,377 records and 54 attributes. The other database is a microarray database at the genetic level for a human lung type of cancer consisting of 12,600 records and 156 columns of gene types. Future simulations are to be performed for the human lung cancer data for each of the four-selected data mining software with their respective available algorithms and compared versus those obtained respectively for the larger database of 63,377 records and 54 attributes of the forest cover type.

CONCLUSION

The conclusions of this research include the fact that each of the software selected for this research has its own unique characteristics and properties that can be displayed when applied to the forest cover type database. As indicated, each software has it own set of algorithm types to which it can be applied. NeuralWare Predict® focuses on neural network algorithms, and Biodiscovery GeneSight® focuses on cluster analysis. Both SAS Enterprise Miner™ and Megaputer PolyAnalyst® employ each of the same algorithms except that SAS has a separate software SAS TextMiner® for text analysis. The regression results for the forest cover type data set are comparable for those obtained using NeuralWare Predict® and Megaputer PolyAnalyst®. The cluster analysis results for SAS Enterprise Miner™, Megaputer PolyAnalyst®, and Biodiscovery GeneSight® are unique to each software as to how they represent their results. SAS Enterprise Miner™ and NeuralWare Predict® both utilize Self-Organizing Maps (SOM) while the other two do not.

The four-selected software can also be compared with respect to their cost of purchase. SAS Enterprise Miner™ is the most expensive and NeuralWare Predict® is the least expensive. Megaputer PolyAnalyst® and Biodiscovery GeneSight® are intermediate in cost to the other two software.

In conclusion, SAS Enterprise Miner™ and Megaputer PolyAnalyst® offer the greatest diversification of data mining algorithms.

ACKNOWLEDGMENT

The authors want to acknowledge the support provided by a 2006 Summer Faculty Research Grant as awarded to them by the College of Business of Arkansas State University. The authors also want to acknowledge each of the four software manufactures of SAS, Megaputer Intelligence,

Inc., BioDiscovery, Inc., and NeuralWare, for their support of this research.

REFERENCES

AAAI (2002), American Association for Artificial Intelligence (AAAI) Spring Symposium on Information Refinement and Revision for Decision Making: Modeling for Diagnostics, Prognostics, and Prediction, Software and Data, retrieved from http://www.cs.rpi.edu/~goebel/ss02/software-and-data.html.

Brooks, P. (1997), Data mining today, *DBMS*, February 1997, retrieved from http://www.dbmsmag.com/9702d16.html.

Data Intelligence Group (1995), An overview of data mining at Dun & Bradstreet, *DIG White Paper 95/01*, retrieved from http://www.thearling.com.text/wp9501/wp9501.htm.

Deshmukah, A. V. (1997), Software review: ModelQuest Expert 1.0, *ORMS Today*, December 1997, retrieved from http://www.lionhrtpub.com/orms/orms-12-97/software-review.html.

Ducatelle, F., *Software for the data mining course*, *School of Informatics*, The University of Edinburgh, Scotland, UK, retrieved from http://www.inf.ed.ac.uk/teaching/courses/dme/html/software2.html.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996), From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, eds. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, 1–30. Menlo Park, Calif.: AAAI Press.

Grossman, R., Kasif, S., Moore, R., Rocke, D., & Ullman, J. (1998), *Data mining research: opportunities and challenges*, retrieved from http://www.rgrossman.com/epapers/dmr-v8-4-5.htm.

Han, J. & Kamber, M. (2006), *Data Mining: Concepts and Techniques*, 2nd edition, Morgan Kaufman, San Francisco, CA.

Kargupta, H., Joshi, A., Sivakumar, K., & Yesha, Y. (2005), *Data mining: Next generation challenges and future directions*, MIT/AAAI Press, retrieved from http:///www.cs.umbc.edu/~hillol/Kargupta/ngdmbook.html.

Kleinberg, J. & Tardos, E., (2005), *Algorithm Design*, Addison-Wesley, Boston, MA.

Lawrence Livermore National Laboratory (LLNL), The Center for Applied Scientific Computing (CASC), *Scientific data mining and pattern recognition: Overview*, retrieved from http://www.llnl.gov/CASC/sapphire/overview/html.

Lazarevic A., Fiea T., & Obradovic, Z., *A software system for spatial data analysis and modeling*, retrieved from http://www.ist.temple.edu?~zoran/papers/lazarevic00.pdf.

Leung, Y. F.(2004), *My microarray software comparison – Data mining software*, September 2004, Chinese University of Hong Kong, retrieved from http://www.ihome.cuhk.edu.hk/~b400559/arraysoft mining specific.html.

Megaputer Intelligence Inc. (2004), *PolyAnalyst 5 Users Manual*, December 2004, Bloomington, IN 47404.

Megaputer Intelligence Inc. (2006), *Machine learning algorithms*, retrieved from http://www.megaputer.com/products/pa/algorithms/index/php3.

Moore, A., *Statistical data mining tutorials*, retrieved from http://www.autonlab.org/tutorials.

National Center for Biotechnology Information (2006), National Library of Medicine, National Institutes of Heath, *NCBI tools for data mining*, retrieved from http://www.ncbi.nlm,nih.gov/Tools/.

NeuralWare (2003), NeuralWare Predict® The complete solution for neural data modeling: *Getting Started Guide for Windows*, NeuralWare, Inc., Carnegie, PA 15106

Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases*, Irvine, CA: University of California, Department of Information and Computer Science, http://www.ics.uci.edu/~mlearn/MLRepository.html

Nisbet, R. A. (2006), Data mining tools: Which one is best for CRM? Part 3, *DM Review*, March 21, 2006, retrieved from http://www.dmreview.com/editorial/dmreview/print_action.cfm?articleId=1049954.

Piatetsky-Shapiro, G. & Tamayo, P. (2003), Microarray data mining: Facing the challenges, *SIGKDD Exploration*, vo.5, n.2, pages 1-5, December, retrieved from http://portal.acm.org/citation.cfm?doid=980972.980974 and http://www.broad.mit.edu/mpr/publications/projects/genomics/Microarray_data_mining_facing%20_the_challenges.pdf.

Prakash, P. & Hoff, B. (2002) *Microarray gene expression data mining with cluster analysis using GeneSight™*, Application Note GS10, BioDiscovery, Inc., El Segundo, CA, retrieved from http://www.biodiscovery.com/index/cms-filesystem-action?file=AppNotes-GS/appnotegs10.pdf.

Proxeon Bioinformatics, *Bioinformatics software for proteomics, from proteomics data to biological sense in minutes*, retrieved from http://www.proxeon.com/protein-sequence-databases-software.html.

SAS® Enterprise Miner™, SAS Incorporated, Cary, NC, retrieved from http://www.sas.com/technologies/analytics/datamining/miner.

Schena, M. (2003*). Microarray analysis*, New York, John Wiley & Sons, Inc.

Segall, R.S. (2006), Microarray databases for biotechnology, *Encyclopedia of Data Warehousing and Mining*, John Wang, Editor, Idea Group, Inc., pp. 734-739.

Segall, R. S, & Zhang, Q. (2004). *Applications of modern heuristics and data mining techniques in knowledge discovery*, funded proposal submitted to Arkansas State University College of Business Summer Research Grant Committee.

Segall, R. S. & Zhang, Q. (2005). *Continuation of research on applications of modern heuristics and data mining techniques in knowledge discovery*, funded proposal submitted to Arkansas State University College of Business Summer Research Grant Committee.

Segall, R.S. & Zhang, Q. (2006a). Applications of neural network and genetic algorithm data mining techniques in bioinformatics knowledge discovery – A preliminary study, *Proceedings of the Thirty-seventh Annual Conference of the Southwest Decision Sciences Institute*, Oklahoma City, OK, v. 37, n. 1, March 2-4, 2006.

Segall, R. S. & Zhang, Q. (2006b). Data visualization and data mining of continuous numerical and discrete nominal-valued microarray databases for biotechnology, *Kybernetes: International Journal of Systems and Cybernetics*, v. 35, n. 9/10.

StatSoft, Inc. (2006a). *Data mining techniques*, retrieved from http://www.statsoft.com/textbook/stdatmin.html.

StatSoft, Inc. (2006b). *Electronic textbook*, retrieved from http://www.statsoft.com/textbook/glosa.html.

StatSoft, Inc. (2006c). *Electronic textbook*, retrieved from http://www.statsoft.com/textbook/glosh.html.

Tamayo, P. & Ramaswamy, S. (2002). *Cancer genomics and molecular pattern recognition*, Cancer Genomics Group, Whitehead Institute,

Massachusetts Institute of Technology, retrieved from http://www.broad.mit.edu/mpr/publications/projects/genomics/Humana_final_Ch_06_23_2002%20SR.pdf.

Witten, IH & Frank E. (2005). *Data mining: Practical machine learning tools and techniques with Java implementation*, Morgan Kaufman.

KEY TERMS

Algorithm: That which produces a particular outcome with a completely defined set of steps or operations.

ARNAVAC: An underlying machine language algorithm used in PolyAnalyst® for the comparison of the target variable distributions in approximately homogeneously equivalent populated multidimensional hyper-cubes.

Association and sequence analysis: A data mining method that relates first a transaction and an item and secondly also examines the order in which the products are purchased.

BioDiscovery GeneSight®: A program for efficient data mining, visualization, and reporting tool that can analyze massive gene expression data generated by microarray technology.

Data Mining: The extraction of interesting and potentially useful information or patterns from data in large databases; also known as Knowledge Discovery in Data (KDD).

Link Analysis (LA): A technique used in data mining that reveals and visually represents complex patterns between individual values of all categorical and Boolean attributes.

Link Terms (LT): A technique used in text mining that reveals and visually represents complex patterns of relations between terms in textual notes.

Market and transactional basket analysis: Algorithms that examine a long list of transactions to determine which items are most frequently purchased together, as well as analysis of other situations such as identifying those sets of questions of a questionnaire that are frequently answered with the same categorical answer.

Megaputer PolyAnalyst® 5.0: A powerful multi-strategy data mining system that implements a broad variety of mutually complementing methods for the automatic data analysis.

Microarray Databases: Store large amounts of complex data as generated by microarray experiments (e.g. DNA)

NeuralWare NeuralWorks Predict®: A software package that integrates all the capabilities needed to apply neural computing to a wide variety of problems.

SAS® Enterprise Miner™: Software that that uses an drop-and-drag object oriented approach within a workspace to performing data mining using a wide variety of algorithms.

Symbolic Knowledge Acquisition Technology (SKAT): An algorithm developed by Megaputer Intelligence and used in PolyAnalyst® 5.0 that uses methods of evolutionary programming for high-degree rational expressions that can efficiently represent nonlinear dependencies.

WEKA: Open-source data mining software that is a machine learning workbench.

This work was previously published in Encyclopedia of Data Warehousing and Mining, Second Edition, edited by J. Wang, pp. 269-277, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.23
Dimensions of UML Diagram Use:
A Survey of Practitioners

Brian Dobing
University of Lethbridge, Canada

Jeffrey Parsons
Memorial University of Newfoundland, Canada

ABSTRACT

The UML is an industry standard for object-oriented software engineering. However, there is little empirical evidence on how UML is used. This article reports results of a survey of UML practitioners. We found differences in several dimensions of UML diagram usage on software development projects including; frequency, the purposes for which they were used, and the roles of clients/users in their creation and approval. System developers are often ignoring the "use case-driven" prescription that permeates much of the UML literature, making limited or no use of either use case diagrams or textual use case descriptions. Implications and areas requiring further investigation are discussed.

INTRODUCTION

The unified modeling language (UML) emerged in the mid-1990s through the combination of previously competing object-oriented analysis and design (OOAD) approaches (Booch, 1994; Jacobson, Christerson, Jonsson, & Overgaard, 1992; Rumbaugh, Blaha, Premerlani, Eddy et al., 1991), along with other contributions to modeling complex systems (e.g., Harel, 1987). Control over its formal evolution was placed in the hands of the Object Management Group, which recently oversaw a major revision to UML 2.0. The UML became widely accepted as the standard for OOAD soon after its introduction (Kobryn, 1999) and remains so today (Evermann & Wand, 2006). A large number of practitioner

articles and dozens of textbooks have been devoted to articulating various aspects of the language, including guidelines for using it. More recently, a substantial body of research on the UML has emerged, ranging from proposals for extending the language (Moore, 2001; Odell, Van Dyke, & Bauer, 2000) to ontological analysis of its modeling constructs (Evermann & Wand, 2001a, 2001b) to analysis of the language's complexity (Siau & Cao, 2001, 2002; Siau, Erickson, & Lee, 2005) and experiments that evaluate various aspects of the effectiveness of UML models (Burton-Jones & Weber, 2003, Burton-Jones & Meso, 2006).

The UML was not developed based on any theoretical principles regarding the constructs required for an effective and usable modeling language for analysis and design; instead, it arose from (sometimes conflicting) "best practices" in parts of the software engineering community (Booch, 1999; Booch, Rumbaugh, & Jacobson, 1999). This resulted in a language containing many modeling constructs, which has thus been criticized on the grounds that it is excessively complex (DeJong, 2006; Dori, 2002; Kobryn, 2002). But, at the same time, the UML has also been criticized for lacking the flexibility to handle certain modeling requirements in specific domains (Duddy, 2002). As a consequence, the UML has evolved to allow for the definition of "profiles" that have enabled domain specific languages (Cook, 2000; DeJong, 2006).

While the UML is intended to be "largely process-independent," some of the key originators recommend a use case-driven process (e.g., Booch et al., 1999, p.33). A majority of UML books since then have endorsed this view, and most contain at least some further prescriptions for applying the language in modeling (Larman, 2005; Schneider & Winters, 2001; Stevens & Pooley, 2000). As would be expected with a best practices approach, their prescriptions sometimes differ. While some accept the original view that only use case narratives (or, more simply, use cases) be used to verify requirements with users (Jacobson, Ericsson, & Jacobson, 1994), others explicitly or implicitly indicate that other UML diagrams can be used for this purpose, for example activity diagrams "can be safely shared with customers, even those unfamiliar with software engineering" (Schneider & Winters, 2001, p.67).

There are also differences in guidelines for using the language, and use case narratives in particular (Dobing & Parsons, 2000). This is not surprising since the official UML 2.0 documentation provides no guidance on Narrative format, stating only that "use cases are typically specified in various idiosyncratic formats such as natural language, tables, trees, etc" (Object Management Group, 2005, p.574).

Finally, when the use case-driven approach is used, concerns have been raised about the potential communication disconnect (Dobing & Parsons, 2000) that can occur when use cases are the primary communication tool among analysts and the clients or users on the project team while class diagrams play that role amoung analysts and programmers. While use case narratives have been found to be the most comprehensible artifact for managers, users and domain experts, they are the least comprehensible for designers and programmers (Arlow & Neustadt, 2004) when they require knowledge of the organizational context that programmers do not have. Conversely, class diagrams are highly comprehensible by programmers, but not clients or users (Arlow & Neustadt, 2004).

In view of these issues, it would not be surprising to find a variety of practices followed by UML practitioners. We believe understanding current practice can make an important contribution to both theoretical and applied research on UML. From a theoretical perspective, understanding how the language is used can support or challenge theoretical analyses of UML capabilities and deficiencies (Evermann & Wand, 2001a, 2001b). From a practical perspective, usage patterns can inform best practices.

However, to our knowledge, only two surveys have addressed the extent to which UML diagrams are used in practice (Grossman, Aronson, & McCarthy, 2005; Zeichick, 2002), and neither examined why analysts choose to use some diagrams and ignore others. (We are defining "UML diagram" to include use case narratives, even though they are generally used to describe use cases in text form.) This is particularly surprising in view of the explosion of academic interest in UML. Our research seeks to address this issue by surveying UML use in practice.

Our objective was to study three key dimensions of UML diagram usage: how often each diagram was being used, the reasons why analysts chose to use or avoid them (emphasizing their role in facilitating team communication), and the roles of clients/users in their creation and approval. Such an understanding can also support the development of theory to explain observed usage patterns. From a practical point of view, understanding how the language is used can help support its evolution. For example, if certain parts of the language are not widely used or seen as useful, further research is needed to understand why this is so, and may lead to evolution or elimination of those parts.

RESEARCH METHODOLOGY

The research began with participation in a local UML user group, along with mostly informal interviews of about a dozen UML practitioners (none belonging to that user group and most in different cities) and some of their clients. Their approaches to using UML all differed to some degree from each other, some substantially. Some of the differences can be attributed to situational factors. For example, one project began with the database, and the associated class diagram, already in place. In other projects, analysts took a use case-driven approach and relied on someone else to do the class diagram later. Some clients wrote most of the use cases themselves, while others reviewed them.

The level of use case modeling varied, even in systems of roughly the same size, from a small number (less than 20) of relatively short use cases to much larger sets of detailed use cases and scenarios (usually defined as paths through a use case illustrating its application to particular instances) that attempted to capture very complex rules and regulations. The use of other UML diagrams depended on the analyst's knowledge of how to use them, client requests (e.g., one client insisted on at least one activity diagram for every use case), system domain, and other factors. Some learned the UML by starting with only a few of the diagram types while others took a more ambitious approach and attempted to use them all.

To get a broader picture of UML usage, a Web survey was developed based on the preliminary interviews and a literature review. The survey contained 38 questions, many with multiple parts (e.g., a list of possible reasons for not using a particular UML diagram). Both the survey and this article use UML 1.5 terminology, such as "collaboration diagrams" rather than the newer "communication diagrams." The original survey was first reviewed by colleagues and then pretested with two people involved in the interviews and one who had not been. Minor wording changes were made to several questions as a result. The pretest data were retained because the changes made were consistent with what these subjects had in mind.

The survey was intended for the population of analysts familiar with object-oriented techniques and UML in particular. To obtain a sample of such analysts, the OMG was contacted and they agreed to support the project. Their members were informed by email of the survey and the OMG endorsement. A link to the survey was also provided from the main OMG Web page. OMG members were encouraged to share the link with others using the UML in their organizations. Subsequently, an invitation to participate in the survey

was posted to the comp.object Usenet newsgroup. No participation incentive was offered. Some limitations of this approach are discussed later. However, other researchers in this area (e.g., Johnson & Hardgrave, 1999; Grossman et al., 2005) have used similar methods due to the difficulty of finding more representative samples.

RESULTS

Almost 2,700 hits on the survey site were recorded during the survey period from March 21, 2003 to March 31, 2004. About half (1,369) provided no response to any item. After eliminating these responses along with test data, minimal responses, meaningless or invalid responses, and inappropriate respondents (primarily students), there were 284 usable responses. While these criteria are difficult to define precisely, invalid responses were easily identified in practice based on either meaningless numerical entries (e.g., 1 for all entries including budget, number of classes, etc.) or comments that showed the response was not serious. Any response that had meaningful comments was included, no matter how incomplete. The 284 analyzed responses either contained data on UML diagram usage (182) or reasons why the UML was not being used (102). Of the 182 analysts using UML diagrams, most (171) responded that they were using the UML while 11 indicated they were using some UML diagrams in conjunction with other modeling approaches.

Demographic Data

The survey gathered some data on respondent experience in IT, but did not ask about age, gender or nationality. Table 1 shows that respondents have a wide range of experience in the IT field, reporting up to 45 years and 200 projects. UML experience is understandably less. In all cases, the minimum value reported was zero except for years of experience in IT (two years) and all IT projects (three). While respondents report more project experience with UML than other object-oriented approaches, it represents less than a quarter of their projects and about a third

Table 1. Respondent experience in years and projects

	Mean	Median	Max	Std Dev	N
Yrs Experience IT	15.1	14.0	45	9.2	96
Yrs Experience OO Prog	8.4	7.5	25	5.1	95
Yrs Experience OOAD	7.4	6.0	25	4.7	95
Yrs Experience UML	4.7	5.0	10	2.4	101
Yrs Experience OO DB	2.5	0.5	20	4.1	84
All IT Projects	27.0	15.0	200	32.6	93
No. of UML Projects	6.2	4.0	51	7.0	168
Other OO Projects	4.0	2.0	50	7.6	127

Table 2. "Typical" project sizes

	Budget (in thousands)	**Person Years**	**Lines Of Code**	**Use Cases**	**Classes**
Mean	5,342	57.5	478,910	88	1311
Median	1,000	6.5	50,000	35	150
Maximum	75,000	3 000	5,000,000	800	25,000
Std Dev	12,000	297	1,050 000	137	4 215
N	71	118	64	75	95

of their years of experience. The figures reported for years of experience with OOAD include both UML and non-UML experience.

The survey also asked in what type of industry the respondent was primarily employed, either as a direct employee or as a consultant. Respondents could select only one industry. Of the respondents using the UML who provided their industry type, 47 percent were in software development, 13 percent in financial services, and 8 percent each in education, aerospace and defense, and health care and pharmaceuticals. About 44 percent also indicated they were associated with their industry through a consulting firm.

The survey asked respondents, "How large are the typical object oriented and/or UML projects you have worked on?" Table 2 shows the results, with budgets in U.S. dollars (with euros and Canadian dollars taken at par). The use cases and classes measures reflect both the size of the project and the extent to which these diagrams were used and exclude responses where they were not used at all. The inclusion of a few very large reported project sizes skewed the means, so the medians are also reported.

Overall UML Diagram Usage

Table 3 shows the relative usage of UML analysis diagrams, with our results compared to others (Grossman et al., 2005; Zeichick, 2002). To keep our survey to a reasonable length, we only asked about use case narratives and UML diagrams covering system structure and behavior that are used to document system functionality. This excluded the object diagram, which is closely related to the class diagram, and the component and deployment diagrams, used in application architecture modeling. Respondents were asked, "What proportion of the object-oriented/UML projects that you have been involved with have used the following UML components?" The five-point usage scale was: None, <1/3, 1/3 – 2/3, > 2/3 and All. The question asked about diagrams used in projects rather than personally by the respondent because the initial interviews found that team members often specialized in one or a few diagrams (e.g., focusing on use case narratives or the class diagram).

Although UML is often presented as being used with a use case-driven approach in the

Table 3. UML diagram usage

UML Diagram	Usage[1]	Never Used (percent)	>2/3 usage (percent)	>1/3 usage (percent)	G[2] (percent)	Z[3] (percent)
class	4.19**	3	73	87	93	75
use case diagram	3.56**	7	51	72	NA	89
sequence	3.51	8	50	75	89	75
use case narrative	3.25	15	44	63	93	NA
activity	2.87**	22	32	55	60	52
statechart	2.82**	19	29	53	63	52
collaboration	2.54**	25	22	42	50	37

[1] *Usage is measured on a scale from 1 (Never Used) to 5 (Used on All Projects)*

[2] *From Grossman et al. (2005)*

[3] *From Zeichick (2002)*

, Significantly different from use case narrative mean, ** p<=0.01, * p<0.05 (t-test)*

UML literature, and in particular by the unified process (Jacobson et al., 1999), only 44 percent of respondents report that use case narratives are used in two-thirds or more of their projects. Over a third of the respondents say their projects never use them, or use them less than a third of the time (15 percent and 22 percent, respectively). Class diagrams were the most frequently used, with 73 percent of respondents using them in two-thirds or more of their projects. Use case narratives were ranked fourth, behind sequence diagrams and use case diagrams. Only three percent of respondents report that their projects never use class diagrams, while collaboration diagrams have the highest non-usage rate of 25 percent. The number of respondents to this question varied from 152 (Statechart) to 172 (class diagram).

Our results are reasonably consistent with other studies (Table 3), except for a much lower use case narrative usage in our study compared to Grossman et al. (2005) and a possibly related lower use of use case diagrams than in Zeichick (2002). In all three studies, collaboration diagrams were found to be the least frequently used. The differences may be attributable to question wording; for example, Grossman et al. (2005) simply asked if the diagram was being used rather than in what percentage of projects. The usage data in all three studies are based on respondents rather than projects. Due to the low correlations (maximum of 0.2) between UML experience and diagram usage, weighting usage by the respondent's number of UML projects increases the averages only slightly.

Most projects made only partial use of the seven UML diagram types studied (Table 4). Of the 135 respondents who reported project usage levels for all seven UML diagrams studied, 51

Table 4. Number of UML diagram types used

UML Diagram Types Used	>1/3 Projects (percent)	>2/3 Projects (percent)
0	6	13
1	4	14
2	8	13
3	10	23
4	21	16
5	16	10
6	19	3
7	16	8

percent reported that five or more of them were used in at least a third of their projects while 21 percent reported five or more used in at least two-thirds of their projects.

Usage rates of the different UML diagram types were all positively correlated with each other, from an r^2 of 0.64 between use case narratives and use case diagrams to 0.16 between use case narratives and Statechart diagrams. Thus, there is apparently no general tendency for projects to use certain diagrams at the expense of others (which would result in a negative correlation). For example, given that sequence diagrams and collaboration diagrams are "semantically close" so that "only minor visual information may be lost" when transforming one to the other (Selonen, Koskimies, & Sakkinen, 2003, p.45), one might expect to find that projects use either the collaboration diagram or the sequence diagram but not both. However, among our respondents, usage of the two was correlated at 0.38 ($p < 0.01$). There were 24 respondents (out of 153) who reported that all their projects use collaboration diagrams and 19 of the 24 reported always using sequence diagrams as well.

In contrast, of the 50 always using sequence diagrams in their projects, 18 used collaboration diagrams less than one-third of the time (and 11 of these never used them). While 87 respondents reported a higher usage level for sequence than for collaboration diagrams, only 12 reported the opposite. Analysts clearly prefer using sequence diagrams but many apparently value depicting the same information in different ways for different purposes. Their isomorphic nature also means that sequence and collaboration diagrams share underlying data, so the incremental cost of producing both (after committing to either one) is low with some UML tools.

UML Diagram Usage Patterns

The survey collected demographic data about respondents, their organizations, use of tools, and types of systems being built. Not all respondents completed these sections so the sample sizes for this analysis are somewhat smaller. Differences that are not reported were statistically insignificant.

Organization Size

There are a number of significant positive relationships between organization size measures and the use of UML diagrams. Comparing organizations above to those at or below $10 million in annual revenue, the former are significantly more likely to use use case narratives ($p=0.001$), use case diagrams ($p=0.02$) and sequence diagrams ($p=0.02$) and they use an average 4.75 diagram types compared to 3.65 for smaller organizations ($p=0.01$). Comparing usage by organizations with 50 or more IT employees to those with fewer, the former are more likely to use sequence diagrams ($p=0.01$) and activity diagrams ($p=0.03$). However, those with more employees use only slightly more use case narratives and total number of diagram types. Both points used to divide the samples were chosen to create roughly equal subsamples so they are somewhat arbitrary. Moreover, the two size measures are not independent ($r=0.72$).

Project Size

Larger projects might be expected to make wider use of UML diagram types, but this is generally not the case. A similar analysis using the five project size measures (Table 2) found that respondents reporting larger than average budgets reported more use of use case narratives and more diagram types used over a third of the time ($p<0.05$). Larger projects based on person-years also reported greater use of use case narratives ($p<0.05$). However, no other comparisons were significant.

UML Tools

The availability of UML tools is also related to the use of UML diagram types. Those with tools are significantly more likely to use class diagrams ($p=0.02$) and sequence diagrams ($p<0.001$) with usage levels higher for all remaining diagram types as well (none significant). Respondents from larger organizations might be expected to have better access to tools and they do, but only slightly so this does not explain why larger organizations are using more diagram types. Correlations between the organization size measures (annual revenue and number of employees) and spending on tools are also low (0.25 and 0.29, respectively, neither significant) even though tool cost is typically partly dependent on the number of installations.

Organizational UML Usage

Overall usage of the UML in an organization could affect practices within individual projects. For example, analysts (and presumably organizations) could begin learning the UML by focusing on a subset of diagrams (Ambler, 2002, pp.46-47). The survey data do not permit any direct testing to determine whether individual analysts are taking this approach. However, respondents from organizations using the UML in 40 percent or fewer of their projects use an average of 2.4 diagram types two-thirds of the time or more. Those from organizations using the UML in over 40 percent of projects average a significantly greater ($p = 0.02$) 3.3 diagram types and are also making significantly ($p = 0.03$) more use of sequence diagrams (3.84 usage level compared to the 3.51 average in Table 3 and 3.23 level for those using the UML 40 percent of the time or less). Usage levels of all the remaining diagram types are very similar to those reported in Table 3 for both groups.

Respondent Experience

There are generally weak relationships between respondent experience and their projects' UML diagram usage. Experience measures (Table 1) were correlated with use of each UML diagram type (Table 3). The strongest relationships involved Statechart diagrams and years of experience in object-oriented analysis and design (0.45, $p<0.01$) and years of experience with UML (0.35, $p<0.01$). Class diagram usage also correlated significantly ($p<0.01$) with these two experience measures at 0.36 and 0.40 respectively, and with years of object-oriented programming (0.31). No other correlations between experience measures and diagram type usage exceeded 0.30 (and thus they explained less than 10 percent of the observed variance).

Industry

The survey provided 15 possible industrial classifications, with all but one receiving 12 or fewer responses (insufficient to be useful in analysis). The 46 respondents working in the software development industry, who do not always have identifiable clients in the same sense as those working in other organizations, had somewhat (but not significantly) lower use of use case narratives, sequence diagrams and activity diagrams.

In our initial informal interviews, consultants were always described (by themselves and by others) as enthusiastic proponents of use case

narratives and the use case-driven philosophy. While we expected similar results from the survey, instead consultants reported lower use case narrative usage than non-consultants (but not significantly, p=0.07). However, consultants were significantly more likely (p=0.04) to use collaboration diagrams.

System Type

Respondents were asked to indicate the application area(s) in which their systems were being built. The seven choices (with the number of responses in parentheses) were e-commerce (90), administrative (71), embedded (36), manufacturing (28), customer relationship management (26), data mining (21) and mobile commerce (17). There were also 58 who provided "other" categories, although many used this option to further describe one of the existing categories. Building software tools (6) was the most common selection not listed in the survey.

Use case narratives were used most by those developing customer relationship management (3.57) and e-commerce systems (3.48), and least by those developing embedded systems (2.64). (The numbers shown use the same five-point scale as in Table 3.) T-test significance levels were 0.01 and 0.001, respectively, after excluding respondents who selected both the system types being compared. However, embedded system projects had the highest reported usage of sequence diagrams (3.56), while customer relationship management had the least (2.14) ($p<0.005$). Activity diagrams were used most in development of manufacturing systems (3.12) and least in embedded (2.58), but this difference was not significant.

Respondents were also asked to identify the proportion of their object oriented/UML projects that would be consider new systems, complete replacements of existing systems, or enhancements to existing systems. Most entered percentages that totaled 100, but others entered the number of each type and these were converted to percentages. There were 154 usable responses, averaging 56 percent new, 20 percent replacement and 24 percent enhancements. Table 5 computed the usage of each UML diagram (computed as in Table 3) for those who reported at least 50 percent of their projects were of that type; there were 96 responses for new systems, 23 for replacement projects and 34 for enhancement projects. (Some responses were split 50/50 between two types and were counted twice while others were split

Table 5. UML diagram usage by project type

UML Diagram	New System	Replacement System	Enhancement of System
class	4.19	4.82	4.38
use case diag	3.62	3.90	3.56
sequence	3.55	3.95	3.43
use case narr	3.08	3.82	3.23
activity	2.99	2.95	2.69
statechart	2.87	2.75	2.63
collaboration	2.59	2.95	2.45

more evenly among all three types and were not counted at all.) Most notable is the greater use of class diagrams and use case narratives when developing replacement systems.

Information Provided by UML Diagrams

There are a number of reasons for using multiple diagram types to describe system functionality, beginning with the possibility that different diagrams convey different information. To investigate this, the survey asked which diagrams provide new information beyond that contained in use case narratives. The use case narratives were chosen as the benchmark because a use case-driven approach had been endorsed by much of the early UML literature. Both the interviews and a literature review (Dobing & Parsons, 2000) showed that use case narratives varied widely in level of detail, so simply knowing that use case narratives are being employed does not answer the question of how much information they contain. In contrast, the level of redundancy across other pairs of diagrams is largely determinable from their syntax. The question used a five-point scale from "No New Info" to "All New Info," with "Some New Info" as the midpoint (3). This item was only seen by those whose projects had used both use case narratives and the other diagram in question so there were fewer respondents, from 89 (collaboration diagram) to 125 (class diagram).

Table 6 shows that the diagram of highest value for conveying new information not already contained in the use case narratives was the class diagram, with a score of 3.51 on the five-point scale, and 86 percent of respondents believe it offers at least some new information (at least 3 on the 5-point scale). The use case diagram was least useful in providing additional information, which is not surprising given its role is to depict the use cases and their relationships to actors and to each other.

Stronger relationships were expected between the belief that a UML diagram provides additional information beyond the use case narrative and the usage level of that diagram. For activity diagrams, the correlation was 0.42 ($p<0.01$). However, other correlations of this type were all weak (i.e., none exceeded 0.30, so none explained more than 10 percent of the variance).

There was also a strong correlation (0.77) between the beliefs that collaboration and sequence

Table 6. New information (not in use case narratives) from UML diagrams

UML Diagram	New Information[1]	Some – All New Information (percent)
class	3.51	86
use case	2.42**	48
sequence	3.37	78
activity	2.89**	63
statechart	3.38*	79
collaboration	2.98**	67

[1] *New information is measured on a scale from 1 (No New Information) to 5 (All New Information)*

*,** *Significantly different from class diagram mean,* ** *p<=0.01,* * *p<0.05 (t-test)*

diagrams provide new information beyond use case narratives, the highest correlation found among all pairs of diagrams. This could be attributed to the isomorphic relationship between collaboration and sequence diagrams (i.e., that they convey similar information but in different ways).

Role of UML Diagrams

Table 7 examines reasons for including each UML diagram in a project, with the focus on communication within the project team. Each respondent who reported using a particular diagram at least a third of the time was asked about four possible purposes. As expected, use case narratives had the highest score for "Verifying and validating requirements with client representatives on the project team" at 4.00 (on a 5-point scale). The use of other diagrams for this purpose was higher than expected, based on interview responses and our review of the UML literature. These high levels of client involvement show that use of the more technical diagrams of the UML is not limited to the technical members of the development team. The survey also included a single item that asked, "How successful has the UML been in facilitating communication with clients?" The items used a five-point scale from Not to Very Successful. The mean was 3.28 with 25 percent choosing the lowest two levels.

Of those respondents who reported using a particular diagram at least a third of the time, Table 8 shows the percentage who rated them from "Moderately Useful" to "Essential" for four different purposes. The results show higher than expected levels of usefulness for all UML dia-

Table 7. Roles for UML diagrams

UML Diagram	**Client Validation[1]**	**Implement[2]**	**Document[3]**	**Clarify[4]**
use case narrative	4.00	3.62†	3.15††	3.52††
activity	3.50**	3.43††	3.35††	3.50††
use case diagram	3.36**	3.06††	2.90††	3.17††
sequence	2.91**	3.71†	3.76††	4.14†
class	2.90**	4.06	4.18	4.35
statechart	2.63**	3.51††	3.35††	3.74††
collaboration	2.62**	3.25††	2.96††	3.40††

[1] Verifying and validating requirements with client representatives on the project team

[2] Specifying system requirements for programmers

[3] Documenting for future maintenance and other enhancements

[4] Clarifying understanding of application among technical members of the project team

*** Significantly different from use case narrative mean,*

*** p<=0.01 (t-tests)*

†,†† Significantly different from class diagram mean,†† p<=0.01, † p<0.05 (t-tests)

Table 8. Percent of respondents who believe each UML diagram is at least moderately useful

UML Diagram	**Client Validation[1]**	**Implement[2]**	**Document[3]**	**Clarify[4]**
use case narrative	87	79	68	74
activity	77	81	73	80
use case diagram	74	62	61	66
sequence	62	84	85	92
class	57	89	92	93
statechart	49	79	71	82
collaboration	51	70	62	74

[1] Verifying and validating requirements with client representatives on the project team

[2] Specifying system requirements for programmers

[3] Documenting for future maintenance and other enhancements

[4] Clarifying understanding of application among technical members of the project team

grams in "Verifying and validating requirements with users." Only Statecharts, at 49 percent, were under the 50 percent level.

The other three purposes listed are more related to communication within the project team, among analysts, programmers and maintenance staff. For these three purposes, the class diagram was considered most useful with the use case diagram least useful (but all diagram types were rated as at least "moderately useful" by over 60 percent of respondents). As noted earlier, the use case diagram provides an overview of the project while programming tends to focus on implementing particular functionality. In Table 8, the usefulness levels reported for sequence diagrams are all significantly higher ($p<0.01$) on the three project team communication measures than those for the isomorphic collaboration diagram.

These reported levels of client involvement with the full range of UML diagrams exceed those generally recommended in the literature and, in particular, seem inconsistent with the dominant use case-driven philosophy. Concerns have been raised about a potential disconnect that could result from relying on use case narratives when working with clients and class diagrams when working with technical team members (Dobing & Parsons, 2000). The survey results confirm that use case narratives are indeed the primary diagram for communication with clients and class diagrams for communication within the technical members of the team. However, all diagrams received at least "moderately useful" ratings from over 50 percent of respondents across all forms of communication (Table 8), except using Statechart diagrams for communication with clients which was 49 percent. In particular, use case narratives are widely used among the technical members of project teams. This suggests that the disconnect problem may well have been addressed in practice, if not in the UML literature.

Table 9. Reasons for not using some UML Diagrams (% responses)

UML Diagram	Not well understood by analysts	Not useful for most projects	Insufficient value to justify cost	Information captured redundant	Not useful with clients	Not useful with programmers
class	50	13	13	25	25	25
sequence	32	23	36	14	23	23
use case narrative	29	26	37	29	11	26
use case diagram	32	32	42	19	29	42
statechart	35	42	28	12	28	33
activity	48	23	35	35	14	25
collab'tion	27	32	24	49	29	24

Those who reported that their projects used a particular diagram less than a third of the time (including not at all) were asked why they were not using it more often. There were fewer respondents for these questions, ranging from only 8 for class diagrams to 59 for collaboration diagrams. Table 9 shows the percentage of respondents who selected each possible reason. Respondents were encouraged to select all reasons that applied so row totals exceed 100 percent. A lack of understanding by analysts was the primary factor among the few not using class diagrams (50 percent). Similar concerns were expressed by 48 percent of respondents about activity diagrams. Leading concerns for the remaining diagrams were over how useful they are (Statechart), their value (sequence and use case diagrams and narratives) and the degree of redundancy (collaboration, presumably with respect to sequence diagrams).

Client Participation

Client participation has long been considered crucial to successful system development. The survey asked about the client's role in relation to each of the UML diagram types being studied. Respondents were able to select more than one (e.g., they could report that clients helped to develop use case narratives, reviewed some or all of them upon completion and had formal approval authority). The results are summarized in Table 10. For example, 76 percent of respondents who used use case narratives reported that clients were involved in their development. When UML diagram types are ranked on the level of client participation, the order is very similar (with only class and collaboration diagrams transposed) to "comprehensibility" rankings for managers, users and domain experts (Arlow & Neustadt, 2004, p.91).

The results show that clients were most likely to be involved in developing, reviewing and approving use case narratives and the use case diagram. Of the remaining diagrams, activity diagrams are probably the easiest for clients to understand and almost half the analysts report some involvement by clients in their development, consistent with the comment above by Schneider and Winters (2001).

Table 10. Client participation

UML Diagram	Develop (%)	Review (%)	Approve (%)	N
use case narr	76	63	54	78
use case diag	57	69	46	77
activity	47	60	19	57
sequence	37	52	16	87
class	33	53	20	103
collaboration	38	48	13	48
statechart	28	36	20	61

While clients were less likely to be involved in developing the class diagram, just over half were involved in reviewing this widely used diagram. The wide range of client involvement practices in our interviews and survey results is not unexpected given that most organizations have relatively limited experience with the UML.

Not surprisingly, clients were least likely to be involved in developing or reviewing Statechart diagrams. The fact that about one quarter to a third were involved in these tasks may reflect the technical sophistication of some clients in the survey sample, since the composition of OMG membership includes many large companies in the computer industry.

Respondents were also asked about possible difficulties that had occurred which "could be attributed to the UML." They could check any or all of the five categories listed. User interface concerns were checked most frequently (36 percent), followed by roles and responsibilities of particular users (21 percent), security (18 percent), data requirements (18 percent), and system capabilities and functionality (13 percent).

Respondents Not Using the UML

Some limited analysis was also done based on the 102 responses from those not using the UML. Software development was also the largest organization type for this group (31 percent) with education second (25 percent). These respondents were less experienced than the UML practitioners (averaging 8.1 years IT experience and 16.3 IT projects vs. 15.1 years and 27.0 projects for UML practitioners). The sample selection method suggests this group is probably more knowledgeable about the UML (and more interested in it) than the average non-practitioner. The primary reasons given by those not using the UML or any object-oriented approach were a lack of people familiar with the UML (51 percent) and a lack of suitable projects (16 percent). Of those whose organizations were using an object-oriented approach but not the UML, 55 percent cited a lack of people familiar with the UML while 23 percent said they had no suitable projects, 17 percent said it was too complex, 17 percent said it was not yet standardized or accepted and 15 percent indicated

their tools were not compatible with the UML. Respondents could select more than one answer so the percentage total exceeds 100 percent.

DISCUSSION AND RECOMMENDATIONS

This is the first survey we are aware of investigating not only how but why UML diagrams are used or not used in systems analysis. We found variations on all three of the major dimensions studied, including frequency of use for each diagram type, the purposes for which they were being used, and the role of clients/users in their creation and approval. While the UML is "unified" in that it brought together elements from disparate modeling notations, considerable variations remain in its use that are somewhat inconsistent with the notion of the UML as a "unified" language in the sense of implying coordinated and cohesive use of diagram types within a development project. Moreover, we found that use of only a subset of UML diagrams on a project is widespread. The data also show a variety of reasons why certain UML diagrams are not used.

The findings of this research can be useful in a number of ways. First, information on UML use can provide valuable input in the evolution of the standard. For example, on the issue of complexity, the language could be simplified by eliminating collaboration diagrams. Based on our findings, collaboration diagrams are used less often, deemed to be less useful, and appear to offer little additional value in relation to sequence diagrams. Statechart diagrams are also used less often than most and seem to be less useful most of the time, but are rated highly for providing new information in some situations (e.g., real-time systems) and have low redundancy. Admittedly, both these diagrams also have some strong supporters. As one interview subject said about Statecharts, "When they are useful, they are very useful."

Second, some projects do not follow a use case-driven approach with over a third of the respondents saying they use them less than a third of the time. At the same time, there is limited empirical evidence to support the proposition that use case narratives are a more effective way to communicate with clients than are the other UML diagrams. Research is also needed to determine an appropriate level of granularity and level of detail for use case narratives.

Third, more attention may be needed on the issue of how clients or users can be better prepared to participate in development and review of artifacts beyond use case narratives. We found that the use of diagrams other than use case narratives among clients or users was higher than expected based on the extant prescriptive literature on 'how to use' the language. The UML practitioner literature generally seems to assume that UML diagrams, except for use case diagrams and narratives, are too complex or technical to be understood by clients. However, our results show that clients frequently approve, review, and even help develop all of the UML diagrams. The views of clients and intended users of systems on the usability and usefulness of UML have received little attention from researchers (including this study). Nor has much consideration been given to how to prepare clients for this level of involvement.

Fourth, research is needed to understand which UML diagrams can best facilitate communication between clients and analysts, particularly as the use of the UML to support agile modeling grows (Ambler, 2002). In addition, work might be needed to modify these diagrams (e.g., by simplifying or otherwise changing the syntax and grammar of the diagram type) to enable them to support communication and verification more effectively.

Fifth, as noted earlier, 36 percent of respondents agreed that they had experienced "difficulties … [with user interfaces] that could be attributed to the UML." User interfaces have become much more complex over the past decade

with the use of both visual programming and Web environments, complicating development using any methodology or notation. Based on accompanying comments, respondents would welcome better ways to integrate user interface design with UML modeling. One approach is to distinguish between "system" and "essential" use case narratives (Constantine & Lockwood, 1999), where essential use cases are independent of technology (and user interfaces) while system use cases include these details. Currently, the UML has no standards for use case narratives (OMG, 2005) or system use cases in particular, which might explain why many respondents experienced user interface issues that they could attribute to the UML. Another approach is to use prototyping or other screen design tools in conjunction with the UML. One respondent noted that, "It is easier for clients to understand the functionality of software through user interface sketches" while another said that clients had difficulty validating use case narratives "without any draft of the [user interface]." The principle that system analysis should be technology independent long precedes the development of the UML and is widely accepted among leading writers in the field but, as several respondents pointed out, some difficulties can arise when applying this principle in practice. Another respondent noted difficulties in creating a vocal interface, pointing out that not all interfaces are purely visual. There are some very interesting research opportunities in this area.

Respondents provided fewer comments on other difficulties with the UML. Concerns about security are to be expected but no suggested solutions were mentioned. One difficulty with database design is that many respondents were using a relational, rather than object-oriented, DBMS. Difficulties identifying the "roles and responsibilities of particular users" suggest that there may be problems mapping the UML "actor" to specific individuals or job descriptions. There are some strong parallels with this issue and user interfaces; at least some clients prefer to work with more concrete designs that clearly show who does what rather than with more abstract approaches that take a higher level view. More research is needed on how clients can more effectively validate designs.

Finally, there have been numerous attempts to evaluate UML from a theoretical standpoint including assigning ontological semantics to UML constructs (Evermann & Wand, 2001a; Opdahl & Henderson-Sellers, 2001) and assessing the complexity of the UML (Siau & Cao, 2001, 2002; Siau et al., 2005). In cases such as these, theoretical conclusions can be substantiated or refuted by empirical data on usage. To illustrate, some UML constructs appear to have no ontological counterpart and such constructs may not be suitable for conceptual modeling (Evermann & Wand, 2001a). We then might expect that diagrams that have more such constructs would be less useful and less used in conceptual modeling. In terms of this study, this would correspond to less use of such diagrams/constructs for verifying and validating requirements with users. Our study did not examine use at the level of constructs within diagrams, but future empirical studies might do an analysis at that level of detail (perhaps for a single diagram type).

SURVEY RESPONDENT CHARACTERISTICS: PROFILE AND LIMITATIONS

Given the lack of any defined population of UML practitioners from which to obtain a random sample, we chose to survey primarily OMG members and those who use its Web site. This may have produced biased responses. However, given that the goals of this research were to examine how UML practitioners (the target population) were using the language, rather than the extent to which it is being used in software development in general, the participation of the OMG seemed appropriate. While respondents may not

be representative of all UML practitioners, they can be considered leading edge adopters. UML experience was naturally low when this survey was conducted, with medians of five years and four projects. As such, respondents might not be typical of the eventual set of UML practitioners (which could become the majority of system developers if object-oriented system development and the UML become more widely accepted). Based on other research in technology adoption, albeit in different areas (e.g., Brown & Venkatesh, 2003), early adopters might approach the UML quite differently from those who come later.

In addition, there are some obvious limitations with using a convenience sample. The number of people who received or read the invitation to participate is unknown because of the possibility of it being forwarded. Visitors to the OMG site need not be members, so the results should not be considered as a survey of OMG's membership even prior to inviting readers of the comp.object Usenet newsgroup. It is also likely that some people found the survey through search engines, since the survey was, for some time, the top result of a Google search on "UML survey." Despite the lack of control over respondents, reviewing the comments and contact information suggests that the group as a whole does belong to the target population and are reasonably diverse on a range of demographic measures. Moreover, whether they worked for OMG member companies or found the survey by other means, the respondents clearly were very interested in the UML, and can reasonably be considered leading edge practitioners. Whether respondents are representative of the target population of all analysts who use the UML is unknown.

A majority of respondents opted to remain anonymous so they could have submitted two or more responses, but there was no reason for them to do so and there are no obvious patterns of duplicate responses. It is also conceivable that results could be skewed by heavy participation by a single organization. However, responses came from a wide variety of organization types and sizes and there were no bursts of unusual activity levels. Among those who did provide an email address, no company domain had more than one response except for email providers (15 from Yahoo! accounts, five from Hotmail, etc.). So there is no evidence to suggest the results were manipulated by any individual or group.

The survey took a use case-driven approach, consistent with a majority of the books written on the UML up to the time of this survey. However, only 44 percent of respondents reported using use case narratives in at least two-thirds of their UML/object-oriented projects. Measuring the value of the information provided by different UML diagrams by comparing them to use case narratives therefore seems insufficient and new measures are needed.

The measures used for project size were also problematic. Low numbers of classes and use case narratives used in some "typical" projects probably reflect limited usage of those diagram types rather than the real size of the project, and many projects are not using them at all. Some unrealistically low budgets (which were coded as missing data) were perhaps intended to be in thousands of dollars while others appear to exclude salaries. On the other hand, some larger budgets might have included training and tool acquisition costs that do not reflect project size. Lines of code is commonly used as a measure of project size, and was used here because of its simplicity. However, respondents may or may not have included shared code, comments, etc. and programming style can also affect code size. Low correlations among these size measures also suggest a lack of reliability. Budget and number of classes correlated at 0.65 while person-years and lines of code correlated at 0.44 (both with $p<0.01$). But the next largest correlation is only 0.25.

Measures of user/client involvement have a long history in the IS literature. This survey measures only the perspective of IT professionals rather than the clients themselves. In the earlier

interviews, there were several cases of strong disagreement between the clients and analysts on their roles and even on appropriate use of some UML diagram types. There can also be differences in what 'client' means across different organizations and situations. Are clients those sponsoring the system or does this term also include the intended direct users? Some external consultants might view the IT Department that hires them as the client, while those developing commercial software may have certain types of clients in mind but have limited interaction with them.

CONCLUSION

The UML has rapidly become the de facto standard for object-oriented systems development. However, this survey suggests there is no standard approach to using the UML within a group of arguably leading edge practitioners. There is considerable variation in use of diagrams across projects and in the role clients/users play in the development of UML models. Clearly, in view of the popular interest in the UML, further research is needed to better understand UML use in order to gain insight on how it can be effectively used to support systems development. The results of this survey suggest several aspects of UML adoption and use that need to be studied.

ACKNOWLEDGMENT

The authors would like to thank the Object Management Group, and Richard Soley in particular, for their support of our research. Thanks also to Dinesh Batra and other anonymous reviewers for their helpful suggestions and comments. Funding for this research was provided by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Ambler, S. (2002). *Agile modeling: Effective practices for extreme programming and unified process.* New York: John Wiley.

Arlow, J., & Neustadt I. (2004). *Enterprise patterns and MDA: Building better software with archetype patterns and UML*. Boston: Addison-Wesley.

Booch, G. (1994). *Object-oriented analysis and design with applications* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Booch, G. (1999). UML in Action. *Communications of the ACM, 42*(10), 26-28.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*. Reading, MA: Addison Wesley.

Brown, S., & Venkatesh, V. (2003). Bringing non-adopters along: The challenge facing the PC industry. *Communications of the ACM, 46*(4), 76-80.

Burton-Jones, A., & Meso, P. (2006). Conceptualizing systems for understanding: An empirical test of decomposition principles in object-oriented analysis. *Information Systems Research, 17*(1), 101-114.

Burton-Jones, A., & Weber, R. (2003). Properties do not have properties: Investigating a questionable conceptual modeling practice. *Proceedings of the 2nd Annual Symposium on Research in Systems Analysis and Design,* St. John's, Canada.

Constantine, L.L., & Lockwood, L.A.D. (1999). *Software for use*. Reading, MA: Addison-Wesley.

Cook, S. (2000). The UML family: Profiles, prefaces, and packages. In *Proceedings of UML 2000 - The Unified Modeling Language. Advancing the*

Standard, Lecture Notes in Computer Science, Vol. 1939, (pp. 255-264). Springer.

DeJong, J. (2006). Of different minds about modeling. *SD Times*, June 15. Retrieved from http://www.sdtimes.com/article/special-20060615-02.html.

Dobing, B., & Parsons, J. (2000). Understanding the role of use cases in UML: A review and research agenda. *Journal of Database Management, 11*(4), 28-36.

Dori, D. (2002). Why significant UML change is unlikely. *Communications of the ACM, 45*(11), 82-85.

Duddy, K. (2002). UML2 must enable a family of languages. *Communications of the ACM, 45*(11), 73-75.

Evermann, J., & Wand, Y. (2001a). Towards ontologically based semantics for UML constructs. *Proceedings of the 20th International Conference on Conceptual Modeling,* (pp. 354-367). Yokohama, Japan.

Evermann, J., & Wand, Y. (2001b). An Ontological Examination of Object Interaction in Conceptual Modeling. *Proceedings of the 11th Workshop on Information Technologies and Systems,* (91-96). New Orleans, Louisiana,

Evermann, J., & Wand, Y. (2006). Ontological modeling rules For UML: An empirical assessment. *Journal of Computer Information Systems, 46*(5) 14-29.

Grossman, M., Aronson, J., & McCarthy, R. (2005). Does UML make the grade? Insights from the software development community. *Information and Software Technology, 47*(6), 383-397.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming,* 8(3), 231-274.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Reading, MA: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). O*bject-oriented software engineering: A use case driven approach*. Reading, MA: Addison-Wesley.

Jacobson, I., Ericsson, M., & Jacobson, A. (1994). *The object advantage: Business process reengineering with object technology.* Reading, MA: Addison-Wesley.

Johnson, R., & Hardgrave, B. (1999). Object-oriented methods: current practices and attitudes. *Journal of Systems and Software, 48*(1), 5-12.

Kobryn, C. (1999). UML 2001: A standardization odyssey. *Communications of the ACM, 42*(10), 29-37.

Kobryn, C. (2002). Will UML 2.0 be agile or awkward? *Communications of the ACM, 45*(1), 107-110.

Larman, C. (2005). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Moore, A. (2001). Extending UML to enable the definition and design of real-time embedded systems. *Crosstalk: The Journal of Defense Software Engineering, 14*(6), 4-9.

Odell, J., Van Dyke, P., & Bauer, B. (2000). Extending UML for agents. *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence,* (3-17). Austin, Texas,

Object Management Group. (2005). *Unified modeling language: Superstructure,* Version 2.0.Retrieved from http://www.omg.org/technology/documents/formal/uml.htm.

Opdahl, A.L., & Henderson-Sellers, B. (2001). Grounding the OML metamodel in ontology. *Journal of Systems and Software, 57,* 119-143.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design.* Englewood Cliffs, NJ: Prentice Hall.

Schneider, G., & Winters, J. (2001). *Applying use cases: A practical guide* (2nd ed.). Boston: Addison-Wesley.

Selonen, P., Koskimies, K., & Sakkinen, M. (2003). Transformations between UML diagrams. *Journal of Database Management, 14*(3), 37-55

Siau, K., & Cao, Q. (2001). Unified modeling language (UML)—a complexity analysis. *Journal of Database Management, 12*(1), 26-34.

Siau, K., & Cao, Q. (2002). How complex is the unified modeling language? *Advanced Topics in Database Research, 1,* 294-306.

Siau, K. Erickson, J., & Lee, L.Y. (2005). Theoretical vs. practical complexity: The case of UML. *Journal of Database Management, 16*(3), 40-57.

Stevens, P., & Pooley, R. (2000). *Using UML: Software engineering with object and components.* Reading, MA: Addison-Wesley.

Zeichick, A. (2002). Modeling Usage Low; Developers Confused About UML 2.0, MDA, *SD Times,* July 15. Retrieved from http://www.sdtimes.com/article/story-20020715-03.html.

This work was previously published in the Journal of Database Management, edited by K. Siau, Volume 19, Issue 1, pp. 1-18, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 4.24
Enterprise Resource Systems Software Implementation

Ganesh Vaidyanathan
Indiana University, USA

ABSTRACT

Enterprise resource planning systems are complex yet single, integrated software programs that runs off a single database so that the various departments can easily share information and communicate with each other. The integrated approach can have a tremendous payback if companies implement the software correctly. This chapter illustrates the implementation steps as followed by major corporations in the United States, and provide an insight into the practical implementation issues. A business case for such systems is introduced in this chapter as well. The chapter provides seven ERP issues and elaborates these issues in the context of implementation. The implementation details during conceptualization, design, implementation, go-live, and operation stages are provided with a note to practitioners on ERP implementation.

INTRODUCTION

Enterprise resource planning (ERP) software attempts to integrate all departments and functions across a company onto a single computer system that can serve all those different departments' particular needs (Koch 2002). Each of those departments typically has its own computer system optimized for the particular way that the department does its work. But ERP combines them all together into a single, integrated software program that runs off a single database so that the various departments can more easily share information and communicate with each other. That integrated approach can have a tremendous payback if companies install the software correctly.

Typically, when a customer places an order, that order begins a mostly paper-based journey from in-basket to in-basket around the company, often being keyed and re-keyed into different departments' computer systems along the way. These activities cause delays and errors. Meanwhile, no

one in the company truly knows what the status of the order is at any given point because there is no way for the finance department, for example, to get into the warehouse's computer system to see whether the item has been shipped. ERP can replace the old standalone computer systems in accounting, human resources, manufacturing, and warehouse with single unified software. This results in integrated software that is linked together so that someone in finance can look into the warehouse module to see if an order has been shipped. Most vendors' ERP software is flexible enough to install certain modules without buying the whole package.

Enterprise systems that encompass all departmental processes can often be complex and interdependent. Highly interdependent technology solutions such as ERP are used by firms to enhance the efficiency and ease of in-house capabilities. The use of ERP is characterized by high levels of task interdependence (Sharma and Yetton 2003). To implement such highly complex and interdependent systems is often a daunting process. Implementing an enterprise-wide application like an ERP system to help run a business is a costly and complex process and is like implementing a civil engineering endeavor or sizable construction project (Hawksworth 2007). A certain amount of planning, discipline and wisdom are required to complete implementation on schedule to meet the requirements of a firm.

Many of the current ERP literature share implementation experiences from various companies. While some of them attempt to explain why the ERP implementation is difficult and what needs to be done to achieve desirable results, others present various models of implementation stages and different implementation methodologies (Moon, 2007). The contributions of this chapter to researchers and practitioners include:

a. Illustration of the implementation steps as followed by major corporations in the United States, and
b. Provision of an insight into the practical implementation issues, and
c. Introduction to a business case for ERP systems.

This chapter details the implementation issues of ERP systems and provides an insight into the practical aspects of such implementation. The next section provides seven ERP issues and elaborates these issues in the context of implementation. The following section describes ERP software and the ERP implementation scheme during conceptualization, design, implementation, go-live, and operation stages of implementation. The chapter concludes with a note to practitioners on ERP implementation.

WHAT CAN ERP DO?

ERP is an enterprise software package. With ERP, it is possible to keep track any transaction in an enterprise in real-time. ERP allows managers to process business information more effectively to support sound decision making. ERP solutions cover all of the core operations necessary to run successful small and midsize businesses, including accounting and banking, customer and vendor management, purchasing and sales, logistics and production, as well as reporting and analysis.

The benefits of ERP systems have been researched extensively in literature. Gefen and Ragowsky (2007) examined associations between the business characteristics of manufacturing firms and their perceived benefits from ERP system investments at both enterprise and a specific IT module level and found that the perceived value for ERP investments was consistently better explained at the specific IT module level. Ranganathan and Brown (2006) found that ERP projects with greater functional scope (two or more value-chain modules) or greater physical scope (multiple sites) result in positive, higher shareholder returns. ERP systems replace com-

plex and sometimes manual interfaces between different systems with standardized, cross-functional transaction automation (Hendricks et al., 2005). Information integration using ERP can replace functionally oriented and often poorly connected legacy software, resulting in savings in infrastructure support costs (Hendricks et al., 2005). Other business benefits include:

- **Improved productivity:** ERP engages and connects users within and beyond the enterprise, including customers, suppliers, and partners. An intuitive, role-based portal environment gives the system wide access to a single, consistent view of the business. Higher levels of efficiency and collaboration may be achieved and as a result firms can respond to new competitive threats, and proactively meet customer needs.
- **Increased insight:** ERP improves decision-making by giving managers a clear understanding of activities across functions. They can retrieve the right information at the right time to address problems and pursue new opportunities.
- **Enhanced governance:** ERP provides comprehensive functionality for corporate governance, enabling the firm to comply with Sarbanes-Oxley and International Accounting Standards. ERP also integrates corporate reporting, analysis, and compliance with underlying business processes and transaction systems.
- **Improved flexibility:** ERP provides a scalable and adaptable solution that seamlessly integrates end-to-end processes with the ability to add other external solutions that may include customer relationship management, supply chain management, and product life-cycle management.
- **Reduced costs:** ERP enables firms to manage IT costs by leveraging the investments they have already made.
- **Increased visibility:** ERP solutions provide real-time visibility across the entire enterprise, so firms can streamline their supply chain, bring products to market faster, get more out of procurement, and eliminate duplication of effort.

ERP IMPLEMENTATION ISSUES

Companies that install ERP do not have an easy time of it. To implement ERP right, the ways that they do business need to change and the ways people do their jobs need to change too. And that kind of change does not come without pain. The important thing is not to focus on how long it will take but rather to understand why the firms need ERP and how they will use it to improve their business.

Cost

Meta Group recently did a study looking at the total cost of ownership (TCO) of ERP, including hardware, software, professional services and internal staff costs. The TCO numbers include getting the software installed and the two years afterward, which is when the real costs of maintaining, upgrading and optimizing the system for business are felt. Among the 63 companies surveyed (Ketbi et al. 2002)—including small, medium and large companies in a range of industries—the average TCO was $15 million (the highest was $300 million and lowest was $400,000). The nature of ERP implementations are such that there are usually unforeseen and unexpected occurrences that increase the overall costs (Al-Mudimigh et al., 2001). In summary, the cost of implementing ERP remains quite high (Wu et al., 2007).

Schedule

The criticality of schedule and budget overruns as risk factors in ERP implementation projects has

been rated to be high by all companies regardless of their size (Laukkanen, 2007). The time and effort to implement is likely to be underestimated in the case of implementing ERP systems. ERP systems come in modules and do not have to be implemented entirely at one time; many companies follow a phase-in approach in which one module is implemented at a time. Some of the most commonly installed modules are sales and distribution (SD), materials management (MM), production and planning, (PP), and finance and controlling (FI) modules. The average length of time for a "typical" implementation is more than a year and can use more than 150 consultants. Corning, Inc. rolled out ERP in ten of its diversified manufacturing divisions in five to eight years (Stedman 1998). The length of implementation is affected to a great extent by the number of modules being implemented, the scope of the implementation (different functional units or across multiple units spread out globally), the extent of customization, and the number of interfaces with other applications.

Customization

ERP packages are very general in nature and need to be configured to a specific type of business. The customization is very tedious and takes a long time, depending on the specific requirements of the business. Customized development can provide a better fit with the operational procedures of a firm, yet often results in a system that is more risky to implement and more complicated to maintain and upgrade (Gebauer and Lee, 2008). An ERP system like SAP is so complex and general that there are more than 8000 switches that need to be set properly to make it handle the business processes in a way a company needs. The more customization needed, the longer it will take to roll out the software and the more it will cost to keep it up-to-date. The length of implementation time could be cut down by keeping the system "plain vanilla" and reducing the number of "bolt-on" application packages that require custom interfaces with the ERP system. However, the downside to this "plain vanilla" approach may or may not completely match business requirements.

Some companies undertake costly customizations to automate its processes on its ERP system—only to learn that the "plain-vanilla" version of the software performed certain functions much better. Companies even pull the plug partway into an ERP project because of functional or even philosophical problems. A technological mistake often made in SAP implementations, says Graham McFarlane, director of Western Management Consultants, is that organizations modify the software more than they should, rather than modifying their business processes (Hilson, 2001). The Military Sealift Command (MSC) successfully implemented Oracle's ERP solution. One rule they cite for achieving ERP success is that agencies must find an ERP package that mirrors their business practices as closely as possible, then resolve to implement the package without significant modifications. MSC managers made a key decision to minimize the risk of ERP implementation by taking a "vanilla" approach. They committed to installing the software as it was packaged, without any modifications. The MSC put together 1,000 requirements the optimal software would fulfill. There were only 11 areas where their processes didn't match the software and the commander made the decision to modify MSC processes to fit the Oracle software (Dean, 2001).

Management Commitment

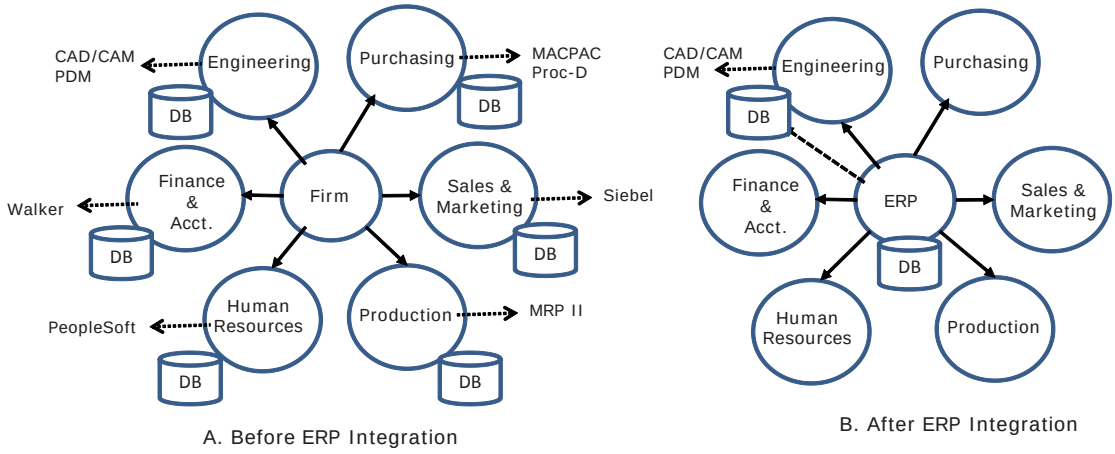
Management commitment for information systems (IS) projects has been researched extensively (Aloini et al., 2007, Aladwani 2002, Ravichandran and Rai 2000). They found empirically that management commitment this work can make human, monetary, and other important resources available for the IS project leading to a conducive and superior problem solving environment as well as

increase the likelihood of IS product quality and efficiency. Top management must consider the strategic implications of implementing an ERP solution (Davenport 1998). Management must also ask several questions before embarking on the project. Does the ERP system strengthen the company's competitive position? How might it erode the company's competitive position? How does ERP affect the organizational structure and the culture? What is the scope of the ERP implementation -- only a few functional units or the entire organization? Are there any alternatives that meet the company's needs better than an ERP system? If it is a multinational corporation, the management should be concerned about whether it would be better to roll the system out globally or restrict it to certain regional units? Management must be involved in every step of the ERP implementation. Handing over the ERP project to the IS department may result as a risk to the entire company's survival because of the ERP system's profound business implications. The top management must not only fund the project but also take an active role in leading the change. The success of a major project like an ERP implementation completely hinges on the strong, sustained commitment of top management. This commitment when percolated down through the organizational levels results in an overall organizational commitment. An overall organizational commitment that is very visible, well defined, and felt is a sure way to ensure a successful implementation (Bingi et al., 1999).

Integration

The system integration between existing information systems and ERP system is a technical problem which might complicate the entire ERP project (Liang and Lien, 2007). Firms have amassed multiple software that are either legacy systems or current add-ons. Different functions of the company have bought or developed different applications and are used to the software. Figure 1 illustrates before and after ERP implementation software integration in a business. Various functions of the firm and possible software associated with each one of the functions and how ERP can be integrated with some of the software are shown in the figure. Some of this software can be replaced

Figure 1. Integrated ERP software in a firm



by the newly proposed system and some of them can be integrated into the software. The databases (DB) of individual software, for instance product data management (PDM) database, can be integrated with ERP database as well. In this case, ERP serves as a backbone, and all the different software are bolted on to the ERP software. There is third-party software, middleware, which can be used to integrate software applications from several vendors to the ERP backbone. Middleware vendors concentrate only on the most popular packaged applications and tend to focus on the technical aspects of application interoperability rather than linking business processes. Many times, organizations have to develop their own interfaces for commercial software applications and the homegrown applications. Integration software also poses other kinds of problems when it comes to maintenance. It is a nightmare for information systems group to manage this software whenever there are changes and upgrades to either ERP software or other software that is integrated with the ERP system.

Until the end of 1996, for example, Dell Computer Corp. planned to roll out SAP's full R/3 suite, but it stopped after implementing only the HR modules. Jerry Gregoire, who joined the company as CIO that year, saw that a single software monolith would not be able to keep pace with Dell's extraordinary corporate growth—the company grows by a billion dollars every six to eight weeks. Instead Gregoire designed a flexible middleware architecture to allow the company to add or subtract applications quickly and selected software from a variety of vendors, including Glovia International LLC, to handle finance and manufacturing functions (Slater, 1999).

Reengineering

Reengineering existing business processes to the best business process standard is one of the activities in ERP implementation. ERP systems are process oriented; therefore only in a process-based organization they can completely express their integration potentiality (Guido and Pierluigi, 2008). ERP systems are built on best practices that are followed in the industry. One major benefit of ERP comes from reengineering the company's existing way of doing business. In this case, all the processes in a company conform to the ERP software and as a result the cost and benefits of aligning these processes with an ERP model could be very high. Since ERP systems such as SAP were built on a foundation of process best practices, it is probably easier and less expensive to change processes to adapt to SAP than the other way around. Many companies have reported good success from combining a SAP implementation with a reengineering project (Bingi et al., 1999).

The concept of reengineering traces its origins back to management theories developed as early as the nineteenth century. The purpose of reengineering is to make all of a firm's processes the best-in-class. In reengineering, there is one best way to conduct tasks.

Initiatives like Business Process Reengineering (BPR) and ERP promise radical improvements in relatively short periods of time. Processes, organization, structure and information technologies are the key components of reengineering. In implementing ERP, both business processes and information technology are combined into integrated software. This automates business processes across the enterprise and provides an organization with a well designed and managed information system. Companies like IBM, Texas Instruments, American Express, Johnson & Johnson, Chrysler, Ford, Shell oil and many others have achieved major reengineering successes. Many organizations have successfully implemented ERP systems and reported huge benefits. Yet many research studies estimate that at least 90 % of ERP implementations end up late or over budget and several failure stories are cited (Jarrar et al., 2000).

Consultants and Other Resources

Consultants are most often used as implementation partners at two to ten times the cost of the ERP software for the initial implementation (Karimi et al., 2007). The client, the management, must be involved in the consulting recruitment process. Simply trusting the consulting firm or not understanding what the consultants know is a risk. The more layers between the project manager and consulting company, the more the consultant's rates are reduced means that the consultants are willing to work for the lower rate and are usually the least knowledgeable.

SAP: ERP SOFTWARE

SAP the company was founded in Germany in 1972 by five ex-IBM engineers. SAP stands *for Systeme, Andwendungen, Produkte in der Datenverarbeitung* which - translated to English - means *Systems, Applications, Products in Data Processing.* SAP AG is now the third largest software maker in the world, with over 17,500 customers (including more than half of the world's 500 top companies). There are more than 50,000 installations of SAP, in over 120 countries, with more then 10 million users! SAP today is available in 46 country-specific versions, incorporating 28 languages including Kanji and other double-byte character languages. SAP R/3 is delivered to a customer with selected standard process turned on, and many other optional processes and features turned off. At the heart of SAP are about 10,000 tables which control the way the processes are executed. Configuration is the process of adjusting the settings of these tables to get SAP to run the way companies want it to. Functionality included is truly enterprise wide including: Financial Accounting (e.g. general ledger, accounts receivable etc), Management Accounting (e.g. cost centers, profitability analysis etc), Sales, Distribution, Manufacturing, Production Planning, Purchasing, Human Resources, Payroll, etc.

SAP is an integrated client/server software application. The features of SAP are as follows:

- Centralized database (ORACLE)
- Planning functions like Material Resource Planning
- Reporting functions
- Business workflow to simulate the business
- Development workbench that uses a language called ABAP (Advanced Business Application Programming)
- Implement Management Guide (IMG) used to configure the system
- An integration of finance, accounting, production planning, logistics, sales, material management, plant maintenance, human resources, etc.

SAP Database

The SAP repository or the data dictionary serves primarily as a tool to enter, manage, delete and evaluate company information. The rules for structuring this information are consistent with the concepts of the relational data model using tables and fields. The data repository is active at all times and therefore the information is always up-to-date and available to all authorized users at all times. The SAP repository is built on a table structure. A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data. As shown the Figure 2, there are four different tables for configuration, control, master data and transaction data. System configuration tables are tables that define the structure of a system. An example of this type of table is one that defines the peripherals such as printers. To customize the system, the other three tables are used. Control tables define

Figure 2. SAP table structure

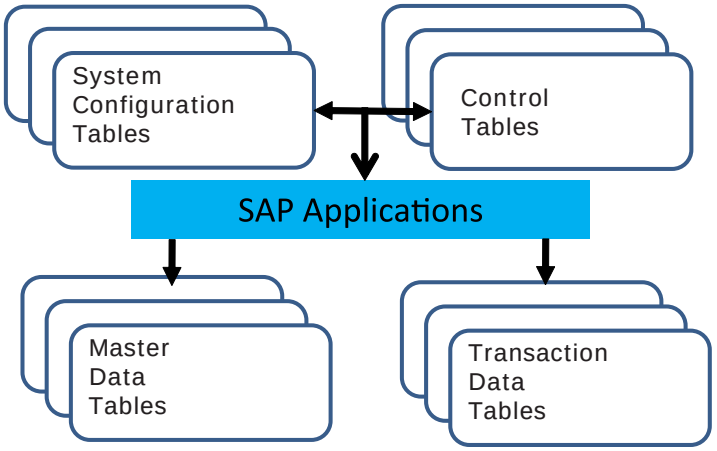


Figure 3. SAP modules

FI (Financial Accounting) • General ledger • Book close • Tax • Accounts receivable • Accounts payable • Consolidation • Special ledgers	**CO (Controlling)** • Cost elements • Cost centers • Profit centers • Internal orders • Activity based costing • Product costing	**MM (Materials Management)** • Requisitions • Purchase orders • Goods receipts • Inventory management • BOMs
SD (Sales & Distribution) • RFQ • Sales orders • Pricing • Picking • Packing • Shipping	**CA (Cross Applications)** • WF – workflow • BW – business warehouse • Office – for email • Workplace • Industry solutions •CRM, PLM, SRM, APO etc	**PP (Production Planning)** • Capacity planning • Master production scheduling • Material requirements planning • Shop floor
AM (Asset Management) • Purchase •Sale •Depreciation •Tracking	**HR (Human Resources)** • Employment history • Payroll • Training • Career management • Succession planning	**QM (Human Resources)** • Planning • Execution • Inspections • Certificates

functions that guide the users their activities. For example, a control table might be designed such that a material master data is entered before a purchase order is accepted. Control tables contain the structure or the process of the company. These tables contain data such as which plants are related which products, which sales organizations are related which products, etc. Master data tables

define customers, vendors, materials, equipment, etc. Customer master data might include customer name, address, contact information, etc. This table might also be related to other tables that represent the location of warehouse or repair shops that support the customer. Transaction data table represents the daily operations data such as sales orders, invoices and shipments. SAP modules are shown in Figure 3.

Business Case for ERP Implementation: IT Strategy

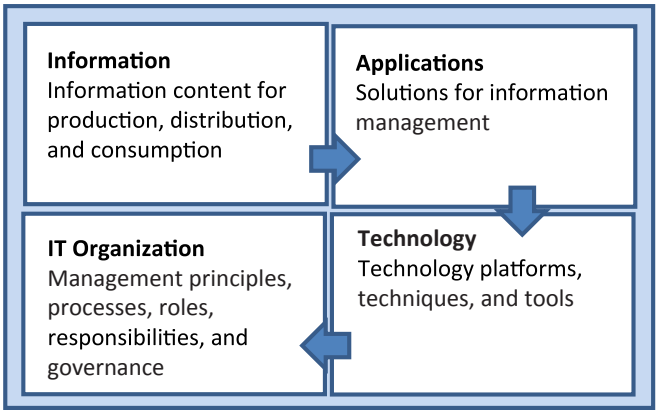
Companies implement ERP for many reasons. Y2K (Year 2000) was a costly fix for many companies and many of them resorted to ERP as a strategy as an easy, less costly to fix the Y2K problem. However, these companies found out that the reality was that ERP takes more time, more difficult to install, and cost as much if not more. As a result of the cost and time involved with the ERP implementation, a business case is needed. The thought process of IT department in many of the organizations can be generalized as below in Figure 4 (Norris et al. 1998). Figure 4 shows the four main components in this thought process. Information is gathered to be consumed and this information is managed by applications. The applications have to be created and managed by technology. Technology has to be managed by IT organization with its roles and responsibilities, and governance. As Norris et al. (1998) points out IT organizations need to answer the following questions on information:

1. What is the information?
2. Where does this information come from?
3. How do we create information from data?
4. How does information get distributed?
5. How does information get consumed?

The answers the above questions lead to the software applications. This is the level that users of IT and developers of IT deal with most of the time. Here the following questions need to be answered:

1. What applications does the company currently use to collect, collate, compute, and transform to useful information, store, distribute, and retrieve data?

Figure 4. IT thought process in a company (Adapted from Norris et al., 1998)



2. Is there a better application software available to carry out these tasks in a more efficient, less time-consuming, less resource-consuming and less expensive way?
3. Is there software available as Complete-Off-The-Shelf (COTS)?
4. By using this software, could the company redirect resources into other areas of the business in which it can establish and maintain competitive advantage?

Then the decision on hardware begins especially on technology platforms, techniques, servers, tools, etc. Each company has a different perspective on each one of these objects. For example, SAP is platform independent and has as a company provided an open-architecture presence which is an advantage for corporations. Finally, the questions to be answered for the IT organization are as follows:

1. What management principles will govern IT organization?
2. What processes for system development, operations, support and maintenance are used?
3. What are the roles and responsibilities of all the individuals in the organization?
4. What is the relationship as in power structure between the IT organization and the business units?

This thought process need to be followed before the implementation of SAP which will ensure smooth implementation. Once an IT strategy has been developed, and once an ERP solution is determined to be right, a company is ready to build a business case. Business case involves analysis of cost and benefits and determining the payback period for the expenses incurred for the implementation and getting the approvals from leadership.

Norris et al. (1998) shows how the complexity of the implementation, as well as the project risk and cost, increases as a company moves out on the continuum of the degree of business process change and up along the continuum of organizational change. As the level of business process change is increased, the level of organizational change necessary to carry it out must also increase. The key to successful implementation is assertive control over the project's scope and over the degree of business process changes to be undertaken. Clear objectives and justification (Weston, 2001), evaluation and selection (Van Everdingen et al., 2000), alignment between the ERP package and business strategies and requirements (Somers and Nelson, 2003) as part of the business case are critical to the success of implementation. To build a cost-based business case for ERP, firms need to extract savings that depend on ERP alone from the total savings to be had from ERP together with other sources. Wagle (1998) proposed five steps that include:

1. Create a base case of year-by-year savings from cost cuts that could be made without the ERP system in place.
2. Create an ERP case of year-by-year savings that could be made with ERP. This should include savings that do not depend on ERP as well as those that do.
3. Subtract the base-case savings (step 1) from the ERP-case savings (step 2) on a year-by-year basis, and calculate the net present value (NPV) of the residual cash flow. A positive NPV will indicate that the firm should probably proceed with the deployment of ERP.
4. If step 3 produces a positive NPV, conduct a sensitivity analysis to ensure that the business case is strong enough to withstand slippage and cost overruns.
5. Back-allocate all ERP system deployment costs to individual business units so that they can factor them into their planning. Ensure each unit is held responsible for producing the promised savings.

To summarize, the following factors need to be considered for the business case:

1. The decision must be driven by the business considerations, not merely the desire for new technology
2. Firms need to extract savings that depend on ERP alone from the total savings to be had from ERP together with other sources
3. ERP system will not solve business process and organizational dynamic problems.
4. ERP implementation cannot be delayed until the company is able to use to its fullest ability.

SAP IMPLEMENTATION STEPS

Botta-Genoulaz et al. (2005) in their survey of research literature note that recent research is on ERP post-implementation issues, customization of ERP projects, and the sociological issues of ERP systems, and return on investment of ERP systems. Implementation should depend on the size of the company (Mabert et al., 2003) and industrial sector (Wu and Wang, 2003). The roadmap to ERP or SAP is shown in the figure 5. The roadmap consists of five key functions that include:

- Project Preparation
- Business Blueprint
- Realization
- Final Preparation
- Go Live and Support

Each one of these functionalities has various steps. The following figure shows the implementation steps for enterprise software. The design and implementation phase consists of the following functionalities:

- Establishing hardware and SAP settings
- Establishing the key master data

Figure 5. SAP implementation roadmap

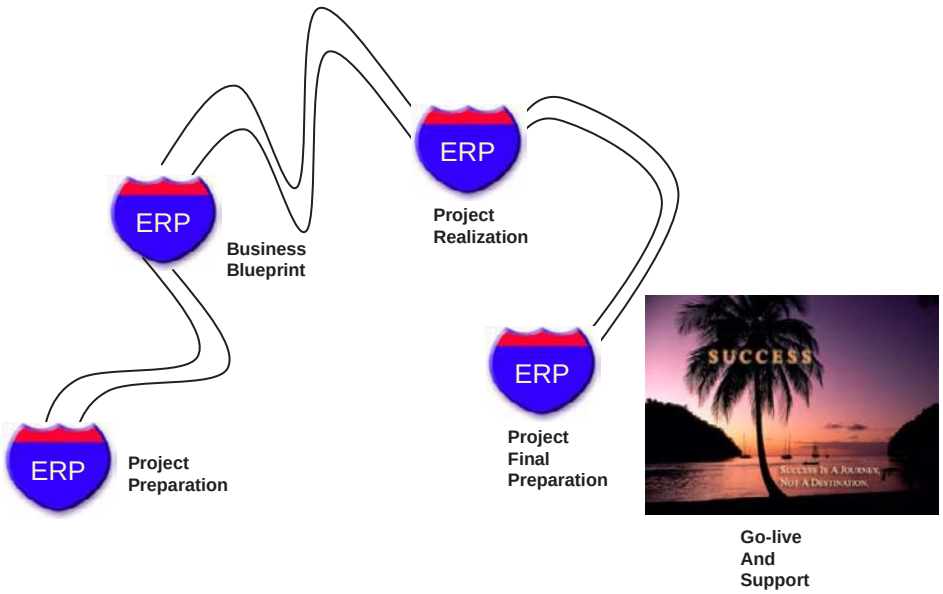
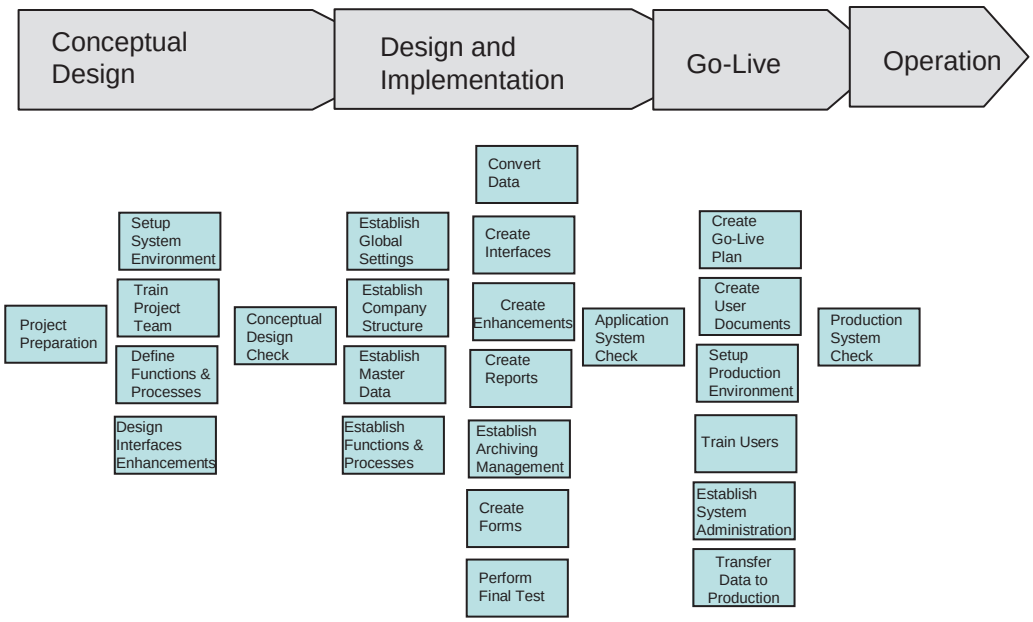


Figure 6. SAP implementation model



- Establishing the processes and functions inside the organization
- Implementing Reports, Interfaces, Conversions and Enhancements (RICE)
- Testing the system
- Training the users
- Go-Live
- Post production maintenance and support

The reason that ERP system implementations are time-consuming and costly is because of the complex nature of the implementation and the dynamics of the implementation involving with the organization. The basic functionalities described above may be expanded and is shown in Table 1.

Project Preparation

The purpose of this activity is to start detailed planning for the project. There are many basic issues to resolve at this early stage to ensure the project's success. The principal elements that determine how the ERP project is accomplished will be established in this phase. These activities set the groundwork for project kickoff, and provide a baseline of information to be referenced throughout the implementation. For all project team members to operate in an effective manner, it is necessary that project standards and procedures are established at the beginning of the project, and then communicated to all team members. Project planning is an ongoing cycle and should be constantly refined. Even when the project plan is created, it must be made clear that project planning is a continual, rolling process. As the project progresses, fewer revisions are necessary. In the SAP Business-to-Business Procurement system, requirement coverage requests for non-production materials or services are directly processed further, and purchase orders are placed. If data is missing for purchase orders, the Materi-

Table 1. Basic functionalities in SAP implementation

Project Preparation	**Business Blueprint**
Initial Project Planning	Project Management Business Blueprint Phase
Project Procedures	Organizational Change Management Blueprint
Create Training Plans	Technical Design Planning
Project Kickoff	Establish Development System Environment
Quality Management Project Preparation Phase	Design Training Plans
	Organizational Structure Definition
Realization	Business Process Definition
Project Management Realization	Business Process Definition - Final
Organizational Change Management Realization	Quality Management Business Blueprint Phase
Create Training Materials	
Baseline Configuration and Confirmation	
Develop System Test Plans	**Final Preparation**
Establish Quality Assurance Environment	Project Management Final Preparation Phase
Establish Production Environment	Deliver End User Training
Final Configuration and Confirmation	System Management
Prepare and Coordinate ABAP Development	Conduct System Tests
Develop Conversion Programs	Detailed Project Planning
Develop Application Interface Programs	Cutover
Develop Enhancements	Quality Management Final Preparation Phase
Create Reports	
Create Forms	
Establish User Role and Authorization Concept	**Go Live and Support**
Establish Data Archiving	Production Support
Final Integration Test	Ongoing KPI Management
Quality Management Realization Phase	Project End

als Management functions are used to complete them. The follow-on documents for the purchase orders are also created in the SAP Business-to-Business Procurement system (confirmation, goods receipt / service entry and invoice). This phase also includes defining end user training and

documentation strategy. This includes analyzing and developing the training and documentation strategy with the overall implementation strategy and project plan.

The results of this activity are:

- Project Plan (Work Plan, Budget Plan, Resource Plan).
- Project management standards and procedures.
- ERP implementation strategy.
- Approved project organization
- Configuration standards
- End user training and documentation strategy
- Testing strategy
- Post-implementation support strategy
- Design of the company's system landscape, including its setup and maintenance strategy
- Official start of the project, presentation of the input mentioned above, explanation for any issues or questions about the project from the kickoff meeting participants.
- All project team members have attended Level 1 courses and have an introduction to ERP, navigation skills, and knowledge of major integration points of the system.
- The formal sign-off of Phase 1.

Business Blueprint

The purpose of this activity is as follows to create the Business Blueprint, which is a detailed documentation of the results gathered during requirements workshops. Furthermore, the Business Blueprint documents the business process requirements of the company. This phase also establishes a proper cycle of project management activities to ensure that the implementation project is on target. It determines all project planning, controlling and updating activities. It also identifies where changes in the relationship between business processes and the organizational structure need to be managed, in consultation with departmental management. The same work package is also in the subsequent phases. It is also defined now what content the customer needs, what user populations and needs that content will serve, and how the content will be structured. Also, definitions of enhancement contexts that customers need to create and maintain their own materials are completed. In general, the idea is to start with a single context and then create additional contexts based on the first one. The relationships between enhancement contexts are called enhancement chains. Description of processes and sub-processes that are particular to the following business processes such as training management, performance assessment, and web content management are also completed in this phase. At this time, all user master records with roles and responsibilities along with authorizations for the project team members are also created.

The business process definitions create the Business Blueprint. In case of a rollout most of the company requirements should already be determined and mapped to the corresponding ERP software settings. The Business Blueprint serves as the conceptual master plan, and eventually becomes a detailed written document. This document shows the business requirements in detail, and serves as the basis for organization, configuration and, if necessary, development activities. End User Requirements Document, Current Data Warehouse and Information Access Environment Document, Master Data/Transaction Data Requirements Document, Data Mapping Document, Data Access Recommendations Document, and Data Management Requirements Document are also completed.

The outputs of this activity are:

- Regularly scheduled project status meetings
- Updated Project Plan
- Definition of end user roles and responsibilities
- Quality Reviews

- Content Requirements document, which identifies user populations, existing information, and new information requirements.
- A business blueprint
- The technical infrastructure (network, system, and front-end environment) is documented
- A signed Business Requirements Analysis Document to include Completed business process questions, Completed CI template, Completed KPI template

Realization

The purpose of this phase is to implement business and process requirements based on the Business Blueprint. The objectives are final implementation in the system, an overall test, and the release of the system for production (live) operation. The project team also receives relevant knowledge. All testing is completed in order to implement the completely functional system

RESULT

The outputs of this activity are:

- Implementation of all necessary enhancement contexts, enhancement releases, and enhancement context chains
- The data extraction is configured.
- Configured and tested ERP system.
- Tested programs to migrate into the QA environment
- ERP system meets the business application requirements as defined in the Business Blueprint.

Project Final Preparation

The purpose of this phase is to perform the established cycle of project management activities to keep the implementation project on target. This determines all project planning, controlling and updating activities, and identifies changes between business processes and the organizational structure. Project management and consulting management must work closely together to guarantee that the project stays on schedule. Also, technical and performance tests will be conducted to verify that the production environment is ready and can be supported for productive operation. These tests should be performed in the actual production environment or in an environment that closely represents the actual environment. At this time, the move from a pre-production environment to live production operation will be done. In addition, it is important to monitor system transactions and overall performance. During the process of going live, there are two critical periods. In the first few days, the team must execute the production support plan and check the results. Any issues or problems that occur in this period must be resolved as quickly as possible. Following the first few days of live operation, monitoring issues for the long term, particularly with reference to system performance, capacity and functions should be addressed.

The results of this activity are:

- All roles and responsibilities defined with proper backups in place
- All procedures and responsibilities documented and signed off
- The entire technical infrastructure is tested and validated, including failure scenarios and disaster recovery.
- The configuration of the system and the procedures defined for the technical environment are tested and validated.
- A stress test is carried out successfully. Its result determines the validity of the planned go-live date.
- All ERP system aspects and support organization is approved, and the production system is released.

- The output of this activity is the formal sign-off of Phase 4.

CONCLUSION

A brief ERP implementation procedure using the SAP implementation documentation as well as issues regarding ERP implementations are detailed in this chapter. ERP implementations in the recent years have raised a number of questions regarding its success. Many companies regard ERP as their one and only savior and many others despise that ERP as a single system has brought them to their knees. Regardless, many more companies, small to medium size companies in particular, are beginning to invest in ERP. An industrial practitioner from such small to medium companies needs to understand how to implement ERP. This study provides the necessary tools and background for the industrial practitioner to implement not only ERP systems but implement the next generation of enterprise applications as well.

REFERENCES

Aladwani, A. M. (2002). An Integrated Performance Model of Information Systems Projects. *Journal of Management Information Systems, 19* (1), 185-210.

Al-Mudimigh, A., Zairi, M., & Al-Mashari, M. (2001). ERP software implementation: an integrative framework. *European Journal of Information Systems, 10*(4), 216-226.

Aloini, D., Dulmin, R., & Mininno, V. (2007). Risk management in ERP project introduction: Review of the literature. *Information & Management, 44*(6), 547-567.

Bingi, P., Sharma, M., & Godla, J. 1999. Critical issues affecting an ERP implementation. *Information Systems Management, 16*(3), 7-14.

Capaldo, G., & Pierluigi, R. (2008). A methodological proposal to assess the feasibility of ERP systems implementation strategies. *Proceedings of the 41st Hawaii International Conference on System Sciences*, Waikoloa, Hawaii, January 7-10, 401-401.

Davenport, T. (1998). Putting the Enterprise into the Enterprise System. *Harvard Business Review, 74*(4), 121-131.

Dean, J. (2001). Weathering the ERP Storm. *Government Executive*, July 1, 2001

Gebauer, J., & Lee, F. (2008). Enterprise system flexibility and implementation strategies – Aligning theory with evidence from a case study. *Information Systems Management, 25*(1), 71 – 82.

Gefen, D., & Ragowsky, A. (2005). A multi-level approach to measuring the benefits of an ERP system in manufacturing firms. *Information Systems Management, 22*(1), 18–25.

Hawksworth, M. (2007). *Six steps to ERP implementation.* IFS White Paper, IFS.

Hendricks, K. B., Singhal, V. R., & Stratman, J. K. (2007). The impact of enterprise systems on corporate performance: A study of ERP, SCM, and CRM system implementations. *Journal of Operations Management, 25*(1), 65-82.

Hilson, G. (2001). Human factor plays big role in IT failures. *Computing Canada, 27*(6).

Jarrar, Y. F., Al-Mudimigh, A., & Zairi, M. (2000). ERP implementation critical success factors-the role and impact of business process management. *Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology, 1*, 122 – 127.

Karim, J., Somers, T. M., & Bhattacherjee, A. (2007). The impact of ERP implementation on business process outcomes: A factor-based study. *Journal of Management Information Systems, 24*(1),101–134.

Ketbi, O. A., Azaizeh, A., Carrico, W., Cook, R., & Cooke, D. (2002). 2002 Industry Studies: Advanced Manufacturing. Report Number: A105624, Industrial Coll of the Armed Forces, Washington, D.C.

Botta-Genoulaz, V., Millet, P. A., & Grabot, B. (2005). A survey on the recent research literature on ERP systems. *Computers in Industry, 56*, 510–522.

Koch, C. (2002). The ABCs of ERP. *CIO Magazine*, March 7, 2002.

Laukkanen, S., Sarpola, S., & Hallikainen, P. (2007). Enterprise size matters: objectives and constraints of ERP adoption. *Journal of Enterprise Information Management, 20*(3), 319–334.

Liang, S., & Lien, C. (2007). Selecting the optimal ERP software by combining the ISO 9126 standard and fuzzy AHP approach. *Contemporary Management Research, 3*(1), 23-44.

Mabert, V. A., Soni, A. K., & Venkataramanan, M. A. (2003). The impact of organization size on enterprise resource planning (ERP) implementations in the U.S. manufacturing sector. *Omega , 31*, 235-246.

Moon, Y. B. (2007). Enterprise resource planning (ERP): A review of the literature. *International Journal of Management and Enterprise Development, 4*(3), 235-264.

Norris, G., Wright, I., Hurley, J., Dunleavy, J., & Gibson, A. (1998). *SAP: An Executive's Comprehensive Guide*. John Wiley & Sons, Inc., New York, NY.

Ranganathan, C., & Brown, C. V. (2006). ERP investments and the market value of firms: Toward an understanding of influential ERP project variables. *Information Systems Research, 17*(2), 145-161.

Ravichandran. T., & Rai. A. (2000). Quality management in systems development: An organizational system perspective. *MIS Quarterly, 24*(3), 381-416.

Sharma, R., & Yetton, P. (2003). The contingent effects of management support and task interdependence on successful information systems implementation. *MIS Quarterly, 27*(4), 533-555.

Slater, D. (1999). How to choose the right ERP software package. *CIO*, February 16, 1999.

Somers, T. M., & Nelson, K. G. (2003). The impact of strategy and integration mechanisms on enterprises system value: empirical evidence from manufacturing firms. *European Journal of Operational Research, 146*, 315-38.

Stedman, C. (1998). Global ERP rollouts present cross-border problems. *Computerworld, 32* (47), 10.

Van Everdingen, Y., Van Hillegersberg, J., & Waarts, E. (2000). ERP adoption by European midsize companies. *Communications of the ACM, 43*(4), 27-31.

Wagle, D. (1998). The case for ERP systems. *The McKinsey Quarterly, 2*, 130-138.

Weston, F. C. (2001). ERP implementation and project management. *Production and Inventory Management Journal, 42*(3/4), 75-80.

Wu, J. H., & Wang, Y. M. (2003). Enterprise resource planning experience in Taiwan: An empirical study and comparative analysis. 30th *Annual Hawaii International Conference on System Science (HICCS '03)*, Big Island, Hawaii, January 6-9.

Wu, L., Ong, C., & Hsu, Y. (2008). Active ERP implementation management: A Real Options perspective. *Journal of Systems and Software, 81*(6), 1039-1050.

KEY TERMS

Customization: Customization refers to modifications to the original software that is typically not supported by the software vendor and those needed by the customer because of their unique business processes.

Enterprise Resource Planning (ERP): Software systems for business management that integrates functional areas such as planning, manufacturing, sales, marketing, distribution, accounting, finances, human resource management, project management, inventory management, service and maintenance, transportation, and e-business.

Enterprise Systems: Enterprise systems are software that provides solutions to an integrated business organization.

Implementation: Implementation consists of defining a project, putting together project teams, reengineering of existing business processes, customizing the software to reflect new business processes, testing the software in organizational environment such that the software is usable for the organizational users.

Integration: ERP software integration is the process of integrating ERP systems with other enterprise information resources or systems within an enterprise.

Management Commitment: Management commitment is direct participation by the highest level executives in a specific and critically important aspect or program of an organization.

This work was previously published in Handbook of Research on Enterprise Systems, edited by J. N.D. Gupta, S. K. Sharma & M. A. Rashid, pp. 245-261, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 4.25
Teaching Operations Management with Enterprise Software

R. Lawrence LaForge
Clemson University, USA

ABSTRACT

Enterprise systems technology is used to enhance the teaching of operations management through development and operation of a virtual manufacturing enterprise. An ongoing, real-time simulation is conducted in which operations management issues in the fictitious factory must be addressed on a daily basis. The virtual manufacturing enterprise is integrated into an operations management course to facilitate understanding of the dynamic and interrelated nature of operations planning and control in a complex manufacturing environment. Enterprise software supports the primary learning objective of understanding how operations management decisions affect customer service, capacity, inventory, and costs.

INTRODUCTION

This chapter presents an approach to teaching operations management (OM) with enterprise systems technology. The approach described here is based on the premise that the topics taught in operations management courses are dynamic and interrelated. Therefore, the teaching and learning of operations management should address not only the content of each specific topic, but also the dynamic interrelationships among the topics.

The integration of OM topics is extremely difficult to accomplish in a traditional classroom setting. There are several excellent OM textbooks available in the market, and they generally provide information needed to study the content of OM topics such as aggregate sales and operations planning, master production scheduling, material planning and control, capacity planning and control, and production activity control. However,

textbooks are a static medium that cannot capture the dynamic interrelationships between and among the topics. The topics in an OM textbook must necessarily be presented in sequential fashion organized by chapter or unit, and conventional testing typically focuses on the content (issues, concepts, tools, techniques) related to each topic. True insights into the connections among the topics are extremely difficult or impossible to glean from even the best textbooks.

As an example of the interrelationship of operations management issues, consider the topics of aggregate planning and production activity control. Aggregate sales and operations planning deals with the company's overall strategy for meeting anticipated demand of broadly-defined product families over a planning horizon of 12-18 months. Production activity control, on the other hand, deals with day-to-day (or hour-to-hour) scheduling and sequencing issues for internal shop orders to make component parts or assemblies needed for specific products. The two topics are at extreme ends of the continuum and are invariably covered in completely different sections of the typical OM text, but they are in fact highly interrelated. While we might be tempted to seek sophisticated algorithms and models to help with complex scheduling issues that occur on a daily basis, it may be that our day-to-day scheduling issues are the result of poor overall planning at the product family level.

The previous argument suggests that conventional classroom lectures, textbook readings, and end-of-chapter exercises are *necessary* but *not sufficient* to gain insights needed to understand operations management. This chapter describes an ongoing project in which the desired synergy is addressed by the introduction and use of enterprise-wide system (ES) technology in the operations management classroom.

The objectives of this chapter are to raise the level of awareness regarding the need for active learning approaches to operations management, to describe in detail an approach to teaching OM with enterprise technology, and to discuss lessons learned that may be of benefit to other scholars and teachers interested in this approach.

BACKGROUND

The practice and teaching of operations management has been impacted significantly by advancements in information technology (Manetti, 2001; Rondeau & Litteral, 2001). The development of material requirements planning (MRP) systems in the 1970's revolutionized thinking about how to manage materials in a manufacturing environment. This provided an alternative to economic lot size models that assumed that demand for all inventory items was independent, and it promoted more of a systems perspective to materials management. This was followed by manufacturing resource planning (MRPII) systems that provided additional functionality related to capacity planning, limited financial analysis, and "what-if" planning. More recent enterprise resource planning (ERP) systems aim to link together the various functional areas of operations, accounting, finance, customer relationship management, and human resources. ERP systems can be enhanced with advanced planning system (APS) modules for obtaining near-optimal scheduling, manufacturing execution system (MES) modules for shop floor control, and/or supply chain management (SCM) modules designed to integrate planning and execution with suppliers and customers.

A key aspect of this evolution is integration. Organizations have made significant investments in hardware, software, and training to install, maintain, and use enterprise-wide systems designed to better integrate their activities (Hitt, Wu, & Zhou, 2002; Mabert, Soni, & Venkataramanan, 2000). It is logical to assume that collegiate schools of business would also value, and seek to achieve, higher levels of integration in the topics taught in the curriculum (Cannon, Klein, Koste, & Magal, 2004; LaForge & Busing, 2000).

In operations management, the objective of topic integration could be pursued in two ways. One way is to seek greater insights into the interrelationships of topics taught in a given OM course. Another way is to focus on the impact of operations management decisions on other functional areas in the organization such as accounting, finance, marketing, and human resources or on inter-organizational issues with suppliers and customers. That is, the goal could be integration within a given course and/or integration across courses. This chapter provides details on the first approach, which has been implemented under the assumption that one must achieve a high level of understanding of operations management issues before assessing their impact on other areas. However, the chapter also includes a discussion of efforts to achieve "across course" integration, which builds on the application of enterprise software in the operations management course described next.

THE OPERATIONS MANAGEMENT COURSE

The section describes an operations management course that has been enhanced by the use of enterprise software. The course is an advanced, undergraduate course in operations planning and control taken by primarily by management majors at Clemson University. The course topics are aggregate sales and operations planning, master production scheduling, material planning and control, capacity planning and control, and production activity control. The textbook used in the course is a primary study reference for the *Certified in Production and Inventory Management* (CPIM) examinations offered by APICS — The Society for Operations Management. Traditional classroom activities such as textbook readings, lectures, end-of-chapter exercises, and testing of content are involved.

A primary learning objective of the course is to understand and appreciate the dynamic interrelationships among the course topics. As discussed earlier, the traditional classroom activities are necessary to establish content knowledge of each major topic, but do not address the integration of the OM activities in a meaningful way. The issue of integration of OM topics in the course is addressed in the same way that a manufacturing company would address the issue of integration of key OM activities in its facilities — through the introduction of information systems technology. The technology is used to create and operate a virtual manufacturing enterprise in which the OM issues under study are carried out simultaneously on a daily basis. The virtual manufacturing enterprise is the focal point for active learning opportunities that address the integration issues and learning objectives. Table 1 shows details from the course syllabus regarding the body of knowledge and learning objectives.

In the following section, the virtual manufacturing enterprise is described in terms of four components necessary to create and maintain it. This will be followed by a detailed description of how the virtual enterprise is used to address the learning objective.

The Virtual Manufacturing Enterprise

The virtual manufacturing enterprise can be described by the four components shown in Figure 1. These elements consist of the body of knowledge, manufacturing database, enterprise software, and enterprise information. The following sections explain the four components of the virtual manufacturing enterprise and address key issues in developing each component.

Body of Knowledge

The most important component of this approach to teaching operations management is a clear

Table 1. Syllabus information for the OM course

Management 402
Operations Planning and Control
Clemson University

NATURE OF THE COURSE:

This course examines important concepts and procedures involved in managing the operations of a business enterprise. All class meetings take place in the Manufacturing Management Laboratory. This laboratory provides access to a realistic simulation of a business operation utilizing a state-of-the-art enterprise resource planning (ERP) system that is widely used in industry.

Students will study important concepts, issues, and procedures of an operations planning and control system, and then put this knowledge to work in a virtual manufacturing enterprise that is supported by the ERP system. Class meetings will be a combination of lecture and student participation. The basic concepts and procedures will be presented through instructor lecture, reading assignments, and class/homework exercises. The concepts will be reinforced through "hands on" experience with the ERP system and the virtual enterprise. Particular attention will be paid to the interaction of planning/control activities, and how decisions made at one level of the planning process affect output and performance at lower levels.

A model manufacturing company — Orange Office Products — is used to illustrate many of the topics in the course. Orange Office Products is a fictitious manufacturer of office furniture equipment that was developed specifically for use in this course. The database for Orange Office Products has been installed on the ERP system, providing live information about products, inventory levels, plant capacity, manufacturing orders, purchase orders, and much more.

Orange Office Products is an ongoing, real-time simulation of a manufacturing operation. The database changes daily to reflect the activities of the company. Students in Management 402 will observe and participate in many of these important operations management activities.

MAJOR TOPICS IN THE COURSE:

1. Overview of Planning and Control Systems
2. Material Planning and Control
3. Production Activity Control
4. Aggregate Sales & Operations Planning
5. Master Production Scheduling
6. Capacity Planning

OBJECTIVES OF THE COURSE:

Students who complete this course should be able to:

1. Explain the basic components of an operations planning and control system;
2. Understand the interrelationships among planning and control activities in a business operation, and their impact on customer service, inventory, capacity, and costs.
3. Develop, apply, and interpret the results of basic procedures for inventory planning, capacity management, shop floor control, aggregate production planning, and master scheduling;
4. Understand and appreciate the importance of operations management and the role of information technology in modern planning and control systems.

continued on

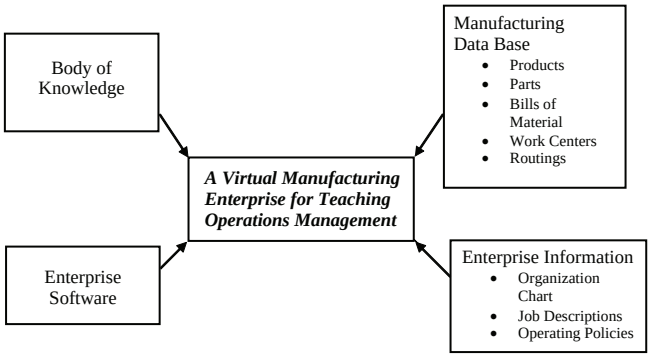
Table 1. Syllabus information for the OM course (continued)

OBJECTIVES OF THE COURSE:

Students who complete this course should be able to:

1. Explain the basic components of an operations planning and control system;
2. Understand the interrelationships among planning and control activities in a business operation, and their impact on customer service, inventory, capacity, and costs.
3. Develop, apply, and interpret the results of basic procedures for inventory planning, capacity management, shop floor control, aggregate production planning, and master scheduling;
4. Understand and appreciate the importance of operations management and the role of information technology in modern planning and control systems.

Figure 1. Components of the virtual manufacturing enterprise



statement of the body of knowledge to be taught. All other components of the virtual manufacturing enterprise, and all other activities, should be undertaken with the objective of addressing or supporting the body of knowledge in some way. This is important because without a clear statement of the body of knowledge, one may fall into the trap of teaching what a software package does rather than teaching what the curriculum says. As shown in Table 1, the body of knowledge for the OM course described here consists of concepts, issues, and tools for performing the previously-mentioned topics of aggregate sales and operations planning, master production scheduling, material planning and control, capacity planning and control, and production activity control.

Manufacturing Database

The database provides needed details of products, parts, bills of material, routings, work centers, customers, and suppliers. These details are essential for developing a realistic enterprise model capable of capturing the issues in the body of knowledge. While realism is needed, this does not mean that the database must consist of hundreds of products and thousands of parts. Realism in the database can and should be achieved through complex interrelationships among the elements, not through sheer size. Experience here suggests that size of the database is not as important as well conceived connections among the elements.

An original database for a fictitious manufacturing enterprise was created for the operations management course in this project. The decision to create an original database, rather than use sample data provided by the software vendor, was made to allow complete control over the size of the data set and the interacting components that result in operations management issues addressed in the course.

The fictitious manufacturer produces metal-based office furniture products. This environment was chosen because the products are common items that are easy to conceptualize for students with little or no practical experience in operations management. The database contains technical descriptions of five standard office furniture products that are manufactured in a make-to-stock environment: (a) three-shelf adjustable book case, (b) three-drawer file cabinet, (c) two-drawer work desk, (d) standard desk chair, and (e) one-shelf credenza. In addition to manufacturing details, a chart of accounts and other financial information were created to facilitate use of the virtual enterprise in accounting and other business courses, as discussed later in the chapter.

The inventory portion of the manufacturing database consists of 45 total items, including the five finished products. Twenty-two of the items are manufactured in house, while 23 are purchased from suppliers. The manufactured items typically consist of metal parts (handles, frames, brackets, shelf units, etc.), while the purchased items include raw sheet metal, hardware, paint, packaging materials, wooden parts, and accessories.

Bills of material were created to link the items needed to make each product. These documents are multi-level in nature, reflecting a process that flows from purchased materials to manufactured parts to assemblies to final products. Several items are common to different products, including handles, shelf units, and a major drawer assembly.

The manufactured items in each bill of material are made in a production facility consisting of nine distinct work centers. A routing file was created for each manufactured item that identifies the work center sequence required and the standard setup time and standard run time per unit for processing the item. Multiple operations are required to make each manufactured item. Defined shift lengths dictate the available capacity of each work center.

The database supports the body of knowledge by being small enough to allow manual calculation and tracking of some issues, but large enough to capture complex interrelationships impacted by operations management decisions. The major challenge in creating the manufacturing database was establishing parameter values (routing steps, standards for setup times and run times, work center capacities, planned lead times, etc.) that result in shop loads that introduce meaningful operations management issues to students. Capacity planning modules in the enterprise software assist in this issue by providing projected workloads for a given configuration of parameter values, facilitating adjustment of planning parameters as necessary. Development of the database is an ongoing activity, allowing the introduction of new parts, revised routings, adjusted planned lead times, and various other issues.

Enterprise Resource Planning Software

Introduction of enterprise resource planning software into an academic course obviously requires an answer to the following question: What software should be used? In this context, software should be viewed as the enabling technology that allows issues in the body of knowledge to "come to life" in a dynamic and realistic way. The functionality needed in the software is defined by the body of knowledge. Ideally, software selection should focus on systems that implement generally-accepted methods to address and integrate issues under study. That is, the software should be consistent with the methods described in the text and established in the curriculum. What the software does is give life to the issues and help create a dynamic and meaningful context for studying them.

Realistically, other issues come into play in the software selection decision. Costs associated with hardware, implementation, training, and maintenance must be considered. With the budget constraints and limited technical support that exist at many universities, these issues must be carefully weighed to ensure that the system can be sustained. However, the compatibility of the software with the body of knowledge must be the overriding consideration. It would be indefensible to introduce a software system into the curriculum simply because it was available or easy to maintain without regard to the value it adds to pre-established learning objectives for the subject matter in the course or curriculum. The teaching approach should not be software specific.

Fortunately, there are many software options available that are consistent with generally-accepted practices in operations management. In addition, many software vendors have special pricing arrangements, or make free donations of software and support, for educational institutions using their products in the curriculum. Over the twenty years in which the project described here has been under development, four different software systems, each from a different vendor, have been utilized.

Current enterprise software system support for this project comes from the Microsoft Dynamics Academic Alliance. This alliance not only provides software, but also support, training, and opportunities to interact with faculty at other institutions. Specifically, the activities in the virtual manufacturing enterprise are supported by the Manufacturing Series of Microsoft Dynamics GP 9.0. The system runs on a Dell PowerEdge Server 2600, with the client applications available through the Clemson University network to smart classrooms and laboratories throughout the campus.

Enterprise Information

This component of the virtual manufacturing enterprise refers to a description of the enterprise in sufficient detail to establish the context in which the simulated company operates. This background information is important for students to get the feel of a realistic enterprise. In addition to background information on the company, this material also includes defined roles in the virtual manufacturing enterprise that relate to the body of knowledge, as well as policies and operating procedures. Details of this information will be presented later in the chapter.

OPERATING THE VIRTUAL MANUFACTURING ENTERPRISE

The virtual enterprise is operated in a continuous, real-time mode without regard to the starting and ending dates of academic terms. The calendar in the enterprise system is maintained so that the factory is closed on academic holidays and other "down time" in the academic calendar. However,

the database is never reset back to initial conditions. In this manner, the virtual manufacturing enterprise is, in every sense, a going concern.

Daily decision making and transaction processing are required in the virtual manufacturing enterprise just they are in a real company. The virtual factory operates in real-time mode in that daily decisions must be made on matters related to accepting customer orders, shipping products, updating operating plans, managing materials, allocating capacity, and implementing scheduled work activity. A number of random events can be incorporated into the virtual factory that must be considered in the daily decision making, such as actual customer demand, vendor performance, and unexpected down time in work centers. Daily transaction processing is required to report all decisions, activities, and events affecting the virtual enterprise and keep the database up to date. Each day materials and products are moving through the virtual factory from receipts of raw materials, and processing of manufacturing and customer orders. At any time that an inquiry is made into the information system, the user sees current dates related to all activities. In short, the system is live and operated on a day-to-day basis.

The previous operational decisions were made to achieve the realism needed for meaningful active learning tasks that capture the dynamic interrelationships among the operations management topics taught in the course. Without such realism, only static inquiries into the system would be possible. While important to achieving the learning objectives of the course, this realism comes with a price. There is an "overhead" associated with operating the virtual enterprise in the form of instructor or graduate assistant time needed to perform the daily decision making and transaction processing necessary to keep the database current. To minimize the overhead of the daily routine, all activities are periodically evaluated to ensure that they add value with regard to accomplishing the learning objectives.

New features or issues that could be incorporated into the operation of the virtual factory are considered by weighing the potential value added to student learning against the additional overhead required to implement and maintain the new feature. For example, should routines be added to introduce uncertainty into all activities of the factory (e.g., processing times, lead times, customer demand, machine availability), or will uncertainty in final customer demand introduce enough uncertainty to create a meaningful scenario? In this project, uncertainty in final customer demand suffices, but that may not necessarily be the case in other situations with more experienced students in graduate programs, or in courses with different learning objectives.

USING THE VIRTUAL ENTERPRISE TO ACHIEVE THE LEARNING OBJECTIVES

The virtual enterprise is integrated into the operations management course in two ways that are described in detail in the following sections. First, activities are described for integrating the virtual factory into routine class activities. Next, a major project is described in which selected students assume managerial roles in the virtual enterprise and participate in daily decision making and transaction processing.

Class Activities Using the Virtual Manufacturing Enterprise

Students in the operations management course are seated at a work station and have access to the live ERP system during all class sessions. This enables the instructor to enhance traditional classroom instruction with the live system, providing a meaningful context for the course topics. The virtual manufacturing enterprise enhances, but does not replace, traditional classroom activities

such as lectures, discussion, and problem solving. After basic concepts and methods for a particular topic, such as material planning, have been addressed, students access data in the live system to explore key issues within the context of the virtual enterprise.

There are generally three objectives in using the virtual factory and enterprise software in the manner described above to enhance traditional classroom instruction: (1) identifying key concepts or issues, (2) verifying outputs of planning functions, and (3) understanding dynamic interrelationships.

Identifying Key Concepts or Issues

This provides a meaningful context for key concepts in the text that might appear to students to be abstract, and it facilitates more meaningful discussion of the issues. Rather than memorize a concept or definition, students have an opportunity to see practical implications of the issue. For example, material planning activities in the MRP module of the system rely on user-supplied parameters referred to as planned lead times. Memorizing the definition of "planned lead time" is one thing, but coming to grips with the fact that MRP-based planning cannot begin until we enter these numbers, understanding what one must do to establish reasonable values for this planning parameter, and realizing the implications of using "bad" numbers are much more valuable lessons for operations management students.

Verifying Outputs of Planning Functions

This enables students to test their knowledge of planning methods with live data in the ERP system. With a carefully constructed database that is realistic in connecting the various elements but small in size, students can manually verify some outputs of the system. For example, the MRP record for a purchased item in the database may show a planned order release on the current date for certain quantity of the item. This means that we should launch a purchase order today to the supplier of the item for the specified quantity. Without the planned purchase order, we will not have sufficient quantity of the item to meet projected requirements for the item in the future. While most ERP training would focus on the transaction processing necessary to create and release the purchase order, the focus here is on the planning logic that resulted in the order recommendation in the first place. Why is the system telling us to do this?

Operations management students who have studied the basic logic of material requirements planning should be able to answer the above question by accessing relevant data from the live system (current master production schedule, bills of material, planned lead times, available inventory, and open orders), and applying the same MRP logic used to work end-of-chapter problems in the text. Successful completion of this exercise goes beyond simply accessing information in a computer system — it requires knowledge of how that information has been generated and, therefore, provides insights into the usefulness and potential limitations associated with the information. Further, because the virtual factory operates on a real time calendar and is updated daily, students can test their skills repeatedly since the numbers are constantly changing with new conditions in the factory.

Understanding Dynamic Interrelationships

This is a key learning objective and is facilitated by the live system in which the various operations planning and control activities are taking place simultaneously. Some key links between planning activities can be examined in classroom sessions in which changes are introduced into the system.

Table 2. Using the virtual manufacturing enterprise for class activities

Identifying Key Concepts and Issues

Using the live system to access data and discuss issues related to:

- Importance of planned lead times
- Impact of ordering policies
- Importance of inventory record accuracy
- Meaning and use of "floor stock"
- Impact of standard queue times in scheduling
- Differences between backward and forward scheduling

Verifying Outputs of Planning Functions

Applying standard methods in the textbook to verify data in the live system related to:

- Projected available inventory
- Available to promise
- Order recommendations in material requirements planning (MRP)
- Rescheduling messages for open orders
- Operation due dates for scheduled manufacturing orders
- Projected work center loads

Understanding Dynamic Interrelationships

Using the live system to assess cause-and-effect relationships, such as:

- Changes in the material plan needed to accommodate a special customer order
- Capacity needed for a change in the master production schedule
- Effect of reducing planned lead times on shop capacity
- Impact of shop downtime on meeting the master production schedule
- Effect of reducing setup times on shop capacity
- Impact of different scheduling approaches in the shop
- Material planning actions needed to overcome scrap on the production floor

For example, a change in the master production schedule of a finished product could be made during a class session to accommodate an unusually large customer order. This in turn will trigger a series of changes in material plans, capacity needs, and work center schedules that are explainable, tractable, and predictable. Working through these changes gives students insights into the dynamics of operations management that are not possible in a traditional setting. These are extremely important insights that help students understand trade-offs involved in balancing customer service objectives with inventory and capacity issues.

Table 2 provides additional examples of the use of the virtual manufacturing enterprise and ERP system to enhance class activities.

Class Project in the Virtual Manufacturing Enterprise

In the first half of the semester, all daily decision making and transaction processing necessary to run the virtual factory are done by the instructor and graduate assistant assigned to the project. During this period, students in the operations management course are engaged with the enterprise system through the class activities described above, but are limited to inquiry capabilities and have no decision making or transaction processing authority in the system.

At approximately mid semester, selected students take an active role in the daily management of the virtual enterprise as part of a course project. In the target OM course, selection is based on student performance, motivation, and career interest. Participating students perform all project activities during special laboratory hours held outside of the normal class meeting time, and must agree that a formal assessment of their performance in the project will become their final examination grade in the course. In this manner, the participants substitute an active learning project that addresses all course concepts for the traditional cumulative final examination.

The student roles relate to the body of knowledge in the course. The three major roles related to the target course are master production scheduler, inventory planner, and work center manager. Multiple students can occupy each of these roles at the same time. In addition, roles are defined for the plant manager, materials manager, and production manager. The plant manager role is performed by an assigned graduate assistant with significant work experience in operations management, while the materials manager and production manager are undergraduate students enrolled in a special independent study course who completed the target operations management course in a previous semester (and therefore previously served as master production scheduler, inventory planner, or work center manager). Brief job descriptions of each of these key roles are provided in Table 3.

The student roles require content knowledge of operations management topics acquired through traditional classroom activities. It is for that reason that assigned student roles go into effect near the midpoint of the semester, at which time the student team replaces the instructor and graduate assistant in the daily decision making and transaction processing necessary to keep the system live and current.

Enterprise policies are published that provide general guidelines for student activity in the virtual manufacturing enterprise and specific rules for assigned roles. General policies apply to all student participants and deal with communication and reporting responsibilities. Rules for specific roles place requirements and responsibilities on students as they deal with unexpected situations in performing their assigned duties. For example, work center managers may schedule overtime in their assigned work center in a given day, but only if they submit a request at least one day in advance with appropriate justification and receive approval from the plant manager. Master production schedulers may make any changes in the master schedule they feel necessary, but changes in the

Table 3. Key OM roles in the virtual manufacturing enterprise

Plant Manager (graduate assistant) Responsible for all activities of the Virtual Manufacturing Enterprise. **Materials Manager** Coordinates activities to provide material needed to meet the master production schedule. Monitors material needs and order status, anticipates problem situations, and coordinates activities to solve problems in a timely fashion. Serves in a supervisory position and does not perform daily transactions related to materials management. **Production Manager** Coordinates shop floor activities required to meet the master production schedule. Monitors work in process, order priorities and capacity at each work center. Anticipates problem situations and coordinates activities to solve problems in a timely fashion. Serves in a supervisory position and does not perform daily transactions related to shop activity. **Master Production Schedulers** Develop and maintain the master production schedule for assigned end items. Receive and enter customer orders, adjust master schedule quantities as needed to meet demand and finished goods inventory targets, receive completed finished goods into inventory, process sales shipments to customers, and track customer service levels. Responsible for all decision making and transaction processing related to the master production schedule. **Inventory Planners** Initiate and monitor actions to obtain needed assemblies, components, and materials to meet the master production schedule. Responsible for all decision making and transaction processing to order, schedule/reschedule dues dates, and receive assigned inventory items. **Work Center Managers** Responsible for managing assigned work centers to meet manufacturing plans. Manage work center capacity, establish and execute daily job schedules, and coordinate operations with related work centers. Responsible for decision making, transaction processing, and reporting requirements in the assigned work area.

very near term must be justified in writing since these changes affect material and capacity plans that have already been implemented. The master production schedulers must be able to explain the impact of changes they propose. These and other policies force students to use the information system to plan ahead, anticipate problems, and communicate with each other.

During the second half of the semester, all decision making and transaction processing are conducted by the student team. Factory conditions resulting from these decisions provide the data used in the classroom activities during the second half of the semester as described in the previous section.

Weekly factory performance metrics are computed and posted in a public place so that factory performance can be viewed by all students and observers, whether or not they are direct participants in the project. These metrics include total sales shipments, customer service level (percent of customer orders shipped on time and complete), overtime hours worked in the facility, work in process inventory, finished goods inventory, and total inventory. Factory performance for several previous semesters is available so that students can see their performance relative to previous classes and to periods in which the instructor and graduate assistant were managing the factory. The formal performance assessment of each student at the conclusion of the project includes both individual performance and factory performance.

Project results continually show that successful management of the virtual factory requires more than simple content knowledge of operations management tools and techniques presented in textbooks. The dynamics of the virtual enterprise require students to deal with uncertainties and ambiguities, to make decisions in real time, and to coordinate their activities and understand how their roles and operations management decisions are interrelated. Successful completion of the project requires not only technical competence in operations management and expertise in using enterprise software, but also teamwork, communication, and leadership.

Overall Effectiveness of the Approach

There are a number of indicators that the use of enterprise software to teach OM as described above is effective. Three indicators discussed below include the success of former students, continuing support from the business community, and external recognition from academic and professional societies.

Student participants in the project have been well received in the job market, and many report that their hands-on experience with enterprise systems was a major factor in attracting interest from potential employers. Many former students hold positions of major responsibility in manufacturing firms, and have supported the program through donations and testimonials.

In the twenty years that the project has been under development, over $1.5 million in hardware, software, and support services has been donated by industrial partners and supporters. The project began in the mid 1980s with assistance from IBM, who provided MAPICS II software running on an IBM System 36 minicomputer. The project evolved with later versions of MAPICS designed for the AS400 series of IBM computers. Various software packages such as JOBSCOPE and FACTOR have been donated and utilized in the manufacturing simulation as the project evolved. Recent support from Microsoft Business Solutions has facilitated more of a total enterprise approach that integrates the virtual factory with accounting courses at Clemson, as described in the "Future Trends" section below. This continuing and evolving industrial support is an indicator of the relevance of the approach, and has enabled the project to stay on the cutting edge of information technology.

The project has also received significant external recognition through formal awards

from academic and professional societies and business organizations, including the Decision Sciences Institute, the Academy of Business, and Microsoft Business Solutions, as well as special recognition from the Carnegie Foundation for the Advancement of Teaching and the Council for the Advancement and Support of Education (CASE). In addition, instructors in the project have won numerous teaching awards from campus student groups.

FUTURE TRENDS

Integration of topics in the business curriculum is likely to be more important in the future as firms continue to rely on enterprise systems to link activities and processes internally and with external suppliers and customers. Enterprise systems are a potentially important resource for collegiate schools of business because they facilitate meaningful integration in the curriculum. As the operations management project described in this chapter continues to evolve, efforts have been undertaken to utilize the virtual manufacturing enterprise in other business courses as well.

The first step in this process involves various accounting courses at Clemson University. Daily operational decision making and transaction processing in the virtual manufacturing enterprise generate accounting data that are examined and processed in a number of different accounting courses. Introductory accounting courses use the ongoing live model to illustrate business events that are discussed in accounting texts, while intermediate accounting students work on financial problems that occur in the virtual enterprise and develop accounting adjustments as needed. This provides opportunities for accounting instructors to address issues involving asset purchases, accounts receivable write-offs, mortgage loan amortizations, and other problems, all in a dynamic environment created by the virtual manufacturing enterprise.

A logical extension of these activities is the development of courses that combine teams of management, accounting, and other business students working together to manage the virtual enterprise. This type of across-course integration could be the wave of the future in business schools, and it will likely blur the boundaries between traditional courses in functional areas of business such as operations, accounting, finance, marketing, and human resources. Such activities require careful planning to ensure that the body of knowledge in each functional area is clearly identified, the learning objectives are clear, and student participants have sufficient content knowledge to engage in their assigned role in the virtual enterprise and benefit from the experience. The enterprise system technology to accomplish this is readily available — the challenge is for business schools to develop meaningful ways to employ it.

CONCLUSION

An approach to teaching operations management has been presented in which enterprise software brings to life key issues in managing the production operations of a business. A key feature of the approach is the realism created by using enterprise software to operate the virtual factory in an ongoing, real-time mode in which various operations management issues related to customer service, inventory, scheduling, and capacity occur simultaneously. The approach provides a way to enhance understanding of operations management topics, as well as a vehicle for students to better understand the integration of operations management with other functional areas of business such as accounting, finance, and marketing. An important by-product of the approach is the hands-on experience students receive with enterprise software that is widely used in the business world.

REFERENCES

Cannon, D. M., Klein, H. A., Koste, L. L., & Magal, S. R. (2004). Curriculum integration using enterprise resource planning: An integrative case approach. *Journal of Education for Business, 80*(2), 93-101.

Hitt, L. M., Wu, D. J., & Zhou, X. (2002). Investment in enterprise resource planning: Business impact and productivity measures. *Journal of Management Information Systems, 19*(1), 71-98.

LaForge, R. L., & Busing, M. E. (1998). The use of industrial software to create experiential learning activities in operations management courses. *Production and Operations Management, 7*(3), 325-334.

Mabert, V. A., Soni, A., & Venkataramanan, M. A. (2000). Enterprise resource planning survey of U.S. manufacturing firms. *Production & Inventory Management Journal, 41*(2), 52-58.

Manetti, J. (2001). How technology is transforming manufacturing. *Production & Inventory Management Journal, 42*(1), 54-64.

Rondeau, P. J., & Litteral, L. A. (2001). Evolution of manufacturing planning and control systems: from reorder point to enterprise resources planning. *Production & Inventory Management Journal, 42*(2), 1-7.

This work was previously published in Enterprise Systems Education in the 21st Century, edited by A. Targowski, pp. 138-151, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Section V
Organizational and Social Implications

This section includes a wide range of research pertaining to the social and organizational impact of software applications. Chapters introducing this section analyze open source software communities, while later selections discuss organizational modeling and the analysis of user interfaces. The inquiries and methods presented in this section offer insight into the implications of software applications at both a personal and organizational level, while also emphasizing potential areas of study within the discipline.

Chapter 5.1
Open Source Software Communities

Kevin Carillo
Concordia University, Canada

Chitu Okoli
Concordia University, Canada

INTRODUCTION

Open source software (OSS) development has continued to appear as a puzzling and enigmatic phenomenon and has drawn increasing attention as its importance has grown. Relying upon an alternative way to develop and to distribute software, open source communities have been able to challenge and often outperform proprietary software by enabling better reliability, lower costs, shorter development times, and a higher quality of code (Raymond, 2004). Behind the software is a mass of people working together in loose coordination, even portrayed as a rowdy marketplace (Raymond, 2001, p. 1):

No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches … out of which a coherent and stable system seemingly emerges only by a succession of miracles.

More precisely, the people behind open source projects have been defined as: "Internet-based communities of software developers who voluntarily collaborate in order to develop software that they or their organizations need" (von Krogh, 2003, p. 14). In contrast to the sacred cathedral-like software development model that gave birth to most commercial and proprietary systems, such bazaar-like communities seem to have based their success on a pseudo-anarchic type of collaboration and developers' interaction (Raymond, 2001). However, in spite of the apparent disorganization of these bazaars, a closer look distinguishes common values and norms that rule them, specific roles that can be identified, similar motives shared by people, and practices that follow patterns. This article highlights key aspects of what forms the communities that support these projects.

Definition of Open Source Software

The basic definition of OSS as expressed by the Open Source Initiative (www.opensource.org) goes beyond the notion of free code. It encompasses broader issues such as distribution and licensing that stipulate free exchange and modification rights of source code (OSI, 1997):

- Free redistribution of source code
- Free redistribution of compiled (binary) programs
- Derived works must be permitted
- Integrity of the author's source code
- No discrimination against persons or groups
- No discrimination against fields of endeavor (e.g., commercial and military uses must be permissible)
- Mandatory distribution of open source license
- License must not be specific to a product
- License must not restrict other software's licenses
- License must not restrict redistribution to a particular delivery technology

A BRIEF HISTORY OF THE OPEN SOURCE PHENOMENON

During the 1960s and 1970s, scientists and engineers in academic and corporate laboratories freely shared, exchanged and modified the software they produced. However, by the early 1980s, software was increasingly shifting from its original shared nature to becoming increasingly commercialized, with licenses that forbade the free sharing of source code. In 1983, Richard Stallman left MIT to found the Free Software Foundation (FSF) with the principle aim of defining and diffusing legal mechanisms and conceptual principles of "free software" (Hars & Ou, 2001; West & Dedrick, 2001). "Free" here refers to freedom, the liberty to do whatever desired with the software. Hence, free software in the open source sense is distinct from "freeware", which is software sold at no price. In fact, one of the explicit rights given to users of "free software" is the right to sell it commercially. It is noteworthy that most OSS is freeware (provided at no charge), but most freeware is not open source (the source code is not provided, and users are forbidden from modifying the program code even if they could).

Stallman's publication of the GNU Manifesto (1985) allowed him to communicate his ideological insights about the nature of software (von Krogh, 2003), and he convinced developers to join him in the GNU Project, whose primary goal was—and still is—the creation of a Unix-like free operating system. ("GNU" is a recursive acronym meaning, "Gnu's Not Unix".) Accompanied by the continuous improvements of networking capabilities and of the Internet, this major step signaled the beginnings of open source practices organized through the formation of virtual communities. In 1989, the Free Software Foundation released the GNU General Public License (GPL) in order to ensure the preservation of certain freedoms in the copies and derivative works of a piece of software. The GPL assures these freedoms via the copyleft mechanism, which permits free copying, modification, and distribution of software, with the condition that any distributed derivative works explicitly accord others the same rights.

In 1991, Linus Torvalds, a 21-year-old Finnish programmer, created Linux, a kernel for a Unix-based operating system that uses the operating system tools created by the GNU Project. Since then, this multi-user/multitasking platform has met tremendous success and is known for being powerful, fast, efficient, stable, reliable, and scalable (Edwards, 1998). In 1999, a survey estimated that the GNU/Linux operating system (popular known simply as "Linux") was the operating system of more than 30% of Internet server sites. A recent release of the kernel (Linux 2.2.10) credits 190 key developers, though the total number of

contributors was estimated to be around 1,200 (Dempsey, Weiss, Jones, & Greenberg, 2002; Stewart & Ammeter, 2002).

In 1998, Bruce Perens and Eric Raymond of the Open Source Initiative pointed out that the mistrust of many traditional firms towards the GPL was in Stallman's term "free", which is not very a attractive idea to the business world (von Krogh, 2003)—conventional wisdom says that you get what you pay for. Thus, they tried to refocus what they rebranded as the "open source" software movement by primarily focusing on the economic and engineering superiority of the "open source" approach to software development, in contrast to FSF's more philosophically antagonistic approach towards "source-hoarding". In response to their positive marketing, the term "open source" has become the preferred terminology for this approach. Other major open source projects have followed Linux's success, including the Apache Web server (started in 1995) and the Mozilla Internet suite project (started in 1999 when Netscape released its Communicator suite as open source; Mozilla released the Firefox Web browser and Thunderbird e-mail client at the end of 2004).

PROFILE OF OPEN SOURCE COMMUNITIES

Lee and Cole (2003, p. 51) defined a virtual community as "a cyberspace supported by computer-based information technology, centered upon communication and interaction of participants to generate member-driven contents, resulting in a relationship being built up." OSS development communities have been defined as groups of loosely connected programmers, who use the Internet as a medium for collaboratively developing, improving, and disseminating software (O'Reilly, 1999), and are recognized as a particular genre of virtual community (Ljungberg, 2000) and as virtual organizations (Crowston & Scozzi, 2002), which are characterized by: group members with a shared interest or goal; geographical distribution; and use of information and communication technology to communicate and manage interdependencies (Ahuja & Carley, 1998). They have been classified as communities of transaction (Hagel & Armstrong, 1997) and as "task- and goal-oriented" communities; that is, "communities striving to achieve a common goal by way of cooperation" (Stanoevska-Slabeva & Schmid, 2001).

Ideology and Values

One of the main emphases in OSS communities is the social interaction among participants through electronic communication. Rheingold (1993, p. 5) defines virtual communities as "social aggregations that emerge from the Net when enough people carry on public discussions long enough, with sufficient human feeling, to form webs of personal relationships in cyberspace". This definition highlights the creation of relationships among people. As a consequence, the participants of an OSS community share a set of common values, and they adhere to the same ideology: the OSS culture.

The most commonly accepted view of OSS communities' culture has been proposed by Raymond (2001) in his landmark paper, "The Cathedral and the Bazaar", where he characterizes it as a "gift culture", as opposed to an "exchange culture". Exchange cultures are based on scarcity whereas gift cultures rely on abundance. Raymond argues that in gift cultures, social status is determined "not by what you control, but by what you give away". Bergquist and Ljungberg (2001) have empirically demonstrated that such giving values and behaviors enable social relationships to be created and maintained in virtual environments. Assuming any ideology is a common set of shared norms, values and beliefs among the members of a community, Raymond's view of OS communities can be characterized by the features listed in Table 1 (Stewart & Gosain, 2001).

Table 1. Ideologies and values of open sources software communities (Stewart &Gosain, 2001)

Norms	▪ Taboo against forking projects ▪ Distributing changes without cooperation of moderators frowned upon ▪ Removing a person's name from project history, credits or maintainers list is not done without explicit consent
Values	▪ The best craftsmanship wins ▪ All information should be free ▪ You don't become a hacker by calling yourself a hacker—you become a hacker when other hackers call you a hacker ▪ Non-trivial extensions of function are better than low-level patches and debugging ▪ Work that makes it into a big distribution is better than work that does not
Beliefs	▪ With enough eyeballs all bugs are shallow ▪ Practice is better than theory
Ideologues	▪ Stallman ▪ Raymond
Language, Symbols	▪ "Distros" ▪ "Suits" ▪ Free Software Foundation ▪ Copyleft ▪ Open source licenses
Narratives	▪ The Halloween Papers ▪ The Cathedral and the Bazaar ▪ Slashdot, Freshmeat, Sourceforge

Furthermore, another dimension that may be added to the OSS ideology is the hostility towards commercial software. The intensity varies from one community to another one, from a fundamentally hostile view of commercialized software (almost universally expressed towards Microsoft Corporation, currently the largest commercial software enterprise) to a moderate view that accepts the symbiotic nature and coexistence of open source and proprietary software (Ousterhout, 1999).

An organizational culture approach elicits further insights about the rules governing OS communities. Schein's (1984) framework posits that the culture of organizations can be understood by examining their artifacts, values and core assumptions. Sharma et al. (2002) have applied this

framework to OSS, with the results summarized in Table 2.

Researchers in the area of virtual communities have pointed out the overall importance of trust. For instance, Carver (1999, 114) affirms, "People are drawn to virtual communities because they provide an engaging environment in which other people … create an atmosphere of trust and real insight". Trust is considered as a core assumption of the OSS ideology, considering that core developers need to work closely with one another as they implement different but interrelated code segments (Sharma et al., 2002). As a result, it was found that trust among group members in OSS project groups plays an essential role in facilitating development success (Stewart & Gosain, 2001). Thus, OS development processes have to be managed through trust-based relationships in order to ensure OSS project success and quality.

Participants' Profiles and Roles

It has been common to characterize OSS community members as anarchistic crackers operating on the fringes of society, a stereotype that has been soundly dispelled by recent surveys (Fitzgerald, 2004). Several surveys have identified consistent traits among OSS community members. Over 95% of members are male. The average age is around 30, with a majority of people between 20 and 30. 20% of members are students and over 50% of people have IT jobs. Most developers have a bachelor's (or equivalent) degree, many have a master degree, and under 10% have a PhD degree (Ghosh, Glott, Krieger, & Robles, 2002; Hars & Ou, 2001; Lakhani, Wolf, Bates, & DiBona, 2002).

OS developers tend to participate in a limited number of projects—an average of just two or three each (Dempsey et al., 2002). Developers modify the program source code and make important decisions concerning any future development. Non-developers are valuable in reporting bugs, and in suggesting feature enhancements. Although it is common to think of OSS participants as willing, independent enthusiasts, on deeper analysis it is evident there is a certain kind of hierarchy inherent in any project (Glass, 2003).

CONCLUSION

OSS communities are an important type of virtual community today, where members convene online with the common goal of producing software that is valuable both to developers and for the general public. While most OSS communities do not explicitly restrict membership (except perhaps to

Table 2. OSS organizational culture (Sharma et al., 2002)

Artifacts	▪ Electronic communication
Values	▪ Altruism ▪ Reciprocity ▪ Gift Giving ▪ Reputation ▪ Fairness ▪ Shared risks and ownership
Core Assumptions	▪ Trust ▪ Loyalty

the inner core of developers), they are particular in their membership criteria, which by its nature requires considerable skills in computer programming and software development. However, as we have discussed, there is also plenty of room for non-developer user members who contribute ideas and directions valuable to projects, such as identifying bugs and suggesting new features. As the open source movement continues to grow both as a software development methodology and as a philosophical/social/political approach to intellectual property, OSS communities will have an increasingly important role in the software industry.

REFERENCES

Ahuja, M.K., & Carley, K.M. (1998). Network structure in virtual organizations. Journal of Computer-Mediated Communication, 3(4). Retrieved from http://www.ascusc.org/jcmc/vol4/issue4/ahuja.html

Bergquist, M., & Ljungberg, J. (2001). The power of gifts: Organizing social relationships in open source communities. Information Systems Journal, 11(4), 305-320.

Carver, C. (1999). Building a virtual community for a tele-learning environment. IEEE Communications Magazine, 37(3), 114-118.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations: Competency rallying for software development. IEE Proceedings Software, 149(1), 3-17.

Dempsey, B.J., Weiss, D., Jones, P., & Greenberg, J. (2002). Who is an open source software developer? Communications of the ACM, 45(2), pp. 67-72.

Edwards, J. (1998). The changing face of freeware. IEEE Computer, 31(10), 11-13.

Fitzgerald, B. (2004). A critical look at open source. IEEE Computer, 37(7), 92-94.

FSF. (1985, May 17). The GNU Manifesto. Retrieved December 2004, from http://www.gnu.org/gnu/manifesto.html

Ghosh, R.A., Glott, R., Krieger, B., & Robles, G. (2002). Free/libre and open source software: Survey and study. Brussels: Report of the FLOSS Workshop on Advancing the Research Agenda on Free/Open Source Software, European Commission.

Glass, R.L. (2003). Practical programmer: A sociopolitical look at open source. Communications of the ACM, 46(11), pp. 21-23.

Hagel, J., & Armstrong, A. (1997). Net gain: Expanding markets through virtual communities. Cambridge, MA: Harvard Business School Press.

Hars, A., & Ou, S. (2001, January 3-6). Working for free? Motivations of participating in open source projects. Paper presented at the 34th Hawaii International Conference on System Sciences, Island of Maui, Hawaii.

Lakhani, K.R., Wolf, B., Bates, J., & DiBona, C. (2002). The Boston Consulting Group hacker survey. Boston Consulting Group and Open Source Developers Network.

Lee, F.S.L., Vogel, D., & Limaya, M. (2003). Virtual community informatics: A review and research agenda. JITTA: Journal of Information Technology Theory and Application, 5(1), pp. 47-61.

Ljungberg, J. (2000). Open source movements as a model for organising. European Journal of Information Systems, 9(4), 208.

O'Reilly, T. (1999). Lessons from open-source software development. Communications of the ACM, 42(4), pp. 33-37.

OSI.The open source definition. Retrieved December, 2004, Retrieved from http://www.opensource.org/docs/definition.php

Ousterhout, J. (1999). Free software needs profit. Communications of the ACM, 42(4), pp. 44-45.

Raymond, E.S. (2001). The cathedral and the bazaa: Musings on Linux and open source by an accidental revolutionary. Sebstapol, CA: O.Reilly.

Raymond, E.S. (2004). Up from alchemy [open source development]. IEEE Software, 21(1), 88-90.

Rheingold, H. (1993). The virtual community: Homesteading on the electronic frontier. New York: HarperPerennial.

Schein, E.H. (1984). Coming to a new awareness of organizational culture. Sloan Management Review, 25(2), 3-16.

Sharma, S., Sugumaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-open source software communities. Information Systems Journal, 12(1), 7-25.

Stanoevska-Slabeva, K., & Schmid, B.F. (2001, January 3-6). A typology of online communities and community supporting platforms. Paper presented at the Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34),January 3-6, 2001 (Vol. 7, p. 7010).

Stewart, K.J., & Ammeter, T. (2002, December 15-18). An exploratory study of factors influencing the level of vitality and popularity of open source projects. Paper presented at the 23rd International Conference on Information Systems, Barcelona, Spain.

Stewart, K.J., & Gosain, S. (2001, December 16-19). An exploratory study of ideology and trust in open source development groups. Paper presented at the 22nd International Conference on Information Systems, New Orleans, Louisiana.

von Krogh, G. (2003). Open-source software development. MIT Sloan Management Review, 44(3), 14.

West, J., & Dedrick, J. (2001, January 3-6). Proprietary vs. open standards in the network era: An examination of the Linux phenomenon. Paper presented at the 34th Hawaii International Conference on System Sciences, Island of Maui, Hawaii.

KEY TERMS

Free Software: An earlier name for open source software, emphasizing the liberties given to end users and developers of derivative works. There is no requirement that the software be distributed at no charge; thus, distinct from freeware.

Freeware: Software provided at no charge to the user. Might be open source or proprietary; that is, the developer only permits redistribution and use, with no modifications permitted. In fact, most open source software is freeware, but most freeware is not open source.

GNU General Public License: The first and still the most radical open source software license, created for the GNU Project. Requires that all derivative works be equally free (in the open source sense); that is, all derivative works must provide the full source code and must permit free use, modification, and redistribution.

GNU Project: (Stands for, "Gnu's Not Unix") Established by Richard Stallman in 1983 under the auspices of the Free Software Foundation. Its goal was, and still is, to create an open source Unix-based operating system. This goal was realized in 1991 by Linus Torvald's creation of Linux.

Linux: A Unix-based open source operating system designed for Intel-based microcomputers. The kernel was created in 1991 by Linus Torvalds, and it was added on to the GNU Project to form what is properly called the GNU/Linux operating system.

Mozilla Project: A project formed in 1998 when Netscape released its Internet tools suite for open source development. Released its flagship Firefox Web browser and Thunderbird e-mail client at the end of 2004. The Sunbird calendar component is currently under development.

Open Source Software: Software whose source code is liberally made available for use, modification, creation of derivative works, and redistribution.

The Cathedral and the Bazaar: Paper by Eric Raymond (most recent version in 2001) that contrasts the "Cathedral" software development approach of a closed hierarchy (e.g. for proprietary software and most open source software such as the earlier GNU Project) with the "Bazaar" approach of loose collaboration with light centralized moderation (as was used for the Linux and Fetchmail open source projects).

This work was previously published in Encyclopedia of Virtual Communities and Technologies, edited by S. Dasgupta, pp. 363-367, copyright 2006 by Information Science Reference (an imprint of IGI Global).

Chapter 5.2
Beyond Development:
A Research Agenda for Investigating Open Source Software User Communities

Leigh Jin
San Francisco State University, USA

Daniel Robey
Georgia State University, USA

Marie-Claude Boudreau
University of Georgia, USA

ABSTRACT

Open source software has rapidly become a popular area of study within the information systems research community. Most of the research conducted so far has focused on the phenomenon of open source software development, rather than use. We argue for the importance of studying open source software use and propose a framework to guide research in this area. The framework describes four main areas of investigation: the creation of OSS user communities, their characteristics, their contributions and how they change. For each area of the framework, we suggest several research questions that deserve attention.

INTRODUCTION

In recent years, the open source software (OSS) development movement has captured the attention of both information systems practitioners and researchers. The "open community model" is one that involves the development and support of software by volunteers with no or limited commercial interest. This model differs from proprietary software development, and with other open source business models such as corporate distribution, sponsored open source and second-generation open source (Watson, Boudreau, York, Greiner, & Wynn, in press). The open community model is appealing to many because of its application of community principles of governance over commercial activities (Markus, Manville, & Agres, 2000; von Hippel & von Krogh, 2003). By describing open source as a "movement," we

reflect the broader excitement about the implications of community governance processes in a knowledge economy (Adler, 2001).

Open source has rapidly become a popular area of study within the information systems (IS) research community, as evidenced by the appearance of special tracks for OSS within conferences and special issues of journals. For example, the Americas Conference on Information Systems sponsored an "Open Source Adoption and Use" minitrack, and the Hawaii International Conference on System Sciences offered a minitrack on "Open Source Software Development." On the journal side, *Management Science* invited submissions to a special issue on "Open Source Software" in 2004. Also, *Journal of Database Management* announced a special issue on "Open Source Software."

Although these calls for OSS research do not limit contributions, the vast majority of the research conducted so far has focused on OSS development rather than use (Fitzgerald & Kenny, 2003). The interest in open community development reflects a desire to explain the counterintuitive practice of treating commercially valuable products as public goods rather than proprietary products for sale. Likewise, the development and maintenance of complex software products by communities of expert volunteers has piqued interest into the incentives for developers. As a consequence of the primary focus on OSS development, little research has yet been conducted on OSS use, especially by non-technical users.

The neglect of OSS use may be attributed to two false assumptions about OSS projects. First, it is known that people often become OSS developers because they intend to use the product being developed. To echo Raymond's (2001) frequently quoted expression, OSS developers are users with an "itch to scratch," so they are willing to devote time and expertise to develop software solutions to their own problems as users. Thus, it is commonly assumed that there is no distinction between OSS developers and users (Feller & Fitzgerald, 2000). Following this line of argument, we might conclude that no special research agenda for OSS use is needed because OSS use is redundant with OSS development.

However, this argument and its underlying assumption can be challenged on the grounds that OSS use by technically experienced developers differs from OSS use by technically naïve users. Verma, Jin, and Negi (2005) argued that technical developers and non-technical users have very different interpretations of OSS's ease of use. Non-technical users may experience difficulty in using OSS products because OSS developers are motivated to improve functionality rather than usability (Nichols & Twidale, 2003). Thus, even though all OSS developers are likely also to be users, the distinction between developers and non-technical users remains important.

The assumption can also be challenged by statistics showing the rapid rise in the number of OSS users, the vast majority of whom have no interest or capability to contribute to modifications of the source code (Fitzgerald & Kenny, 2003). For widely distributed OSS such as Linux, it makes no sense to assume that more than a small percentage of users could possibly become developers (von Hippel & von Krogh, 2003). Clearly, users vastly outnumber developers in larger OSS projects. As OSS development becomes increasingly targeted toward productivity and entertainment applications, the relative proportion of non-technical users can be expected to increase.

The second assumption dissuading research on OSS use is that the OSS movement is unique solely because of the way software is developed, but that its use is similar to any other type of software. Given the abundance of IS research that is focused on the adoption and use of software applications, one might assume that no special research program is needed for OSS use.

This assumption can be challenged by examining some differences between OSS and proprietary software. Users of OSS are typically confronted by a fundamentally different type of technical

support than that found with proprietary software. Rather than relying on a vendor's customer support, users of OSS generally need to search for community resources for help for installing, learning and using their freely acquired software. OSS users are likely to receive such help through participation in user groups or mailing lists that are supported by volunteers, similar to the communities supporting OSS development (Golden, 2005; Lakhani & von Hippel, 2003; Raymond, 2001). Given these distinctive features of OSS products and support, we do not agree with the assumption that OSS use is the same as proprietary software use.

These arguments justify research into OSS use. In this article, we adopt a community perspective on OSS use, which is explained in the following section. We then present a framework that includes four main areas of investigation: *creation* of OSS user communities, their *characteristics*, their *contributions* and how they *change*. For each element of the framework we pose several research questions.

A COMMUNITY PERSPECTIVE ON OSS USE

The term "community" was introduced into the English language in the 14th century from Latin to refer to a group of people living in a common geographical location. Between the 17th and 19th centuries the meaning of community was extended to describe people who shared common characteristics, interests or identities – even if they were not geographically close (Cole, 2002; Williams, 1973). As the 21st century begins, people have grown more accustomed to participating in virtual communities that are enabled by Internet technology and the World Wide Web (Rheingold, 2000). Virtual communities differ from colocated communities by offering a wider range of options for participation and by allowing community size to grow, unconstrained by physical space. As Cole (2002) noted, virtual communities can be more heterogeneous: "each [community] is unique in the combination of institutional arrangements, educational content, forms of Internet communication, and participant goals that it embodies" (p. xxvi). Moreover, individual members may tailor their virtual communities to satisfy personal preferences (Wellman, 2001).

The power of collaboration within geographically dispersed communities is demonstrated by the making of the first edition of the *Oxford English Dictionary*, one of the earliest examples of an open community project (Watson, Boudreau, Greiner, Wynn, York & Gul, 2005). The dictionary, which took about 70 years to complete, was compiled primarily from definitions submitted by thousands of volunteers fluent in the English language. In his book, *The Professor and the Madman*, Winchester (1998) reported that an insane American prisoner became the most prolific contributor to the original compilation of the *Oxford English Dictionary*. This story illustrates two principles that also sustain modern OSS communities: everyone is welcome to contribute regardless of personal circumstance, and value can be produced through community effort.

OSS development largely depends upon the ability of developers to contribute as members of virtual communities (Markus et al., 2000; Lakhani & von Hippel; 2003; Raymond, 2001). Given that much OSS development transpires in online communities, we expect that OSS use also relies on virtual communities for software acquisition, implementation, maintenance and support. For example, Lakhani and von Hippel (2003) suggested that successful OSS projects were capable of delivering high quality field support – mundane but necessary tasks – to users through voluntary effort. Field support primarily involves experienced users answering questions posted by novice users through an archived mailing list. Indeed, a highly organized system of OSS user groups has grown around major OSS products. Taking Linux user groups (LUGs) as an example,

in July 2005 there were 846 registered LUGs in 108 countries, including 299 in the United States (http://www.linux.org/groups).

Despite the importance of electronically mediated interaction within OSS communities, we do not assume that OSS user communities are exclusively virtual. Indeed, one of the naïve assertions about OSS development is that software can be developed by a community of complete strangers who interact only through electronic media. To the contrary, experienced OSS participants have opportunities to attend conferences and regular meetings held in physical places. For example, the O'Reilly Open Source Convention and LinuxWorld Conferences are popular venues for OSS developers to meet and exchange ideas. Research on Linux user groups (Jin, Robey, & Boudreau, 2006) reveals that the Silicon Valley LUG holds face-to-face meetings at least monthly and organizes InstallFests, where new users can bring in their computers and allow experienced volunteers to install Linux, diagnose problems and repair configurations. The LUG of Davis, California, sponsors a Linux Emergency Relief Team, staffed with experts who may even travel to users' homes to service their Linux systems (Jin et al., 2006). According to Moen (2003), LUGs are vital to the Linux movement, taking on many of the same roles that a regional office does for a large organization:

LUGs' role in Linux advocacy cannot be overestimated, especially since wide-scale commercial acceptance of Linux is only newly underway. While it is certainly beneficial to the Linux movement each and every time a computer journalist writes a positive review of Linux, it is also beneficial every time satisfied Linux users brief their friends, colleagues, employees, or employers.

The following research agenda focuses on questions about the creation, characteristics, contributions and change in OSS user communities. There is no direct empirical rationale for dividing the agenda into these four categories because research on OSS user communities is just beginning. Rather, the research agenda reflects a progression through different phases of an OSS community over its life cycle. Creation is the beginning, characteristics describe the makeup of the community, contribution describes the major functions performed, and change describes later growth. The four phases are associated with different kinds of issues, which we offer as researchable topics. Because the life of a community is not biologically limited, our agenda shows that communities may repeat the cycle as they change.

The agenda is proposed at a high level due to the novelty of the phenomenon and the paucity of existing research efforts. By restricting our attention to a community perspective, we purposefully omit consideration of individual and organizational influences on OSS use. However, we believe that a community perspective on OSS research is valuable because it has played such a prominent role in research on OSS development.

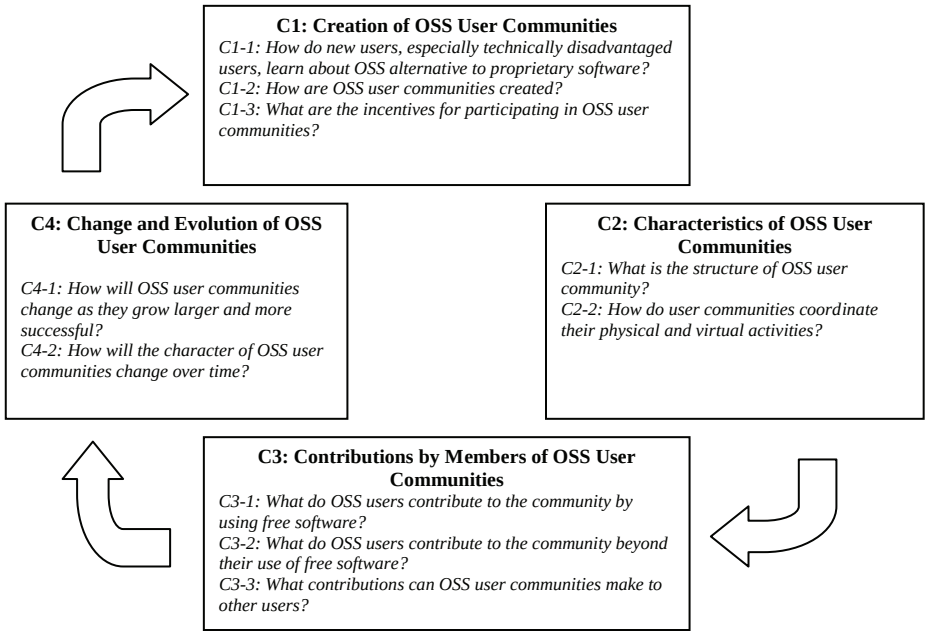
A RESEARCH AGENDA

Figure 1 identifies the four main areas where research into OSS user communities should be undertaken: creation (C1), characteristics (C2), contributions (C3) and change (C4). As shown, these areas are related sequentially, beginning with creation. It is also shown that change not only completes the cycle but potentially begins a new cycle.

C1: Creation of OSS User Communities

The creation of an OSS user community requires potential members to learn about OSS as an alternative to proprietary software and in-house development, to engage in community activities and to respond to incentives for starting or joining

Figure 1. Research areas



a user community. The research questions below address these three important issues.

C1-1: How do new users, especially technically disadvantaged users, learn about OSS alternatives to proprietary software?

This issue is interesting because, compared to proprietary software, open-community software projects lack specialized teams to market the product and to promote it through mass media. Although the notion of gift culture is well established within the OSS development community, users are likely to be suspicious of the opportunity to receive software as a free gift. Although OSS has made news headlines and gained immense exposure on the Internet, a nontechnical user would still be expected to have difficulty installing and using a new operating system such as Linux. Indeed, many new users are introduced to OSS through personal friends, whom they trust and rely upon for support when they encounter problems. The role of the broader OSS user community in introducing new users to OSS products stands as an important research question.

C1-2: How are OSS user communities created?

Creating and sustaining a software user community requires resources such as meeting places, newsletters, and Web hosting services. Traditional software user groups are often sponsored by vendors, who provide considerable benefits to both users and themselves (Buckner, 1996). On the one hand, users may benefit from discounted prices negotiated with the vendors and the chance to suggest improvements in the software's next release. On the other hand, vendors receive a low cost marketing opportunity and obtain feedback on the usability of their products.

Because software vendors are rarely involved in open-community OSS user groups, their creation depends on a group of enthusiasts sharing the same passion for the software. These enthusiasts

may be among the original developers, motivated to increase the software's user base. Alternatively, these enthusiasts may be pure users who believe in the principles behind OSS, therefore promoting its use and maximizing the benefits of their own use. Large companies may also be the principal instigators of OSS user groups. If an OSS product is used extensively within a company, creation of a user group for that product provides free training opportunities for the company's employees. Large technology companies with a stake in an OOS product may also support OSS user groups indirectly by providing meeting space or donating Web hosting services. Companies providing such support may enhance their reputations within the OSS community, which in turn may improve their opportunities to recruit talented new members.

Although we acknowledge that OSS user groups are important components of OSS communities, the creation of other community components should not be overlooked. Special interest groups and forums may be established by universities, government agencies and other organizations. Although their activities may be less visible to the public, they may generate additional resources, such as training provided by the sponsoring organizations.

C1-3: What are the incentives for participating in OSS user communities?

Because users may obtain OSS freely, with no obligation to contribute to development, their use of the software is likely to be based primarily on cost and quality considerations (Fitzgerald & Kenney, 2003). The incentives for community participation, however, differ from the incentives for using OSS (Wang, 2005). For developers who incur substantial private costs by investing their own resources into development, incentives include not only the ability to use the software but also benefits related to reputation and learning. It is conceivable that users may also obtain such benefits, gaining reputations as skilled implementers who are helpful to novice users.

It has been argued that the true benefit for using open source software goes beyond its low initial cost and involves a user's long-term control over information technology. By contrast, users of proprietary software become dependent on software that remains inside the vendor's black box. With no ability to change the source code, users of proprietary software lose control and subject themselves to a monopoly relationship with their vendor (Moen, 2000). OSS may provide an incentive by restoring the user's control.

It is also likely that OSS user groups offer more value than user groups organized by proprietary software vendors. Vendor-sponsored user groups often charge membership fees and use their meetings to promote new products. These commercial interests may interfere with the activities of OSS user groups. For the more technically inclined, OSS user groups offer solutions in the form of code modifications that address specific problems of individual users. By contrast, vendors are more likely to avoid solutions involving code modification and focus on software configuration or settings, which may or may not solve the user's problems. As stated by Moen (2003), OSS user groups are ready and willing to modify source code on the spot:

Traditional groups must closely monitor what software users redistribute at meetings. While illegal copying of restricted proprietary software certainly occurred, it was officially discouraged – for good reason. At LUG meetings, however, that entire mindset simply does not apply: Far from being forbidden, unrestricted copying of Linux should be among a LUG's primary goals.

Outside geopolitical forces could also provide powerful incentives for OSS user community participation. When a U.S. company refused to develop a Portuguese language version of its software for a company studying rain forest

biodiversity in Brazil, the Brazilians turned to an inexpensive OSS solution. The OSS software not only delivered the functionality that the company needed, but also provided all prompts in the Portuguese language (Hall, 2002). Consequently, the Brazilian government became a strong advocate of the OSS movement, as explained by a Brazilian representative at the LinuxWorld 2004 conference:

The question is . . . how the universities can have better tools to teach their students, how government can save millions or billions of dollars each year and still be able to use all kinds of technology. Our government decided that OSS is a good idea; it is something to be supported. In fact, we are the biggest user of OSS in Latin America. We have something around now, eight or ten million users of OSS.

Understanding the creation of OSS user communities requires attention to many issues pertaining to users learning about OSS products; the creation of specific mechanisms, such as user groups, to support user activities, and incentives for wide participation. Because these issues are raised at the community level of analysis, answers cannot be obtained by looking at individual decisions to join and contribute to user communities. Studies of OSS user communities may draw insights from OSS development communities, but it should not be assumed that use is the same as development. For these reasons, our research agenda focuses specifically on the creation of OSS user communities.

C2: Characteristics of OSS User Communities

Like all communities, OSS user communities are likely to be differentiated by the roles that different members play. It is also important to recognize the relationships between user and developer communities. Indeed, it may be desirable to consider users and developers as subcommunities within an OSS project. The questions below address issues related to the characteristics of OSS user communities.

C2-1: What is the structure of an OSS user community?

An OSS community is built around a specific OSS project, with shared interests of improving and using the software. As more people become interested in using the software, the community grows and differentiates into various roles. For example, Ye, Kishida, Nakakoji, and Yamamoto (2002) defined eight roles in OSS communities: (1) project leader, (2) core members, (3) active developers, (4) peripheral developers, (5) bug fixers, (6) bug reporters, (7) source code readers and (8) passive users.

Although helpful in defining the characteristics of an OSS community, the roles defined by Ye et al. (2002) appear most suitable as a description of an OSS development-oriented community (Jin, Verma, & Negi, 2005). Of the eight roles identified, only the last three include users, while the first seven describe developers. According to Jin et al. (2005), an OSS development-oriented community is "exclusively dedicated to develop, support, and maintain a single or multiple open source project(s)" (p. 16522). An OSS user-oriented community, by contrast, is typically created by users for users to "promote the use of OSS products" (Jin et al., 2005, p. 16523). The primary activities of OSS user-oriented communities are attracting new members to adopt OSS products and educating existing members about the best use practices for OSS products. Linux user groups are excellent examples of OSS user-oriented communities. Other examples include the perl mongers community, BSD User Groups, and MYSQL User Groups. While developers could be involved in an OSS user-oriented community, the roles that its members play would differ from those played in an OSS development-oriented community.

Based on our participation in various LUGs, the following roles of OSS user oriented community are evident: (1) founder/officer, (2) meeting and facilities coordinator, (3) public relations officer, (4) Installfest coordinator, (5) mailing list/Web site administrator, (6) face-to-face meeting attendees and (7) mailing list subscribers.

Depending on the size of a particular user-oriented community, some of these roles could be combined and performed by one person. An important issue deserving research attention is the way members assume these roles and the relationships among the various roles. Such research could contribute to more effective designs for community structure.

C2-2: How do user communities coordinate their physical and virtual activities?

Given that OSS development communities operate both physically and virtually, it is an important question to understand how they use these different arenas of community life. Theories of virtual organizing have pointed to the possibility for virtual and physical activities to reinforce, complement, compensate and produce synergies with each other (Robey, Schwaig & Jin, 2003). At one level, OSS virtual activities allow developers to access and modify source code, access necessary archives, and interact through mailing lists, Internet Relay Chat channels, and Web logs. At another level, OSS physical activities are important for forging social ties and clarifying communications (Jin et al., 2006). Moen (2003) emphasized that LUGs' socializing function was effective for members to become acculturated into the Linux user community:

By "socializing", here I mean primarily sharing experiences, forming friendships, and mutually-shared admiration and respect. In other words, acculturation turns you from "one of them" to "one of us." ... LUGs are often much more efficient at this task than are mailing lists or newsgroups, precisely because of the former's greater interactivity and personal focus.

Besides the socializing benefit from face-to-face meetings, physical activities like InstallFests can be effective for advanced users to tailor their help to new users' particular needs. Because new users bring their computers to InstallFests, OSS experts can demonstrate software installation, diagnose problems, and even conduct hands-on training individually with new users.

Understanding the characteristics of OSS user communities requires attention to the roles played by members and the mechanisms for coordinating community action. Again, such matters cannot be undertaken from an individual or even small group perspective. The virtual nature of OSS user communities makes their potential size practically unlimited. Yet, research on the characteristics of OSS communities needs to understand their virtual nature without losing sight of the importance of their face-to-face, physical activities.

C3: Contributions by Members of OSS User Communities

In the OSS development literature, much is made of the voluntary gifts donated by skilled designers for the creation of a public good (Fitzgerald & Kenney, 2003). Indeed, communities are likely to fail if such contributions are not made. The following questions are posed with the same issue in mind for the OSS user community.

C3-1: What do OSS users contribute to the community by using free software?

If contributions are not made by users, OSS users assume the status of free riders who simply take from the community without paying back. Paradoxically, free ridership by OSS users is actively encouraged rather than discouraged. Because the

number of OSS users is a measure of a project's success, users are not pressured to contribute as developers do (von Hippel & von Krogh, 2003). Indeed, their most important contribution may simply be their use of the OSS product.

For example, one of the goals of the Firefox team (a popular OSS Internet browser) is to acquire ten percent of the browser market share. According to one of the team's leaders, "It really doesn't matter how great your technology is. If nobody's using it, then you are not achieving that mission of bringing back choice" (personal communication, 2005). Although most users may never contribute toward product improvement, they are valuable to the success of the Firefox project because their usage adds to Firefox's market share. Indeed, some software projects have found an OSS strategy to be an effective solution for acquiring a customer base before pursuing more commercial objectives. Onetti and Capobianco (2005) found a positive correlation between the number of downloads generated on a Sourceforge OSS project site and the number of new clients interested in purchasing the product's commercial license. As more users download and use an OSS product, the more credible and successful the project becomes, potentially attracting even more new users.

C3-2: What do OSS users contribute to the community beyond their use of free software?

As Raymond (2001) pointed out, some of the most successful OSS projects are created by the most talented software developers. Because the OSS community tends to attract people with extensive technical backgrounds, there is a risk that resulting products would reflect the "geek" culture and be less useful to ordinary users. For example, the user interfaces of OSS tend to be command-line driven, making their installation and configuration technically demanding. It is conceivable that less technical users might contribute to development by making OSS projects easier to install, configure, use and maintain.

In the case of the Firefox browser project, project leaders not only encouraged contributions from nontechnical users but also valued them as vital assets to the project's success. Instead of contributing code, nontechnical users contributed artistic skills to design the logo and images, marketing skills to help spread the word about Firefox, and even monetary contributions to help place an advertisement in *The New York Times* (McHugh, 2005). Whether such participation would be welcomed on other projects remains uncertain, so the issue presents a good research opportunity to study the impact of less technical users on the development process.

C3-3: What contributions can OSS user communities make to other users?

Fitzgerald and Kenney (2003) reported an interesting case of an Irish hospital using OSS software for a number of internal operations. Although the hospital's IT staff had no intention of ever contributing modifications to the software's source code, they had begun to offer the applications, which they had tailored for themselves, to other health care organizations free of charge. In this manifestation of community spirit, one user was giving back to the community of other users. The study suggests that users may add further value by making OSS programs fit specific industry needs. While these contributions may not earn great reputations, they may provide value for the user community. Future research is warranted on the practice of users making vertical applications more useful for other users, in contrast to the traditional focus in OSS development on horizontal infrastructure systems (Fitzgerald & Kenny, 2003).

Studying contributions is important because communities are held together by such contributions. The research questions posed above guide these potential research opportunities.

C4: Change and Evolution of OSS User Communities

It is clear that OSS user communities are new phenomena that have only become significant economically in the last half decade. We expect the nature of OSS communities to change, perhaps rapidly, as software development and use practices evolve. This, in turn, will affect the creation of new communities, as our cyclical representation in Figure 1 suggests. The following questions address the evolution and change of OSS user communities.

C4-1: How will OSS user communities change as they grow larger and more successful?

Although core developers initiate and contribute the major portion of source code (e.g., 80% in the case of the Apache project (Mockus, Fielding, & Herbsleb, 2002)), the largest growth in community size comes from supporting roles like bug fixers, bug reporters and users. Core developers may even recede in importance as a project stabilizes and does not require major revision. Thus, users may assume more prominent stature in mature communities, partly from their endorsement of a particular OSS project.

Communities are sensitive in the longer run to the free ridership phenomenon because attempts to control free ridership involve increased monitoring costs that eventually outweigh the rewards from contributions (von Hippel & von Krogh, 2003). Because free ridership is viewed positively in OSS development communities, it would be useful to study whether growth will strengthen the community or whether growth can ultimately erode feelings of community solidarity. In other words, will the trust that characterizes a community form of governance (Adler, 2001) disappear as community size increases?

One could also argue that size may not be the most important factor promoting community change. The expansion of low cost communication channels allows OSS users to reach out globally and stay connected. Thus, the cost of monitoring members could remain low despite growth. In addition, increasing user community size could distribute monitoring costs across more members. Simultaneously, community trust could be reinforced at more local levels through the events and social activities discussed earlier. These potential effects of changing size could be studied as part of a research program on community change.

C4-2: How will the character of OSS user communities change over time?

As OSS communities grow and prosper, commercial interests may be expected to appear and taint the gift culture characterizing open communities. Firms such as RedHat emerged quickly to create value by improving software distribution and by adding supplementary services such as installation, support and training. JBoss promoted its application server software as "professional open source," charging users for the expertise and services of key developers (Watson et al., 2006). User groups associated with these and other commercial open source companies may come to resemble proprietary software user groups more than the open communities described earlier. Such changes would potentially affect the character of OSS user communities currently operating under volunteer arrangements.

Although user communities affected by commercial open source companies may lose free access to developer expertise, ultimately less experienced users may benefit more from subscribing to paid services at different levels of support. In addition, companies like JBoss have operated community forums in which independent developers can still contribute free advice. In such cases, commercial and noncommercial interests co-exist in the community. One benefit to the commercial interest is that community forums become fertile recruiting grounds for

new developers. As a JBoss representative at the JavaOne 2005 conference explained:

There are individuals on our forums that don't work for us. They answer questions. We actually try to get a ranking system of who answer the most questions, so it gives us an idea of who we can hire for support. So the forums might turn into a recruiting ground for us.

As the interests of commercial software developers and open source communities become reconciled, the line between the OSS user community and closed source user communities may blur. As more enterprises adopt Linux, for example, users have become more interested in running both Linux and Windows platforms (Adelstein, 2004). The OSS user community could attend to the needs of these ambidextrous users by addressing issues related to usability, hardware and software compatibility.

Alternatively, OSS user communities may try to preserve their character to some degree. For example, although the Firefox team had little marketing budget, they tapped into the resources of the OSS community to launch a promotional campaign. They were able to raise $200,000 from about 10,000 donors in 10 days to pay for a two-page print advertisement in *The New York Times*. In addition, the Firefox team worked closely with international OSS communities to achieve the simultaneous launch of Firefox in 14 languages. Firefox's experience illustrates that communities can remain powerful without having to co-opt commercial interests.

Although the nature and direction of change in OSS user communities cannot be predicted with certainty, it seems inevitable that changes will come, probably rapidly. However, there are probably no easily identifiable drivers of community change. Researchers investigating change in OSS user communities should approach this area with an appreciation for the complexity of organizational and institutional change. This recommendation lies beyond our present scope, but it remains an important consideration.

CONCLUSION

The framework offered in this article is designed to stimulate a new direction in OSS research, one that focuses primarily on software use rather than software development. Although development has attracted the bulk of research interest to date, many important issues pertain to OSS use. OSS users far outnumber OSS developers, and as OSS products become more popular, the number of OSS users will continue to increase. We have identified many of the issues that make OSS different from the use of proprietary and in-house developed software and posed our research questions accordingly. Our research agenda emphasizes the community perspective that has attracted such interest in OSS development research. We believe that many valuable insights can be generated by a focus on OSS user communities.

Beyond the present agenda, we may speculate on the impact of OSS user communities as a model for industries besides software. Although market research and customer relationship management are time-honored ways for companies to feel the pulse of their customers, the phenomenon of OSS user communities suggests that a more active role for customers might be valuable. For example, Threadless.com holds a weekly competition, in which anyone who wishes can upload T-shirt artwork to the company's Web site. Online shoppers can then vote for their favorite designs. Threadless.com prints the winning graphics on limited-edition t-shirts, which are available for purchase at the Web site. Winners are awarded cash or store credits. By 2005, threadless had attracted over 40,000 design submissions (Luman, 2005). This experience offers innovative ideas for community participation in electronic commerce.

In conclusion, the study of OSS user communities offers the potential to learn not only

about OSS projects but also about communities in general. The research agenda proposed in this article suggests many avenues for investigating OSS user communities. Given that there is little extant research on any of the questions raised in this article, our objective is to stimulate thinking about important research areas rather than to summarize findings. Hopefully, the IS research community will begin systematic investigation of the research questions posed here.

REFERENCES

Adler, P. (2001). Market, hierarchy, and trust: The knowledge economy and the future of capitalism. *Organization Science 12*(2), 215-234.

Adelstein, T. (2004). Desktop linux: New linux users changing the face of community. Retrieved July 27, 2006, from http://www.desktoplinux.com/articles/AT3791991696.html

Buckner K. (1996). Computer user groups: The advantage of successful partnership. *International Journal of Information Management, 16*(3), 195-204.

Cole, M. (2002) Virtual communities for learning and development – A look to the past and some glimpses into the future. In K. Ann Renninger & W. Shumar (Eds.), *Building virtual communities: Learning and change in cyberspace***.** Cambridge: Cambridge University Press.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the OpenSource software development paradigm. In *Proceedings of the International Conference of Information Systems* (pp. 21, 58-69).

Fitzgerald, B., & Kenny, T. (2003). OpenSource software the trenches: Lessons from a large-scale OSS implementation. In *Proceedings of the International Conference on Information Systems* (pp. 24, 316-326).

Golden, B. (2005). *Succeeding with OpenSource*. Boston: Addison-Wesley.

Hall, J. (2002, September 6). Free software in Brazil. *Linux Journal, 101.*

Jin, L., Robey, D., & Boudreau, M. C. (2006). Exploring the hybrid community: Intertwining virtual and physical representations of Linux user communities. In *Proceedings of the Administrative Science Association of Canada*, Banff, Canada.

Jin, L., Verma, S., & Negi, A. (2005). Profiling OpenSource: A use perspective across OpenSource communities in the US and India. In *Proceedings of the 36th Annual Meeting of the Decision Sciences Institute*, San Francisco, California.

Lakhani, K. R., & von Hippel, E. (2003). How OpenSource software works: "Free" user-to-user assistance. *Research Policy, 32*(6), 923.

Luman, S. (2005, June). OpenSource Software. *Wired Magazine*, p. 68.

Markus, M. L., Manville, B., & Agres, C. E. (2000). What makes a virtual organization work? *Sloan Management Review, 42*, 13-26.

McHugh, J. (2005, February). The Firefox explosion. *Wired Magazine*, pp. 92-96.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of OpenSource software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Moen, R. (2000). Eric Raymond's tips for effective OpenSource advocacy. Retrieved July 28, 2006, from http://www.itworld.com/AppDev/344/LWD000913expo00/pfindex.html

Moen, R. (2003). Linux user group HOW TO. Retrieved July 28, 2006, from http://www.linux.org/docs/ldp/howto/User-Group-HOWTO.html

Nichols, D. M., & Twidale, M. B. (2003).The usability of OpenSource software. *First Monday, 8*(1).

Onetti, A., & Capobianco, F. (2005). OpenSource and business model innovation: The Funambol case. In M. Scotto & G. Succi (Eds.), In *Proceedings of the 1st International Conference on Opens Source Systems* (pp. 224-227).

Raymond, E. (2001). *The cathedral and the bazaar: Musings on Linux and OpenSource by an accidental revolutionary.* Sebastopol, CA: O'Reilly & Associates.

Rheingold, H. (2000). *Virtual community: Homesteading on the electronic frontier.* Cambridge: The MIT Press.

Robey, D., Schwaig, K., & Jin, L. (2003). Intertwining material and virtual work. *Information and Organization, 13*(2), 111-129.

Verma, S., Jin, L., & Negi, A. (2005). OpenSource adoption and use: A comparative study between groups in the US and India. In *Proceedings of the 11th Annual Americas Conference on Information Systems* (pp. 960-972).

von Hippel, E., & von Krogh, G. (2003) OpenSource software and the "private-collective" innovation model: Issues for organization science. *Organization Science, 14*(2), 209-223.

Wang, J. (2005). The role of social capital in OpenSource software communities. In *Proceedings of the 11th Annual Americas Conference on Information Systems* (pp. 937-943).

Watson, R. T., Boudreau, M. C., Greiner, M., Wynn, D., York, P., & Gul, R. (2005). Governance and global communities. *Journal of International Management, 11,* 125-142.

Watson, R. T., Boudreau, M. C., York, P., Greiner, M., & Wynn, D. (in press). The business of OpenSource. *Communications of the ACM.*

Wellman, B. (2001). Physical place and cyberplace: The rise of personalized networking. *International Journal of Urban and Regional Research, 25*(2), 227-252.

Williams, R. (1973). *Keywords.* Oxford: Oxford University Press.

Winchester, S. (1998). *The professor and the madman: A tale of murder, insanity, and the making of the Oxford English Dictionary.* New York: HarperCollins.

Ye, Y., Kishida, K., Nakakoji, K., & Yamamoto, Y. (2002). Creating and maintaning sustainale OpenSource software communities. In *Proceedings of International Symposium on Future Software Technology 2002 (ISFST '02),* Wuhan, China.

This work was previously published in Information Resources Management Journal, Vol. 20, Issue 1, edited by M. Khosrow-Pour, pp. 68-80, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.3
Social Network Structures in Open Source Software Development Teams

Yuan Long
Colorado State University-Pueblo, USA

Keng Siau
University of Nebraska-Lincoln, USA

ABSTRACT

Drawing on social network theories and previous studies, this research examines the dynamics of social network structures in open source software (OSS) teams. Three projects were selected from SourceForge.net in terms of their similarities as well as their differences. Monthly data were extracted from the bug tracking systems in order to achieve a longitudinal view of the interaction pattern of each project. Social network analysis was used to generate the indices of social structure. The finding suggests that the interaction pattern of OSS projects evolves from a single hub at the beginning to a core/periphery model as the projects move forward.

INTRODUCTION

The information system development arena has seen many revolutions and evolutions. We have witnessed the movement from structured development to object-oriented (OO) development. Modeling methods, such as data flow diagram and entity relationship diagram, are facing new OO modeling languages, such as the unified modeling language (UML) (see Siau & Cao, 2001; Siau, Erickson, & Lee, 2005; Siau & Loo, 2006) and OO methodologies, such as unified process (UP). The latest development includes agile modeling (see Erickson, Lyytinen, & Siau, 2005), extreme programming, and OSS development. While many of these changes are related to systems development paradigms, methodologies, methods, and techniques, the phenomenon of OSS development entails a different structure for software development teams.

Unlike conventional software projects, the participants of OSS projects are volunteers. They are self-selected based on their interests and capability to contribute to the projects (Raymond, 2000). In addition, the developers of OSS projects are distributed all around the world. They communicate and collaborate with each other through the Internet, using e-mails or discussion boards.

Therefore, effective and efficient communication and collaboration are critical to OSS success. However, little empirical research has been conducted to study the underlying interaction pattern of OSS teams, especially the dynamics of the social network structures in OSS development teams. To fill this gap, this study examines the evolvement of social structure in OSS teams. The study contributes to the enhancement of the understanding of OSS development, and provides foundation for future studies to analyze the antecedents and consequences of social networks in the OSS context.

The remainder of the paper is structured as follows. First, prior studies on social network structures in OSS teams are reviewed. Second, theories related to social structure and social network theory are discussed. Third, the research methodology is presented, and the research results are reported. Next, discussions of the results, the limitations, and the implications are provided. The paper concludes with suggestions for future research.

LITERATURE REVIEW

The phenomenon of OSS development has attracted considerable attention from both practitioners and researchers in diverse fields, such as computer science, social psychology, organization, and management. Because of the multifaceted nature of OSS, researchers have investigated OSS phenomenon from varied perspectives. For example, focusing on technical perspective, researchers studied issues such as OSS development methodology (e.g., Jørgensen, 2001) and coding quality (e.g., Stamelos, Angelis, Oikonomu, & Bleris, 2002). Based on social psychology, researchers investigated individual motivation (e.g., Hann, Robert, & Slaughter, 2004), new developers (Von Krogh, Spaeth, & Lakhani 2003), the social network (e.g., Madey, Freeh, & Tynan, 2002), and the social structure (e.g., Crowston & Howison, 2005). In terms of organizational and managerial perspective, researchers examined knowledge innovation (e.g., Hemetsberger 2004; Lee & Cole 2003, Von Hippel & von Krogh, 2003) and the governance mechanism (e.g., Sagers 2004).

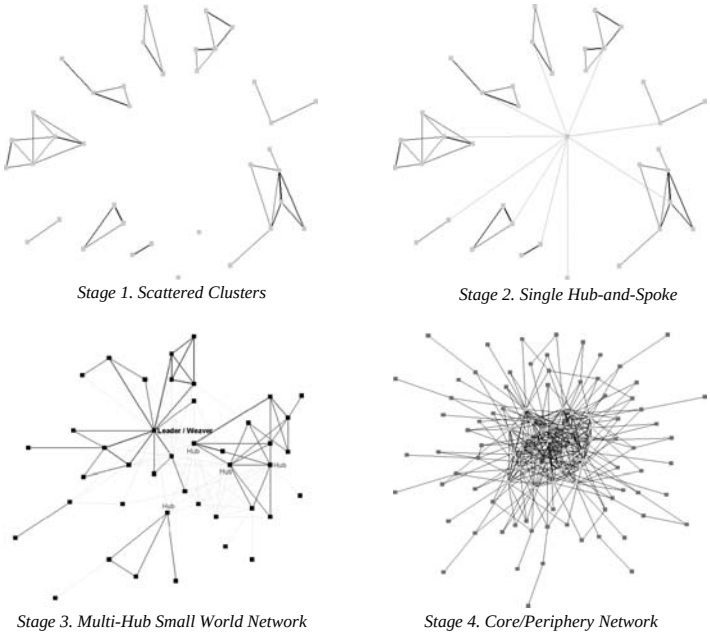
An OSS development team is essentially a virtual organization in which participants interact and collaborate with each other through the Internet. Compared to conventional organizations, the structure of virtual organizations is decentralized, flat, and nonhierarchical (Ahuja & Carley 1999). However, some researchers challenge the belief (e.g., Crowston & Howison 2005; Gacek & Arief, 2004;; Mockus, Fielding, & Herbsleb, 2000; Mockus, Fielding, & Herbsleb, 2002; Moon & Sproull, 2000). They argue that the social structure of OSS projects is hierarchical rather than flat, like a tree (Gacek & Arief, 2004) or an onion (Crowston & Howison, 2005). The social structure of OSS teams directly influences the collaboration and the decision-making process and further affects the overall performance of the teams as well as individuals' perception of belonging and satisfaction. Therefore, one wonders what form of social structure might be present in the OSS development and what type of structure will emerge—centralized or decentralized, hierarchical or nonhierarchical, onion-like or tree-like, or a combination of the above depending on certain specific situations?

A social network, as stated by Krebs and Holley (2004), is generally built in four phases, each with its own distinct topology (as shown in Figure 1).

1. scattered clusters,
2. single hub-and-spoke,
3. multihub small-world network, and
4. core/periphery.

Most organizations start from isolated and distributed clusters (Krebs & Holley, 2004). Then an active leader emerges and takes responsibility for building a network that will connect the sepa-

Figure 1. Four phases of social structures (from Krebs and Holley 2004)



Stage 1. Scattered Clusters

Stage 2. Single Hub-and-Spoke

Stage 3. Multi-Hub Small World Network

Stage 4. Core/Periphery Network

rate clusters. However, this single-hub topology is fragile. With more participants entering the group, the leader changes his/her role to a facilitator and helps to build multiple hubs, which is stage three. The core/periphery model, the last stage, is the most stable structure. In the core/periphery model, the network core encompasses key group members who are strongly connected to each other, while the periphery contains members who are usually weakly connected to each other as well as to the core members. With the general network building phases in mind, one can argue that OSS projects may follow the same four stages (i.e., scattered clusters, single hub-and-spoke, multihub small-world network, and core/periphery model). But is that true for OSS projects? How does the social structure of OSS teams evolve over time?

Our research addresses the following two questions:

1. What is the social structure of OSS teams?
2. How does the social structure evolve over time?

THEORETICAL FOUNDATION

Social Structure and Social Interaction

Social structure, as suggested by Schaefer and Lamm (1998), refers to the way in which society is organized into predictable relationships. Social structure can be considered in terms of three aspects—actors, their actions, and their interactions. The social actor is a relatively static concept addressing issues such as roles, positions, and statuses. Individual actors are embedded in the social environment and, therefore, their actions are largely influenced by the connections between each other. Social interaction is generally regarded as the way in which people respond to one another. These interaction patterns are to some extent independent of individuals. They exert a force that

shapes both behavior (i.e., actions) and identity (i.e., actors) (Schaefer & Lamm, 1998).

Research on social interaction focuses on how individuals actually communicate with each other in group settings. These studies address issues such as the interaction patterns, the underlying rules guiding interaction, the reasons accounting for the way people interact, and the impacts of the interaction patterns on the individual behavior and the group performance. These issues begin by questioning what might be the interaction pattern in a specific social setting and that addresses our research question—understanding social interaction of OSS project teams.

Social Network Theory

Social network theory focuses on studying actors as well as their relationships in specific social settings. Network theory is analogous to systems theory and complexity theory. It is an interdisciplinary theory stemming from multiple traditional fields, including psychology, which addresses individuals' perception of social structure; anthropology, which emphasizes social relationships; and mathematics, which provides algorithms (Scott, 2000).

Based on the view of social network, the world is composed of actors (also called nodes) and ties between them. The ties can represent either a specific relationship (such as friendship and kinship) between a pair of actors or define a particular action which an actor performs. Different kinds of ties specify different networks and are typically assumed to function differently. For example, the ties in a family network are distinctive from those in a working network, and the centrality in the "who loves whom" network obviously has different meaning than the centrality in the "who hates whom" network.

Social network theory is based on the intuitive notion that the social interaction patterns are essential to the individuals who reflect them. Network theorists believe that how individuals behave largely depends on how they interact with others and how they are tied to a social network. Furthermore, besides individual behavior, network theorists believe that the success or failure of societies and organizations often depends on the internal interaction pattern (Freeman, 2004).

Besides the theoretical essence, social network theory is also characterized as a distinctive methodology encompassing techniques for data collection, statistical analysis, and visual representation. This approach is usually called social network analysis and will be discussed in the research methodology section. This paper draws on the social network theory to study the interaction pattern of OSS development project.

RESEARCH METHODOLOGY

Social Network Analysis

Social network analysis is used in our study to investigate the interaction pattern of the OSS development process. Social network analysis focuses on uncovering the interaction pattern of interdependent individuals (Freeman, 2004). Through a structural analysis of a social network diagram, a map depicting actors as well as the ties that connect them, social network analysis can reveal the patterns of relationships and the relative position of individuals in a specific social setting. This approach has been effectively used in organizational research, social support, mental health, and the diffusion of information (Freeman, 2004).

Social network analysis is used in our study for two primary reasons. First, the purpose of social network analysis fits our research objective. Social network analysis aims to analyze the relationship among a set of actors instead of their internal attributes. Our research aims to reveal the interaction pattern of OSS project teams. Therefore, social network analysis is helpful in answering our research questions.

Second, the rich interactive data extracted from OSS projects presents a "gold mine" for social network analysis. Social network analysis is grounded in the systematic analysis of empirical data. However, there is usually a lack of convenient and objective resources from which to draw the links (i.e., relationships) among actors. Most OSS projects have online mailing lists, forums, and tracking systems that are open to public, thus providing a rich set of longitudinal data. Based on these public data, researchers are able to capture input data sets for social network analysis.

Longitudinal Data

Because we are interested in studying how the interaction pattern of OSS projects evolves over time, cross-sectional observations of interaction networks are not sufficient. Cross-sectional observations of social networks are snapshots of interactions at a point in time and cannot provide traceable history, thus limiting the usefulness of the results. On the other hand, longitudinal observations offer more promise for understanding the social network structure and its evolvement. In this study, we extracted longitudinal data on OSS projects.

Case Selection

OSS projects were selected from the SourceForge[1], which is the world's largest Web site hosting OSS projects. SourceForge provides free tools and services to facilitate OSS development. At the time of the study, it hosted a total of 99,730 OSS projects and involved 1,066,589 registered users (This data was retrieved on May 4, 2005). Although a few big OSS projects have their own Web sites (such as Linux), SourceForge serves as the most popular data resource for OSS researchers.

Following the idea of theoretical sampling (Glaser & Strauss, 1967), three OSS projects were selected from SourceForge in terms of their similarities and differences. Theoretical sampling requires theoretical relevance and purposes (Orlikowski, 1993). In terms of relevance, the selection ensures that the interaction pattern of OSS projects over time is kept similar. Therefore, the projects that are selected have to satisfy two requirements. First, the projects must have considerable interaction among members during the development process. All three projects had more than 10 developers, and the number of bugs reported was more than 1,000. Second, since we are interested in the interaction over time, the projects must have a relatively long life. In our case, all three projects were at least three years old.

In addition to similarities, differences are sought among cases because the study aims to study interaction patterns of various OSS projects. Therefore, the three projects differ on several project characteristics, such as project size, project type, and intended audience. These differences enable us to make useful contrasts during data analysis.

The Table 1 summarizes the three projects with a brief description.

Data Collection and Analysis

Social network analysis can be divided into the following three stages (Borgatti, 2002).

1. Data collection. In this stage, researchers collect data, using surveys and questionnaires, or from documents and other data resources, and generate input data sets for social network analysis.
2. Statistical analysis. Based on mathematics algorithms, this stage generates network indices concerning group structure (such as centralization and density) as well as individual cohesion (such as centrality and bridges).
3. Visual representation. This stage employs network diagrams to show the interaction structure as well as the position of specific actors.

Table 1. Summary of three projects

		Net-SNMP	Compiere ERP + CRM	J-boss
Description		Net-SNMP allows system and network managers to monitor and manage hosts and network devices.	Compiere is a smart ERP+CRM solution covering all major business areas—especially for small-medium enterprises.	JBoss is a leading open source Java application server. After Linux and Apache, it is the third major open source project to receive widespread adoption by corporate IT.
Similarities	Bug reports (more than 1,000 bugs)	1,361	1,695	2,296
	Development duration (more than 3 years)	55 months (registered on 10/2000)	47 months (registered on 6/2001)	50 months (registered on 3/2001)
Differences	Software type	Internet, network management	Enterprise: ERP+CRM	J2EE-based middleware
	Group size (number of developers)	Small (14)	Median (44)	Large (75)
	Intended audience	Developers, system administrators	Business	Developers, system administrators

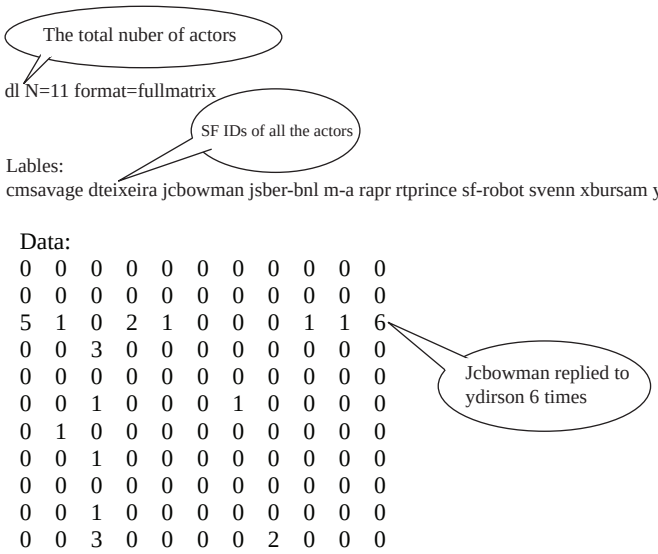
Note: Data was retrieved in April 2004 from SourceForge.net.

First is the data collection. The data were collected in April 2005 from SourceForge.net. Data were extracted from the bug tracking system of each project. We chose the bug tracking system as the primary data resource for three reasons. First, open source software is characterized as peer review of open codes. Raymond (1998) proposed the "Linus' law" in his well-known essay *The Cathedral and the Bazaar*— "Given enough eyeballs, all bugs are shallow." Therefore, the bug system can be viewed as the representative of open source spirit. Second, compared to other development activities, such as patch posting and feature request, the bug-fixing process is the most active procedure to illustrate the close collaboration between developers and users as well as among developers themselves. Finally, the bug tracking system provides rich data that record the interactive process.

A Web spider program, which is based on the work of Crowston and Howison (2005) with necessary revision, was used to download the bug tracking Web pages from the project Web site. After that, a Web parsing program was developed to analyze the Web pages. The interaction data was extracted from the bug tracking Web pages month-by-month, starting from the date the project was registered until the date the data was downloaded for this study. The output of this stage is a social matrix describing the interaction among users. Figure 2 shows an example of such a social matrix for an OSS project. In the matrix, each row or column represents a distinctive participant, which is identified by a unique SourceForge user identity. The values of cells indicate the degree of the interaction between each pair of participants, which is counted by the amounts of messages that participant A (i.e., row A) replied to participant B (i.e., column B).

Second is the statistical analysis. Our study focuses on two important and distinctive properties of network structure—group centralization and core/periphery fitness. Ucinet, which was devel-

Figure 2. An example of the social matrix for an OSS project



oped by Borgatti, Everett, and Freeman (2002), was used to calculate these two properties.

Group centralization, as suggested by Wasserman and Faust (1994), refers to the extent to which a network revolves around a single center. A typical case of centralized structure is a "star" network. Group centralization can be viewed as a rough measure of inequity between individual actors, and the variability and dispersion of the interaction pattern.

The other property is core/periphery fitness. It measures the extent to which the network is close to a perfect core/periphery structure. The core/periphery structure depicts a dense, connected group surrounded by a sparse, unconnected periphery. The opposite structure is clique, which represents a structure of multiple subgroups, each with its own core and peripheries (Borgatti, 2002).

Finally is the visual representation. We used Ucinet (Borgatti et al., 2002) to draw the interaction networks for each of the three projects.

RESEARCH RESULTS

Snapshots of the Three Projects

Monthly data were extracted from the bug tracking system of each project. To illustrate the trend of interaction pattern, we provide three snapshots for each project (see Figures 3-5)[2].

Table 2 summarizes the relevant network characteristics of each project. In addition to the group centralization and core/periphery fitness, we also report other network characteristics, such as density, average distance, and distance-based cohesion. Density depicts how "close" the network looks, and it is a recommended measure of group cohesion (Blau, 1977; Wasserman & Faust 1994). The value of density ranges from 0 to 1. Average distance refers to average distance between all pairs of nodes (Borgatti, 2002). Distance-based cohesion takes on values between 0 and 1—the larger the values, the greater the cohesiveness.

Looking at the statistical results and the network plots, we can observe the following.

First, the evolvement of interaction patterns of the three projects reveals a general trend. As

Table 2. Three snapshots for each project

		Net-SNMP	Compiere	JBoss
Group centralization (%)	1st.	9.420	15.624	4.931
	2nd.	3.071	2.294	4.45
	3rd.	2.316	1.288	4.12
Core/periphery fitness	1st.	0.674	0.774	0.485
	2nd.	0.654	0.796	0.477
	3rd.	0.651	0.765	0.501
Density	1st.	0.0235	0.0584	0.0073
	2nd.	0.0109	0.0610	0.0039
	3rd.	0.0072	0.0571	0.0026
Average distance	1st.	2.546	2.711	3.438
	2nd.	2.794	2.302	3.281
	3rd.	2.917	2.278	3.239
Distance-based cohesion	1st.	0.181	0.198	0.118
	2nd.	0.143	0.253	0.147
	3rd.	0.141	0.279	0.136

shown in the network plots (i.e., Figures 3-5), the interaction pattern develops from a centralized one with a single (sometimes dual) hub with several distributed nodes, to a core/periphery structure that has a core (a group of core developers) together with several hangers-on (periphery). Intense interactions exist within the core (among several core developers) and between each core member and his/her periphery. However, only loose relationships exist among peripheries. This pattern suggests a layer structure (i.e., core with its periphery) instead of a complete flat one with equal positions across all the members.

Second, although the interaction patterns of the three projects share some commonalities, their exact shapes are different. The shape of Net-SNMP (as shown in Figure 3) is more like a typical core/periphery compared to the other two. Compiere (as shown in Figure 4) keeps two cores, and the shape looks like a dumbbell. Jboss (as shown in Figure 5), which is the largest project among the three, maintains a more complex structure that shows multiple layers instead of just one core with the rest as peripheries (e.g., Net-SNMP)

Third, as time goes by, the group centralization decreases across the three projects, showing a trend that moves from a centralized structure to a decentralized structure irrespective of project sizes (The three projects with different project sizes are shown in Table 1), project types, and intended audience.

Fourth, the indices of core/periphery fitness of each project fluctuate slightly but maintain a relatively high value (larger than 0.5 on average). However, no observable trend was found across projects.

Fifth, since each project has a relatively large group (i.e., more than 100 actors including all the registered users), the values of density are relatively low with little variation. Therefore, density is not appropriate for comparing the projects.

From the snapshots, we observed the following trend. First, the OSS interaction network evolves into a core/periphery structure. Second,

Figure 3. Interaction patterns of Net-SNMP

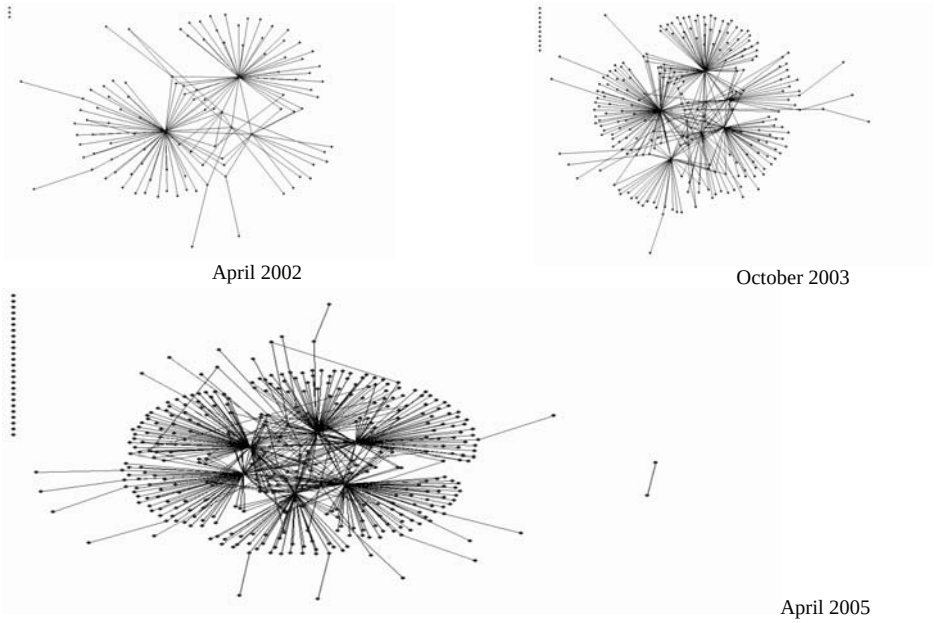
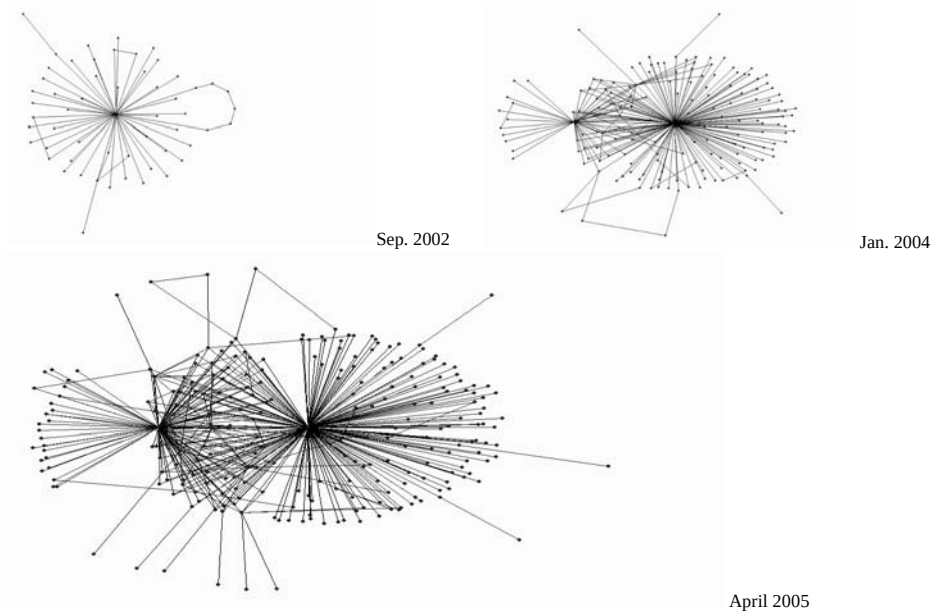
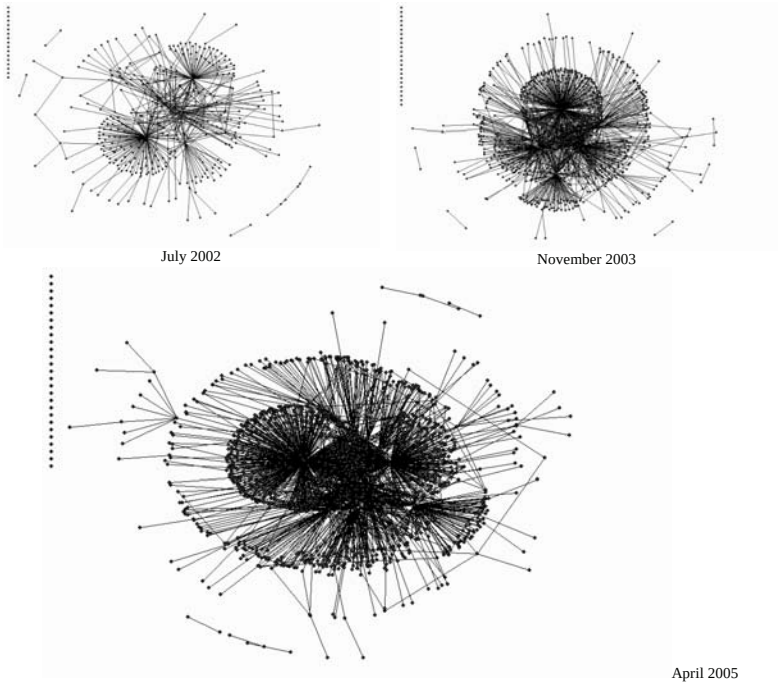


Figure 4. Interaction patterns of compiere CRM+ERP



group centralization decreases over time. Third, core/periphery fitness stays relatively stable. To verify our observations, we used longitudinal data generated from the bug tracking systems to analyze the evolvement of interaction pattern (discussed in the following section).

Figure 5. Interaction patterns of JBoss



Group Centralization and Core/Periphery Fitness

Table 3 shows the values of both group centralization and core/periphery fitness over time based on the monthly interaction data. For each figure, the Y-axis indicates the social structure indices (i.e., group centralization or core/periphery fitness), and the X-axis reflects the time dimension.

Two primary observations can be made based on the statistical analysis.

First, the group centralization shows a decreasing trend across the three projects. This observation indicates that as OSS projects progress, the social network structure evolves from centralized to decentralized and then stabilizes. Also, the three figures suggest no substantial differences in the trend among the three projects.

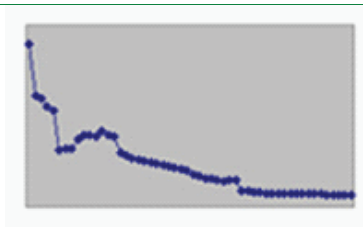
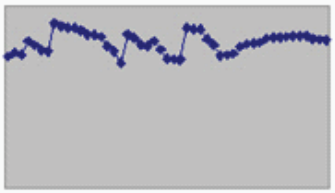
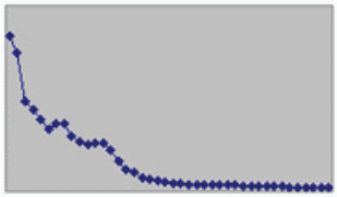
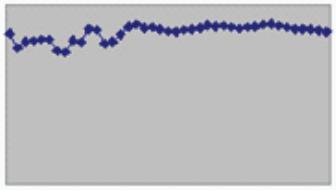
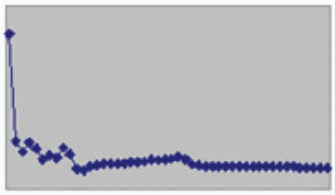
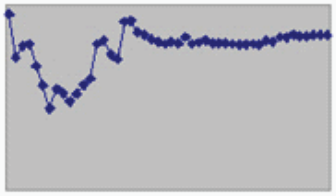
Second, the core/periphery index is maintained at a relatively stable level for each project over time. In addition, the average fitness value stays relatively high for each project (larger than 0.5), indicating a closeness to a perfect core/periphery structure.

Besides a holistic view of network structure for OSS projects, the results also reveal other interesting findings. For example, by examining the core members over time, we found a relatively stable core for each project. The cores are usually project developers and administrators. This observation further demonstrates the existence of strong and stable core as well as loose hangers-on in OSS projects.

DISCUSSION

This research uses the longitudinal data of three OSS projects selected from SourceForge to study the social network structures of OSS teams. The purpose of this study is to investigate the evolvement of interaction patterns of OSS project teams.

Table 3. Group centralization and core/periphery fitness based on longitudinal data

	Group centralization	Core/periphery fitness
Net-SNMP		
Compiere		
JBoss		

The research results suggest a decrease of group centralization over time and a tendency of core/periphery structure in OSS project teams.

The network plots (as shown in Figures 3-5) indicate a layer structure instead of a flat one as suggested by earlier literature. The interaction pattern evolves from a single hub to a core/periphery structure. As the number of participants increases, a core with only one person (who may be the starter/initiator of the project) cannot satisfy the increasing requirements of development and communication. Therefore, other developers or active users join the core to serve as key members of the project. This results in a more stable structure, and the project is less dependent on a single leader.

With the growth of a software project, more people are attracted to the project. The original leader may not be able to solve all the technical problems encountered in the development process. Each key member has his/her specialty, is responsible for solving relevant problems, and has his/her own periphery in the network plot. Although there are multiple peripheries in the project, collaboration among key members in the project is vital. This phenomenon of distribution and collaboration can be viewed as a critical success factor of OSS development. And the evolvement is vividly demonstrated in our social network analysis.

In a way, the social structure of OSS projects is both centralized and decentralized. On one hand, it is centralized in the sense that there is a core that consists of key members. These key members are responsible for various issues encountered during the development process. On the other hand, it is decentralized in the sense that the decision or communication core is not concentrated on one or two members but a group of key members.

Like any other research, this research has its share of limitations. First, the cases were only selected from SourceForge.net. Although Source-

Forge is the world's largest Web site hosting open source software, there are also some other similar Web sites. Therefore, the total number of OSS projects in SourceForge cannot be viewed as the whole population. However, as we argued before, SourceForge is probably the best data collection site for this research.

Second, the bug tracking system was chosen as our data resource. The selection of bug tracking system as our research setting and data resource may have had an effect on the outcome and results. Besides the bug tracking forum, there are other forums that also provide space for participants to communicate with one another, such as mailing lists and feature requests. However, as we highlighted earlier, the bug systems are the most active forum, providing rich interaction data. The bug tracking systems also represent the spirit of open source software development. Examining the interaction data from other forums can be one of our research extensions in the future.

Third, because our research objective is to investigate interaction pattern, we chose projects that have a relatively large number of developers, a large number of bug reports, and relatively long history. Although we tried to involve different types of projects (i.e., different project sizes, project types, and intended audience), these three cases may not be representatives of OSS projects, for example, small projects with only one or two developers and few interactions. Increasing the sample size and including various types of OSS projects is one of our future research directions.

IMPLICATIONS AND CONCLUSION

This paper examines the interaction patterns of OSS teams. The research findings suggest that the interaction structure starts from a single hub and evolves to a core/periphery model. We argue that the social structure of OSS teams is both centralized and decentralized. It is centralized in the sense that there exists a relatively stable core that consists of a group of key developers. It is also decentralized because of distributed decision making among key developers and the broad collaboration between developers and users as well as among developers themselves.

The paper presents the evolvement of the social structure of OSS projects from a longitudinal perspective. It also provides empirical evidence of the change of interaction patterns from a single hub to a core/periphery model. Moreover, the paper utilizes social network analysis as the research method. This approach has been shown in this research as an effective tool in analyzing the social structure in OSS teams.

Social structure is an important variable for understanding social phenomenon. Open source software, with its open and unique nature, attracts researchers to ask a series of questions. For example, how do participants of OSS projects interact and collaborate with each other? What factors facilitate the interaction and the collaboration? And further, how does the collaboration affect project performance of OSS teams? Social network analysis is a good approach to investigate these questions. This research represents a pioneering effort in this direction.

REFERENCES

Ahuja, M., & Carley, K. (1999). Network structure in virtual organizations. *Organization Science, 10*(6), 741-747.

Blau, P. M. (1977). *Inequity and heterogeneity.* New York: Free Press.

Borgatti, S. (2002). *Basic social network concepts.* Retrieved from http://www.analytictech.com/networks/basic%20concepts%202002.pdf

Borgatti, S. P., Everett, M. G., & Freeman, L. C. (2002). Ucinet for Windows: Software for

social network analysis. Harvard, MA: Analytic Technologies.

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday, 10*(2).

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management, 16*(4), 88-100.

*Freeman, L. C. (*2004). *The development of social network analysis: A study in the sociology of science*. Vancouver, Canada: Empirical Press.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE Software, 21*(1), 34-40.

Glaser, B. G., & Strauss, Anselm L., (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Chicago, Aldine Publishing Company

Hann, H., Robert, J., & Slaughter, S. (2004). Why developers participate in open source software projects: An empirical investigation. In *Twenty-Fifth International Conference on Information Systems*, (pp. 821-830). Washington, DC: .

Hemetsberger, A. (2004). Sharing and creating knowledge in open-source communities: The case of KDE. *The Fifth European Conference on Organizational Knowledge, Learning, and Capabilities in Innsbruck, Austria*.

Jørgensen, N. (2001). Putting it all in a trunk: Incremental software development in the free-BSD open source project. *Information Systems Journal, 11,* 321-336.

Krebs, V., & Holley, J. (2004). *Building sustainable communities through network building.* Retrieved from http://www.orgnet.com/BuildingNetworks.pdf

Lee, G. K., & Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science, 14*(6), 633-649.

Madey, G., Freeh, V., & Tynan R. (2002). The open source software development phenomenon: An analysis based on social network theory (AMCIS2002). Dallas, TX.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2000). A case study of open source software development: The Apache server. ICSE 2000.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309–346.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of Linux kernel. *First Monday, 5*(11).

Orlikowski, W. J. (1993). CASE tools as organizational change: investigating incremental and radical changes in systems development. *MIS Quarterly, 17*(3), 309-340.

Raymond, E. S. (1998). The cathedral and the bazaar. *First Monday, 3(3),* Retrieved January , 2005, from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Sagers, G. W. (2004). The influence of network governance factors on success in open source software development projects. In *Twenty-Fifth International Conference on Information Systems* (pp. 427-438). Washington, DC:

Schaefer, R. T., & Lamm, R. P. (1998). *Sociology* (6th ed.). McGraw-Hill.

Scott, J. (2000). *Social network analysis. A handbook* (2nd ed.). London: SAGE Publications.

Siau, K., & Cao, Q. (2001). Unified modeling language—A complexity analysis. *Journal of Database Management, 12*(1), 26-34.

Siau, K., Erickson, J., & Lee, L. (2005). Theoretical versus practical complexity: The case of UML. *Journal of Database Management, 16*(3), 40-57.

Siau, K., & Loo, P. (2006). Identifying difficulties in learning UML. *Information Systems Management, 23*(3), 43-51.

Stamelos, I., Angelis, L., Oikonomu, A., & Bleris, G. L. (2002). Code quality analysis in open-source software development. *Information Systems Journal, 12*(1), 43-60.

Von Hippel, E., & Von Krogh, G. (2003). Open source software and the „private-collective" innovation model: Issues for organization science. *Organization Science, 14*, 209-223.

Von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: A case study. *Research Policy, 32*(7), 1217-1241.

Wasserman, S., & Faust, K., (1994). *Social Network Analysis: Methods and Applications.* New York: Cambridge University Press.

ENDNOTE

1 The Web address for SourceForge is www.sourceforge.net.

2 The three time stamps for Net-SNMP are 4/2002, 10/2003 and 4/2005; for Compiere are 9/2002, 1/2004 and 4/2005; and for Jboss are 7/2002, 11/2003 and 4/2005.

This work was previously published in Journal of Database Management, Vol. 18, Issue 2, edited by K. Siau, pp. 25-40, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.4
The Impact of Ideology on the Organizational Adoption of Open Source Software

Kris Ven
University of Antwerp, Belgium

Jan Verelst
University of Antwerp, Belgium

ABSTRACT

Previous research has shown that the open source movement shares a common ideology. Employees belonging to the open source movement often advocate the use of open source software within their organization. Hence, their belief in the underlying open source software ideology may influence the decision making on the adoption of open source software. This may result in an ideological—rather than pragmatic—decision. A recent study has shown that American organizations are quite pragmatic in their adoption decision. We argue that there may be circumstances in which there is more opportunity for ideological behavior. We therefore investigated the organizational adoption decision in Belgian organizations. Our results indicate that most organizations are pragmatic in their decision making. However, we have found evidence that suggests that the influence of ideology should not be completely disregarded in small organizations.

INTRODUCTION

The free software movement—led by Richard M. Stallman—has always taken an ideological, political view on software. Adherents to the free software movement advocate that all software should be free, in the sense that it should be free to read, modify, and distribute. The open source movement on the other hand was created in order to facilitate the introduction of free software in organizations and takes a more pragmatic stance in its efforts to market open source software (OSS). Previous research has shown that the open source movement is characterized by a shared, underlying ideology (e.g., Ljungberg, 2000; Bergquist & Ljungberg, 2001). Lately, an increasing number of developers are hired by commercial organizations to work on OSS projects. These developers may or may not share the OSS ideology. Nevertheless, many adherents to the open source movement still feel connected to the OSS ideology. Moreover, commercial organizations still need to find a balance between their commercial objectives and the

traditional values of the open source movement (Fitzgerald, 2006).

Many organizations have already adopted OSS, especially mature server software such as Linux and Apache. Research on the organizational adoption of OSS has shown that its use was frequently a bottom-up initiative, suggested by technical employees within the organization who are an adherent to the open source movement (Dedrick & West, 2003; West & Dedrick, 2005; Lundell, Lings, & Lindqvist, 2006). In some cases, decision makers could also be considered an adherent to the open source movement. These employees will take on the role of *boundary spanners* in their organization, bringing the organization in contact with new innovations (Tushman & Scanlan, 1981). West and Dedrick (2005) have found in their study on American organizations that although such employees try to ensure that an open source alternative is considered in the decision making, the final decision is made on pragmatic grounds (i.e., based on characteristics of the software such as cost, reliability, and functionality), and not based on ideological feelings towards OSS. The organizations included in their study are rather large,[1] which may have had an impact on their results.

We argue that it is useful to perform a similar study in a context in which there is more opportunity for ideological behavior. We expect that this might be the case in smaller organizations. In order to investigate whether decision making in small organizations is ideological, we have conducted 10 case studies in Belgian organizations to investigate the organizational adoption of OSS. The article is structured as follows. We will start by discussing the theoretical background of this study. Next, we will discuss our research design. Subsequently, we will present the results of our study, focusing on three organizations that used fairly ideological decision making. This is followed by a discussion of our findings. Finally, we will offer our conclusions.

THEORETICAL BACKGROUND

OSS Ideology

Numerous definitions have been proposed in literature for the term "ideology." Usually, the term is used in a pejorative meaning. Such use implies that an ideology is based on false beliefs of reality. Several authors however recommend against using such a perspective (e.g., Hamilton, 1987). The definition of ideology that we will use in this article is proposed by Hamilton (1987, p. 38):

"An ideology is a system of collectively held normative and reputedly factual ideas and beliefs and attitudes advocating a particular pattern of social relationships and arrangements, and/or aimed at justifying a particular pattern of conduct, which its proponents seek to promote, realise, pursue or maintain."

This definition is non-judgmental, and as a result we do not make any pronouncements with respect to the correctness of the beliefs, values, and norms that characterize an ideology. Hence, acting according to an ideology will not necessarily have negative consequences for the organization.

Previous research has described several ideological principles of the open source movement (e.g., Markus, Manville, & Agres, 2000; Ljungberg, 2000; Stewart & Gosain, 2006). This ideology has been shown to enhance the effectiveness of the OSS community (Stewart & Gosain, 2006). Stewart and Gosain (2006) identified a number of underlying norms, beliefs, and values of the open source movement (see Table 1). These norms, beliefs, and values are proposed as the tenets of the OSS ideology.

The tenets listed in Table 1 are used to describe the attitudes of developers within the OSS community. We argue however that some of the OSS beliefs and values (i.e., tenets 4–15 in Table 1) can also be shared by technical employees and decision makers in organizations. Hence, it is interesting to investigate whether decision mak-

Table 1. Tenets of open source ideology (Stewart & Gosain, 2006, pp. 294–295)

OSS Norms	OSS Beliefs	OSS Values
(1) *Forking*—There is a norm against forking a project, which refers to splitting the project into two or more projects developed separately. (2) *Distribution*—There is a norm against distributing code changes without going through the proper channels. (3) *Named Credit*—There is a norm against removing someone's name from a project without that person's consent.	(4) *Code Quality*—Open source development methods produce better code than closed source. (5) *Software Freedom*—Outcomes are better when code is freely available. (6) *Information Freedom*—Outcomes are better when information is freely available. (7) *Bug Fixing*—The more people working on the code, the more quickly bugs will be found and fixed. (8) *Practicality*—Practical work is more useful than theoretical discussion. (9) *Status Attainment*—Status is achieved through community recognition.	(10) *Sharing*—Sharing information is important. (11) *Helping*—Aiding others is important. (12) *Technical Knowledge*—Technical knowledge is highly valued. (13) *Learning*—There is a value on learning for its own sake. (14) *Cooperation*—Voluntary cooperation is important. (15) *Reputation*—Reputation gained by participating in open source projects is valuable.

ers who share these ideological ideas of the open source movement make an ideological—rather than pragmatic—decision. Although the study of West and Dedrick (2005) has shown that decision making on OSS is pragmatic, we believe that this may be different in small organizations. Some authors have pointed out that decision making with respect to IT in small organizations is often the responsibility of a single individual (Harrison, Mykytyn, & Riemenschneider, 1997; Riemenschneider, Harrison, & Mykytyn, 2003). We argue that the impact of the OSS ideology will be greater if a single decision maker—who can be considered an OSS advocate—is present in the organization. In such situations, the adoption decision may be ideological since personal traits and beliefs of the decision maker are more likely to impact the final decision than in larger organizations.

Mindful Innovation

Nowadays, many things require the attention of managers, making their attention a scarce resource (Hansen & Haas, 2001; Swanson & Ramiller, 2004). One of the consequences is that much innovation in organizations is actually driven by bandwagon phenomena, in which organizations mimic the adoption behavior of other organizations and do not properly evaluate alternatives (Abrahamson, 1991; Swanson & Ramiller, 2004). Recently, the bandwagon phenomenon has been framed into the broader context of *mindful innovation* (Swanson & Ramiller, 2004; Fiol & Connor, 2003). The concept of mindfulness originated in psychology and denotes a state of an individual involving: (1) openness to novelty; (2) alertness to distinction; (3) sensitivity to different contexts; (4) implicit, if not explicit, awareness of multiple perspectives; and (5) orientation in the present (Sternberg, 2000). Decision makers in organi-

zations who are mindful have a "watchful and vigilant state of mind" (Fiol & Connor, 2003). An organization that innovates mindfully with IT will therefore not take generalized claims about advantages for granted, but will critically examine their relevance and validity in the organization-specific context (Fiol & Connor, 2003). Mindless innovation, on the other hand, is characterized by "…acting on automatic pilot, precluding attention to new information, and fixating on a single perspective" (Fiol & Connor, 2003; Weick, Sutcliffe, & Obstfeld, 1999).[2] Such innovation may result in making premature decisions based on beliefs that do not necessarily accurately reflect reality (Butler & Gray, 2006). Hence, a dogmatic belief in the OSS ideology may lead to mindless adoption, in which no proprietary alternatives are considered.

Swanson and Ramiller (2004) note that boundary-spanning activities are important for mindful organizational decision making, in order to obtain information on the innovation. We argue that in the case of OSS, this information may be ideologically colored. As a result, the presence of boundary spanners in the adoption of OSS may actually lead to ideological (mindless) behavior instead, especially if decision makers share the OSS ideology. There are at least two factors that can facilitate ideological behavior in such context. First, decision structures in small organizations tend to be less formal (bureaucratic) than in large organizations. Fiol and Connor (2003) argue that underspecified decision structures may encourage further mindless behavior, if decision making was mindless to begin with. Second, Swanson and Ramiller (2004) point out that although personal mindfulness with respect to innovation does not necessarily equate to organizational mindfulness, it will definitely have an impact on it.

Ideology vs. Pragmatism

In order to investigate whether decision making in organizations exhibits ideological characteristics, we need to determine how ideological behavior can be identified. Based on the work of Stewart and Gosain (2006), we determine whether decision makers and other employees shared some of the beliefs and underlying principles (tenets) of the free and open source movements (see Table 1), *and* did not properly assess their relevancy for the organization. For example, proponents may argue that software should be free (similar to the views of the FSF), may have a negative attitude towards proprietary software, or may be convinced that OSS delivers software of a higher quality (Stewart & Gosain, 2006; Ljungberg, 2000). Consequently, decision makers may have a strong preference for using OSS, without (properly) considering proprietary alternatives. Such decision making may result in a less than optimal solution for the organization. In fact, decision makers are in that case rather mindless in their decision making. Mindless organizations will pay little attention to the organization's specifics or to studying new innovations. This will result in making decisions on "autopilot," using a single perspective (Swanson & Ramiller, 2004; Fiol & Connor, 2003). This means that the beliefs of the OSS ideology are taken for granted, without considering their suitability in the organization-specific context.

On the other hand, we consider an organization to be pragmatic in its decision making when the organization does not exhibit any of the tenets of the OSS ideology, or when decision makers do not take any claims of the OSS ideology for granted, but carefully examine their implications in the organization-specific context. Such organizations are mindful in their decision making. This means that decision makers base their decision on the characteristics of the innovation itself and consider how well the innovation fits within the organization. Pragmatic decision makers will probably consider both proprietary and OSS alternatives, outweigh the benefits of all alternatives, and choose the best solution based on factors such as cost and product features. In

this case, no favoritism towards using OSS should be present.

It must be noted that ideological and pragmatic decision making is not a black and white phenomenon. In practice, we expect organizations to exhibit some ideological and some pragmatic characteristics. This is consistent with Geuss (1994), who remarks that an ideology is generally not only composed of the beliefs and values that are shared by *all* members of a group. Consequently, not all adherents to the open source movement will share *all* values proposed by the OSS ideology. This is similar to the statement of Ljungberg (2000) who suggests that developers vary in their adherence to the OSS ideology. Hence, there are many shades of gray in this classification. In this article, we will discuss decision making in three organizations in our sample which clearly exhibited ideological behavior.

RESEARCH DESIGN

To investigate whether decision making is ideological or pragmatic, we studied the organizational adoption of OSS in Belgian organizations. In this study, decision makers were questioned about the reasons for using OSS and their attitudes towards the open source movement. Based upon the information obtained from these organizations, we were able to determine whether their decision making was either pragmatic or rather ideological.

Scope

We decided to focus mainly on the adoption of open source *server* software. We use the term open source server software to refer to both open source operating systems (such as Linux and FreeBSD) and other OSS for server use (for example, the Apache Web server or the Bind name server). This choice is motivated by the fact that this type of OSS is generally considered to be stable and mature, and is already in use by a significant number of organizations. A similar research approach has been undertaken by other researchers (e.g., West & Dedrick, 2005). On the other hand, we also gathered information on other OSS that was being used in the organizations (such as desktop software, development, and networking tools).

Methodology

We used the exploratory case study approach to study the organizational adoption decision on open source server software. The case study approach is well suited to study a contemporary phenomenon in its natural setting, especially when the boundaries of the phenomenon are not clearly defined at the start of the study (Yin, 2003; Benbasat, Goldstein, & Mead, 1987). We conducted a series of in-depth face-to-face interviews with informants from 10 Belgian organizations to identify the factors that influence the decision to use open source server software as well as their attitudes towards the open source movement. Organizations were selected from the population of all Belgian organizations and were sampled on the basis of two criteria: the size of the organization measured by the number of employees and the sector in which the organization operated. Organizations were only included in our sample if they were using open source server software at the time of our study. Informants within each organization were selected using the *key informant method*. Since the use of a single informant has been shown to give inconsistent results (Phillips, 1981), we tried to speak to both a senior manager (e.g., the IT manager) and a technical person (e.g., the system administrator) whenever possible.

The interviews took place between July and November 2005. An overview of the cases in our study is shown in Table 2. As can be seen from this table, the organizations in our sample are considerably smaller than those in the study of West and Dedrick (2005).[3] In each organization, we have

Table 2. Overview of the organizations in our study

Name	Sector	Employees	Informants	Extent of adoption
OrganizationA	Audio, video, and telecommunications	11	2	moderate
OrganizationB	Machinery and equipment	749	2	extensive
OrganizationC	Telecommunications	1346	1	limited
OrganizationD	Publishing and printing	31	1	extensive
OrganizationE	Food products and beverages	204	2	moderate
OrganizationF	Research and development	152	2	extensive
OrganizationG	Information technology	583	1	moderate
OrganizationH	Chemicals	4423	1	moderate
OrganizationI	Education	3303	3	limited
OrganizationJ	Publishing and printing	12	1	extensive

conducted a single interview during which all informants in the organization were present. The interviews were semi-structured, and the format was revised after each interview to incorporate new findings (Benbasat et al., 1987). In the first part of the interview, informants were asked to freely discuss their reasons for adopting OSS. In the second part of the interview, we probed for specific factors that were found relevant in previous studies, as well as the informants' perceptions of the free and open source movements. Each interview lasted 45-90 minutes, was recorded and transcribed verbatim. In order to increase the validity of our findings, informants were sent a summary of the interview and were requested to suggest any improvements if necessary. Follow-up questions were asked by telephone or via e-mail. The transcripts were coded and then further analyzed using procedures to generate theory from qualitative data, as described in the literature (e.g., Benbasat et al., 1987; Eisenhardt, 1989; Dubé & Paré, 2003). Various data displays were used to visualize and further analyze the qualitative data (Miles & Huberman, 1994; Eisenhardt, 1989).

EMPIRICAL FINDINGS

The dominant attitude towards OSS in seven organizations in our sample was pragmatism. These organizations did not exhibit any of the tenets of the OSS ideology, or their decision makers considered how the advantages of OSS could be realized in their organization. Consequently, these organizations could be considered pragmatic (and mindful) in their decision making with respect to the adoption of OSS. The most commonly cited advantages—and reasons for the adoption—of OSS were *cost* and *reliability*. In general, decision makers tended to consider both proprietary and OSS alternatives, and based their decision on the cost and functionality offered by the various alternatives. Some organizations even explicitly mentioned that they made a pragmatic adoption decision. These seven organizations did not have a preference for using OSS over proprietary software, except OrganizationB where a slight preference for OSS was present. Although they would accept a minor workaround in order to be able to use OSS, this effort should be limited. Or, as expressed by an informant:

We are not going to program around something, because we really want to use that [open source] component. But if there is a little workaround, we will certainly take it.

The other six organizations were quite agnostic about using OSS. One informant in OrganizationF expressed this as:

[The fact that the software is open source] does not really matter for a company.

Some of the technical employees who served as informants in our study had a background in OSS. Although some indicated that they did suggest the use of OSS when appropriate, they did not try to force its use and remained pragmatic. Nevertheless, many OSS development and networking tools (e.g., Nagios, Eclipse, and Maven) were being used by the organizations in our sample.

The results obtained from these seven organizations are quite consistent with the results obtained by West and Dedrick (2005). On the other hand, we observed a different behavior in the three very small organizations in our sample (OrganizationA, OrganizationD, and OrganizationJ) consisting of less than 50 employees. In those organizations, we were able to detect several characteristics of ideological behavior. In the remainder of this section, we will discuss these three cases in more detail.

OrganizationA

OrganizationA specialized in telecommunication devices. It originally started as a research and development company. Initially, all projects within the organization aimed to gather knowledge and experience in order to develop the initial product. Developers were free in their decision making on which products to incorporate into the final product. Consequently, decision making was significantly influenced by the personal experience of developers.

Our informants indicated that at the time of the organization's founding, many employees—including the organization's founders and the CIO—shared the same background, were very familiar with Linux, and shared the philosophical ideas of the open source movement. These employees had a "firm conviction" in OSS:

The firm conviction was coming from a number of people who said: 'It must be [OSS], we do not want anything else!'…The choice for using OSS was…just a conviction, rather than the result of a comparative assessment.

As a result, most software that was used in the organization was OSS. During package selection, no objective evaluation of (proprietary) alternatives was performed. Although some proprietary software was used, this was either on demand of a customer, or the software was eventually replaced by an OSS alternative.

The choice for OSS at that time was primarily motivated by the lower or non-existing license cost, the fact that there was more confidence in OSS, and the fact that OSS provides access to the source code. Our informants however admitted that these reasons were influenced by the philosophical view towards OSS and that this view on OSS dominated the adoption decision. They were for example aware that using OSS includes additional costs (e.g., packaging and updates), which makes it less clear whether OSS really offers a cost advantage. Such considerations were however not taken into account at that time.

Another factor that has influenced the decision is the avoidance of vendor lock-in. The open source movement generally depicts Microsoft as their common "enemy." This feeling was also present in the organization at that time. Vendor lock-in with Microsoft was feared, partly due to negative experiences in the past. The adoption decision appeared to be anti-Microsoft oriented. As expressed by one informant:

If you mentioned Microsoft, things exploded!

The organization also initiated its own OSS project. It consisted of a Java virtual machine for embedded devices. This project was started to try to benefit from the OSS community model (cf. tenets 4–15). This project was in fact quite successful, and the organization took the role of project maintainer. In the course of time, the project became less interesting for the community (as the product further matured) and participation of the community declined. The software is however still used in the organization's products.

As illustrated, the choice for using OSS was quite ideological in the early years of the organization. Interesting to note is that over the years, several employees of the organization who were adherents to the open source movement, and who advocated the use of OSS, left the organization. As a result, the choice for OSS became much more pragmatic. Another factor that may have influenced this evolution is that the organization finished its software products, gradually became less of an R&D organization, and other goals such as efficiency started to become more important.

At the time of our study, a slight preference for OSS still existed. One informant stated:

Our choice will in the first place go to open source or Linux, but less fanatical than in the past.

Furthermore, the organization seemed to be less willing to take risks in using OSS, or to invest additional effort to get OSS working. This was expressed by an informant as:

I think we are looking rather quickly towards open source products. But if it looks that it will deliver us more worries than it yields advantages, we will not doubt to use a commercial product.

Hence, the organization will only consider using OSS if the product complies with the requirements. The "firm conviction" that was present in the organization has now faded away. The choice for OSS is now mainly based on the potential cost advantages.

Nevertheless, it appears that the organization still felt connected to the principles of the open source movement. When asked whether the organization contributed back any modifications they made to OSS, one informant appeared to feel guilty about not contributing:

…we did contribute quite little, rather naughty, isn't it?

He further noted that the organization tried to participate in OSS projects in other ways, for example by filling in bug reports or by participating in mailing lists (cf. tenets 10–15).

OrganizationD

OrganizationD was active in the publishing and printing sector. The organization had a single person responsible for decision making on IT, and had no internal IT staff. The organization used OSS on a variety of systems (i.e., one Internet gateway, two file servers, and one intranet server). The organization also had 3 LAMP (Linux–Apache–MySQL–PHP) servers, running custom-developed software for time registration. Finally, three desktops were equipped with the Linux operating system in the offices, and an additional 11 PCs function as terminals for the time registration system. The main reason for choosing OSS was to reduce vendor lock-in and maximize the freedom of the IT infrastructure. Consequently, the decision maker investigated OSS solutions without considering proprietary alternatives. Other reasons for using OSS were an increased control over the software, cost advantages, and an increased flexibility. These factors are consistent with the advantages proposed by the OSS community. We were able to detect a few additional ideological characteristics, although they were not that strong.

Our informant indicated that his extensive personal experience with Linux influenced his decision to start using OSS within the organization:

Following [new evolutions] is not enough: you try out software, and free software has the advantage that it is much easier to try out. And of course, since you have tried it yourself, it did influence the [organizational] decision.

His decision to start using OSS within the organization was also influenced by some negative experiences with proprietary software in the past (including vendor lock-in). For example, some proprietary application the organization was using contained a bug which the vendor refused to resolve. As a result, our informant tried to remain in full control over his IT infrastructure. He therefore wanted to maximize the degree of freedom in the IT infrastructure, not only by using open standards, but by using OSS as well: "I wanted to go a step further: not only by using open standards, but also by using open source applications to have full insurance" (cf. tenets 5–6). He felt that by having access to the source code of OSS, he had maximum control over his applications.

The organization was remarkably committed to its pursuit of freedom. This commitment has moved the organization to start its own OSS project, namely a time registration system for employees. Existing software either did not satisfy all requirements, or was too expensive and did not allow for customizing the software. Hence, the software needed to be custom developed. The decision maker did not want to become dependent on an external organization—not even on the external programmer who develops the software. Instead of performing in-house development or closing an escrow agreement, the organization has chosen a different path. The organization has hired a programmer from an external organization to develop the software, and our informant decided to release the software under an OSS license (the GPL) to ensure that the software would remain completely free (cf. tenet 5). This way, the organization aimed to remain in control over the application, avoid vendor lock-in, and be allowed to make modifications to the software at a later time. The software is being developed as a cooperation between our informant (who is mainly responsible for the analysis) and the paid external programmer. It was the intention of our informant to eventually share this application with other organizations in the same sector. He strongly valued the ability to cooperate with other organizations, and hoped that he would be able to leverage the OSS development model (cf. tenets 4–15) and to receive comments, bug fixes, and maybe even new code submissions.

Interestingly, he was the only informant in our sample who deliberately used the term *free software*.[4] He preferred this term since—in his experience—the term OSS is misused by some vendors to refer to software of which the source code is available, but whose license is still proprietary and does not offer the same freedom as OSS licenses. He felt that the Dutch term for *free software* did not suffer from the confusion in English, and that it better articulated the spirit of the open source movement (cf. tenet 5).

OrganizationJ

The most prominent form of ideological behavior was found in OrganizationJ. Our informant was the IT and business manager of the organization, who was the only one responsible for the IT infrastructure. No internal IT staff was present. The complete IT infrastructure of the organization was based on OSS. This included two important servers: an intranet server running ERP software and an Internet server running the e-commerce site of the organization. Recently, all desktops in the organization were migrated from MS Windows to Linux. The desktops consisted of lightweight terminals which booted from a server. All appli-

cations ran on the server, which placed very low demands on the desktop itself. All administration could be performed on the server. The desktops were running the XFCE desktop environment and OpenOffice.org was used as the office suite.

Our informant had a technical background and was an experienced programmer. In fact, he developed his own e-commerce application and was currently rewriting his own ERP software. His personal experience with Linux dates back from 1999. Based on this personal experience, he decided to migrate his Unix-based server to Linux when he was experiencing difficulties with that server.

Similar to our informant in OrganizationD, the IT manager wanted to remain in control of his IT infrastructure (cf. tenets 5–6). Consequently, he tried to make exclusive use of open standards. Moreover, he stated that he only considered using OSS (except for one PC running Microsoft Windows on which specific banking software was installed that is unavailable for Linux). He also did not want to pay for software, hence he did not use any of the commercial Linux distributions.

Similar to the other two organizations, our informant indicated that his organization had bad experiences with proprietary vendors in the past. In fact, when migrating the server that ran the ERP software, the organization faced huge switching costs when transferring the software from the Unix-based system (developed by a small company) to Linux. He was also suspicious of proprietary software, because it could contain hidden features. This prevented him from having total control over the software. OSS was believed to be more secure, thanks to the availability of the source code: "I think there are thousands, ten thousands or millions of people who use and study it, so I don't have to worry" (cf. tenets 4 and 7).

As a result, he had a rule that proprietary software should not be used under Linux. Proprietary software was simply not considered as an alternative during decision making. This non-pragmatic decision making can be illustrated with two examples. First, the organization recently acquired a new printer/copier. Although the manufacturer provided drivers for Linux, they were proprietary; and the source code of the drivers was not provided. Consequently, the drivers were not installed on the Linux desktops. This means that default Postscript and PCL drivers were used. If specific features would be required, the IT manager stated that he would rewrite the drivers, based on the Postscript definition. He motivated his choice as follows:

Nothing is installed from which the source code is not available: I need control.…[The manufacturer of the printer] will probably have no bad intentions, probably, but nowadays you never know.

Second, when the IT manager decided that the ERP software needed replacement, he reviewed some OSS alternatives. One of the reasons why Compiere was not properly examined as an alternative, was that it required the Oracle database server.[5]

The IT manager also started a small OSS project. It consisted of a Perl module to create OpenOffice.org documents. He also indicated that he valued the OSS development model. Two important advantages of this model were the peer review process (see supra) and that it offers more continuity. Although his ERP software was using a graphical library that was maintained by a single person, he was not afraid of becoming too dependent. If the maintainer would quit, our informant was convinced that other people would take over the project. Otherwise, he would still have access to the source code of the library and make any required changes himself (cf. tenets 5 and 14).

DISCUSSION

As can be gathered from our findings, ideological or pragmatic decision making is not a binary

variable. Instead, decision making will exhibit both ideological as well as pragmatic characteristics, which places the organization's decision making on a continuum between both extremes. In practice, most organizations clearly use a pragmatic decision-making process with respect to the use of OSS. Nevertheless, we were able to detect rather ideological decision making in three small organizations in our sample. The degree of ideology varied between these three cases. A summary of the ideological characteristics in the decision-making process of these organizations is shown in Table 3.

Identifying Ideology

There were clear distinctions between the seven organizations that we labeled "pragmatic" and the three we identified as "ideological." First, within the three latter organizations, there was a clear push behind—or favoritism towards—using OSS. This was caused by the fact that decision makers were adherents to the open source movement and wanted to use OSS as much as possible, or even exclusively. Their personal experience and background was a major factor in this decision. The other seven organizations did consider OSS as one of the alternatives, but would not give preferential treatment to OSS.

Second, the tenets of the OSS ideology were only present in the three organizations. Among

Table 3. Ideological characteristics in the decision making of organizations in our sample

OrganizationA:
• Employees, including the organization's founders, shared the philosophical and cultural views of the OSS movement.
• A strong anti-Microsoft sentiment was present.
• Vendor lock-in was feared.
• The organization started its own OSS project to benefit from the OSS development model.
• All software that was used had to be OSS.
• The adoption decision was based on a "firm conviction" in OSS, not on an objective evaluation of alternatives.
OrganizationD:
• The IT manager strives to maximize the freedom in the IT infrastructure by using open standards and OSS.
• Extensive personal experience of the IT manager with Linux influenced the organizational adoption decision.
• The organization started its own OSS project to ensure that the software would remain totally free.
• Driven to OSS by negative experiences (including vendor lock-in) with proprietary software in the past.
• The IT manager uses the term "free software."
OrganizationJ:
• The IT manager does not want to pay for software, including application software.
• The switch to Linux was influenced by personal experience with Linux.
• All software that was used had to be OSS.
• Proprietary printer drivers were not used, even if this means that a work-around must be devised.
• Commercial software is not trusted because the source code is not available.
• Driven to OSS by negative experiences (including vendor lock-in) with commercial software in the past.
• The OSS development model is valued, because thousands of developers are reading the source code, correcting bugs, and ensuring the continuity of the project.
• The complete IT infrastructure was migrated to OSS.
• The IT manager started his own OSS project.

the tenets that were most prominently present were software freedom (tenet 5), information freedom (tenet 6), and cooperation (tenet 14).[6] These tenets are indeed central to the OSS ideology. The other seven organizations were rather agnostic about the values and beliefs of the open source movement and considered the OSS character irrelevant during decision making.

Third, several of the factors that influenced the adoption decision are consistent with the advantages put forward by the open source movement. Evidently, this is not sufficient to claim that these organizations shared the OSS ideology. However, there are indications (particularly in Organization A and Organization J) that the perceptions with respect to these adoption factors are influenced by the belief in the OSS ideology, and that their relevancy in the organization-specific environment were not or insufficiently evaluated. This indicates mindless decision making.

Finally, these three organizations were the only ones in our sample that initiated their own OSS projects. Organization A and Organization D clearly indicated that by starting their own OSS projects they wanted to try to leverage the OSS community model. This indicates a belief in the underlying principles of the open source movement (cf. tenets 10–15). If organizations would not be convinced of the advantages of the OSS development model, it seems likely that they would not initiate an OSS project and they would simply develop the software in-house. Nevertheless, principles such as sharing (tenet 10) and cooperation (tenet 14) were deemed quite important by the three organizations.

The previous four points demonstrate that the three organizations discussed in this article exhibited some form of ideological behavior. It is however not trivial to identify ideological tenets in organizations, since the ideas and beliefs of the OSS ideology are not explicitly formulated, as is often the case with ideologies (Hamilton, 1987). A second difficulty is that the presence of one of these characteristics by itself does not automatically lead to ideological decision making. A good example is the avoidance of vendor lock-in. All three organizations indicated having had bad experiences with proprietary vendors in the past and wished to minimize vendor lock-in. The desire to avoid vendor lock-in can be a pragmatic reason for choosing OSS. It may however also lead to a situation in which the decision maker—based on negative experiences with some vendors in the past—only wants to use OSS without considering proprietary alternatives, leading to an ideological position towards OSS. Similarly, the list of characteristics in Table 3 is not exhaustive, and there may be other indicators of ideological behavior. A third issue is that there may be "instances where actors, genuinely or otherwise, do not interpret their behavior in terms of any commitment to a set of beliefs but as simply pragmatic, but where it is clear to the observer that it is, in fact, in conformity with such a set of beliefs" (Hamilton, 1987, p. 21). Nevertheless, the evidence presented in this article and the impression of the decision makers obtained during the interview allowed us to identify ideological characteristics in the decision making of these three organizations. These characteristics had a clear impact on the adoption decision on OSS, resulting in a strong favoritism towards OSS. The attitude in these three organizations was fundamentally different from the other seven organizations in our sample.

Limitations

This study has a number of limitations. First, we used a qualitative approach consisting of 10 case studies. Although we have found that small organizations may engage in ideological decision making, a large-scale quantitative study could provide more insight into the generalizability of this result.

Second, we only included organizations that have adopted OSS. Future research may provide more insight into the attitudes of non-adopters. We can make a meaningful distinction between

two groups of non-adopters. On the one hand, there can be organizations that have considered using OSS, but decided not to adopt. The experiences of these organizations may provide more insight into the main drawbacks of using OSS. On the other hand, there are organizations that did not consider OSS as one of the alternatives. Such organizations may have negative perceptions towards OSS and did not further investigate them. For example, organizations may be convinced that OSS costs more in maintenance or is unreliable. Similarly, organizations may also have unverified ideas with respect to proprietary software. They may believe that using proprietary software is less expensive or may place more trust in a closed, proprietary software model. In the most extreme case, organizations may even only consider using software from one specific vendor. In either case, decision making will not be mindful, as not all alternatives are being considered.

Another interesting avenue for future research is to investigate whether decision making on OSS will become less ideological. Since the adoption of OSS is still a relatively recent phenomenon, less information is available on OSS than on proprietary software. It can be expected that as time passes, more information on an innovation becomes available, and decision makers will be able to make better informed choices. On the other hand, Swanson and Ramiller (2004) point out that later adoption can also be driven by diffusion itself, making later adoption not necessarily more mindful than early adoption.

A final topic for further investigation concerns situations in which the decision to start using OSS is triggered by the mere availability of OSS, rather than a concrete problem situation that gives rise to a search, evaluation, and decision-making process. This process resembles the *garbage can model* of decision making (Cohen, March, & Olsen, 1972). Hence, future research could investigate the applicability of this theory in situations in which decision makers share the OSS ideology.

CONCLUSION

The contribution of this article is that we were able to identify ideological characteristics in the decision making on OSS in very small organizations. This result further elaborates on the study of West and Dedrick (2005), who did not detect such behavior in their sample. We argue that while medium to large businesses are likely to be pragmatic in their decision making, the influence of ideological beliefs should not be completely disregarded in small organizations.

Although a minority of organizations in our sample has exhibited ideological behavior, it is remarkable that all three very small organizations in our sample—with a single decision maker—did to some degree. If that decision maker can be considered an open source advocate—which was definitely the case in OrganizationA and OrganizationJ—it is more likely that personal beliefs and values of the decision maker have an impact on the final decision making. Hence, the adoption decision with respect to OSS is more likely to be ideological. This is consistent with the observation of Fiol and Connor (2003) who argue that mindlessness in combination with the absence of formal procedures will further enable mindlessness. In larger organizations, decision making is more likely to be pragmatic, since there are more decision makers and procedures involved in the OSS adoption decision.[7] Ideological decision making is however not necessarily a static phenomenon. Since it appears that ideological decision making is closely related to a single decision maker, the situation may change if that person leaves the organization, or if other decision makers join the organization. This could be observed in OrganizationA.

The definition of ideology we have used in this article is non-judgmental. Consequently, we do not want to make any claims with regard to whether the organizations have made a wrong decision in choosing for OSS. We have found no evidence to suggest that the decision has had a negative impact

on the organizations. In fact, Organization A actually seemed to be able to innovate by using OSS and proved to be quite successful. On the other hand, it could be established that Organization A (at the time of founding) and Organization J were not sufficiently mindful in their decision. These organizations only considered using OSS and did not properly investigate alternatives. Such mindless behavior always entails the risk that the organization does not properly reflect on whether the innovation is suitable within the organization, resulting in a less-than-optimal solution for the organization (Swanson & Ramiller, 2004). A mindful organization that adopts OSS should not take the claims proposed by the OSS ideology for granted. Instead, it should investigate the implications of using OSS in the organization-specific environment. This is important since this situational context can be complex, rendering some claims irrelevant for the organization.

Swanson and Ramiller (2004) however point out that notwithstanding the risks, mindless decision making can have its merits for organizations. This can be the case when the rewards are likely to outweigh the risks, or when time limitations do not allow for a thorough decision-making process. Hence, mindless decision making can be a valid strategy for routine decisions and does not necessarily imply ideological decision making. However, we were able to exclude this possibility in the three small organizations in our sample by investigating the background of the decision-making process. In all three organizations, the adoption of OSS constituted an important change that concerned the replacement of existing proprietary software or the use of a new type of software. Therefore, no similar evaluation of OSS was previously undertaken, and decision making was indeed ideological.

REFERENCES

Abrahamson, E. (1991). Managerial fads and fashions: The diffusion and refection of innovations. *Academy of Management Review, 16*(3), 586–612.

Benbasat, I., Goldstein, D.K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly, 11*(3), 368–386.

Bergquist, M., & Ljungberg, J. (2001). The power of gifts: Organizing social relationships in open source communities. *Information Systems Journal, 11*(4), 305–315.

Butler, B.S., & Gray, P.H. (2006). Reliability, mindfulness, and information systems. *MIS Quarterly, 30*(2), 211–224.

Cohen, M.D., March, J.G., & Olsen, J.P. (1972). A garbage can model of organizational choice. *Administrative Science Quarterly, 17*(1), 1–25.

Dedrick, J., & West, J. (2003). Why firms adopt open source platforms: A grounded theory of innovation and standards adoption. In J.L. King & K. Lyytinen (Eds.), *Proceedings of the Workshop on Standard Making: A Critical Research Frontier for Information Systems* (pp. 236–257), Seattle, WA.

Dubé, L., & Paré, G. (2003). Rigor in information systems positivist case research: Current practices, trends, and recommendations. *MIS Quarterly, 27*(4), 597–635.

Eisenhardt, K.M. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532–550.

Fiol, C.M., & Connor, O.J. (2003). Waking up! Mindfulness in the face of bandwagons. *Academy of Management Review, 28*(1), 54–70.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly, 30*(3), 587–598.

Geuss, R. (1994). Ideology. In T. Eagleton (Ed.), *Ideology* (pp. 260–278). Essex, UK: Longman Group.

Hamilton, M.B. (1987). The elements of the concept of ideology. *Political Studies, 35*(1), 18–38.

Hansen, M.T., & Haas, M.R. (2001). Competing for attention in knowledge markets: Electronic document dissemination in a management consulting company. *Administrative Science Quarterly, 46*(1), 1–28.

Harrison, D.A., Mykytyn, P.P. Jr., & Riemenschneider, C.K. (1997). Executive decisions about adoption of information technology in small business: Theory and empirical tests. *Information Systems Research, 8*(2), 171–195.

Ljungberg, J. (2000). Open source movements as a model for organizing. *European Journal of Information Systems, 9*(4), 208–216.

Lundell, B., Lings, B., & Lindqvist, E. (2006). Perceptions and uptake of open source in Swedish organizations. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, & G. Succi (Eds.), *IFIP international federation for information processing: Volume 203 open source systems* (pp. 155–163). Boston: Springer.

Markus, M.L., Manville, B., & Agres, C.E. (2000). What makes a virtual organization work? *Sloan Management Review, 42*(1), 13–26.

Miles, M.B., & Huberman, A.M. (1994). *Qualitative data analysis: An expanded sourcebook* (2nd ed.). Thousand Oaks, CA: Sage.

Phillips, L.W. (1981). Assessing measurement error in key informant reports: A methodological note on organizational analysis in marketing. *Journal of Marketing Research, 18*(4), 395–415.

Riemenschneider, C.K., Harrison, D.A. & Mykytyn, P.P. Jr. (2003). Understanding IT adoption decisions in small business: Integrating current theories. *Information & Management, 40*(4), 269–285.

Sternberg, R.J. (2000). Images of mindfulness. *Journal of Social Issues, 56*(1), 11–26.

Stewart, K.J., & Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly, 30*(2), 291–314.

Swanson, E.B., & Ramiller, N.C. (2004). Innovating mindfully with information technology. *MIS Quarterly, 28*(4), 553–583.

Tushman, M.L., & Scanlan, T.J. (1981). Characteristics and external orientations of boundary spanning individuals. *Academy of Management Journal, 24*(1), 83–98.

Weick, K.E., Sutcliffe, K.M., & Obstfeld, D. (1999). Organizing for high reliability: Processes of collective mindfulness. In R.I. Sutton & B.M. Staw (Eds.), *Research in organizational behavior* (vol. 21, pp. 81–123). Greenwich, CT: JAI Press.

West, J., & Dedrick, J. (2005). The effect of computerization movements upon organizational adoption of open source. *Proceedings of the Social Informatics Workshop: Extending the Contributions of Professor Rob Kling to the Analysis of Computerization Movements,* Irvine, CA.

Yin, R.K. (2003). *Case study research: Design and methods* (3rd ed.). Newbury Park, CA: Sage.

ENDNOTES

1 These organizations had on average 41,885 employees (25,529 when only counting the unit studied in the organization).

2 The term "mindless" generally has a pejorative meaning, such as "unintelligent." In academic literature however, the term is used to refer to automatic or inattentive behavior (e.g., Swanson & Ramiller, 2004;

Fiol & Connor, 2003; Butler & Gray, 2006; Sternberg, 2000). We use the term "mindless" in the second sense. Hence, we do not wish to imply any negative connotations.

[3] The organizations in our case studies have on average 1,081 employees.

[4] Actually, the Dutch equivalent was used, namely "*vrije* software," which is similar in meaning as the French term *libre* software and refers to "freedom" rather than "free of charge."

[5] Other reasons were that it used Java (which the IT manager did not like very much), and the fact that he preferred using custom-developed software that fits his business.

[6] This may indicate that these organizations preferred to cooperate with other organizations within the same industry in order to extend their own capabilities, rather than to outsource development to an external firm.

[7] On the other hand, Fiol and Connor (2003) have noted that formal procedures may also lead to mindlessness (i.e., when decision makers follow procedures without critically considering them).

This work was previously published in Journal of Database Management, Vol. 19, Issue 2, edited by K. Siau, pp. 58-72, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.5
Volunteers in Large Libre Software Projects:
A Quantitative Analysis Over Time

Martin Michlmayr
University of Cambridge, UK

Gregorio Robles
Universidad Rey Juan Carlos, Spain

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain

ABSTRACT

Most libre (free, open source) software projects rely on the work of volunteers. Therefore, attracting people who contribute their time and technical skills is of paramount importance, both in technical and economic terms. This reliance on volunteers leads to some fundamental management challenges: Volunteer contributions are inherently difficult to predict, plan, and manage, especially in the case of large projects. In this chapter we present an analysis of the evolution over time of the human resources in large libre software projects, using the Debian project, one of the largest and most complex libre software projects based mainly in voluntary work, as a case study. We have performed a quantitative investigation of data corresponding to roughly seven years, studying how volunteer involvement has affected the software released by the project, and the developer community itself.

INTRODUCTION

Volunteer contributions are the basis of most libre[1] software projects. However, the characteristics, and the way of working of volunteers, can be quite different from those of employees who are the main force behind traditional software development. Volunteers can contribute with the amount of effort they want, can commit for the time period they consider convenient, and can devote their time to the tasks they may prefer, if the context of the project makes that possible (Michlmayr & Hill, 2003). But even in this apparently difficult environment, many libre software projects have produced systems with enough quality and functionality to gain significant popularity. Therefore,

the fairly unstructured collaboration of volunteers has been demonstrated as a viable software development strategy, even if it is associated with certain challenges related to project management and quality (Michlmayr, 2004). In this chapter we explore how these voluntary contributions evolve over time in one of the largest libre software projects, Debian.

For our purposes, we will define volunteers as those who collaborate in libre software projects in their spare time, not profiting economically in a direct way from their effort. Volunteers can be professionals related to information technologies, but in that case their activity in the libre software project is not done as a part of their professional activity. Although the vast majority of participants in libre software projects comply with our definition, it is important to note that there are also non-volunteers, that is, paid people (normally hired or contracted), who produce libre software. German has studied paid employees from various companies in the GNOME project (German, 2004). He notes that they are usually responsible for less attractive tasks, such as project design and coordination, testing, documentation, and bug fixing. Also, "[m]ost of the paid developers in GNOME were, at some point, volunteers. Essentially for the volunteers, their hobby became their job."

The involvement of volunteers, of course, raises new economic and business model issues that have to be taken into account in commercial strategies around libre software. Collaboration from volunteers is difficult to predict, but if it is given, it may add value to a software system in very economic terms for a software company.

The structure of this chapter will be as follows. In the second section we discuss the nature and, in particular, the tasks performed by volunteers, paying special attention to those who contribute to Debian, the case study investigated in this chapter. Following this section, a set of research questions regarding volunteer participation will be raised. The primary goal of this chapter is to answer these questions based on quantitative data. The methodology for retrieving the quantitative data used in this study is first given. In this section, we also propose a number of measures that will allow us to answer the questions. The results we have obtained as part of this study will be presented and commented on in depth for the Debian project. Finally, conclusions, applicability of the methodology, and further research will be discussed.

THE DEBIAN PROJECT AND ITS VOLUNTEERS

Debian is an operating system completely based on libre software (Monga, 2004; O'Mahony, 2003). It includes a large number of applications, such as the GNU tools and Mozilla, and the system is known for its solid integration of different software components. Debian's most popular distribution, Debian GNU/Linux, is based on the Linux kernel. Ports to other kernels, such as Hurd and FreeBSD, are in development.

One of the main characteristics of the Debian distribution is that during the whole life of the project it has been maintained by a group of volunteers, which has grown to a substantial number. These individuals devote their own time and technical skills to the creation and integration of software packages, trying to supply users with a robust system which provides a lot of functionality and technical features.

Following our definition of volunteers, all maintainers in Debian are volunteers. Some employers of people who act as Debian maintainers in their spare time permit their staff to devote some of their time to Debian during work hours. Nevertheless, the majority of work by most Debian maintainers is performed in their spare time. In contrast to some projects, such as the Linux kernel and GNOME, there are no Debian maintainers who are paid to work on the system fulltime, even though a number of organizations have a com-

mercial interest in Debian and contribute varying degrees of manpower (and other resources) to the project. For example, a number of regions in Spain have their own operating systems based on Debian; HP made Debian more suitable for large telecom customers, and Credativ provides commercial services for Debian and similar systems.

There are several tasks that volunteers can do in Debian: maintaining software packages, supporting the server infrastructure, developing Debian-specific software, for instance, the installation routine and package management tool, translating documentation and Web pages, and so forth. From all these tasks, we will focus in this chapter on package maintainers, whose task it is to take existing libre software packages and to create a ready-to-install Debian package. Debian maintainers are also called Debian developers, although their task is really not to develop software but to take already developed software for the creation of a Debian package. This, of course, does not mean that a Debian maintainer may not develop and maintain software, but this is not usually the case: the original author (or developer), known as the 'upstream' developer, and the Debian maintainer, are usually, but not necessarily, not the same person.

Besides its voluntary nature, the Debian project is unique among libre software projects because of its social contract (Debian Social Contract, 2006). This document contains not only the primary goals of the Debian project, but also makes several promises to its users. Additionally, there are a number of documents Debian maintainers have to follow in order to assure quality, stability, and security of the resulting distribution. In particular, Debian's Policy document ensures that the large number of volunteers working independently will produce a well-integrated system rather than merely an aggregation of software packages which do not play together very well (Garzarelli & Galoppini, 2003).

There has recently been some interest in studying how the voluntary status of Debian members affects the quality of the resulting product. Managing volunteer contributors is associated with certain problems that "traditional" software development usually does not confront (Michlmayr & Hill, 2003). It is known that there are some intrinsic problems when the development process is carried out in a distributed fashion (Herbsleb et al., 2001). The situation in Debian and similar projects is even more complex because the development process is not only distributed but also largely based on volunteers. This can lead to certain challenges, such as the unpredictability of the level of their involvement (Michlmayr, 2004).

To some degree, the volatility of voluntary contributors can be limited by the introduction of more redundancy, such as the creation of maintainer teams. The creation of teams and committees for specific purposes, such as management, or for complex tasks has been already reported in other libre software projects (as for instance, German's work on the GNOME project [2004]).

RESEARCH OBJECTIVES AND GOALS

Related research has been very active in studying the static picture of a libre software project community over the last years, as can be seen from studies performed on Apache and Mozilla (Mockus et al., 2002), FreeBSD (Dinh-Trong & Bieman, 2005) or GNOME (Koch & Schneider, 2002), leading to models that discuss onion-like structures of libre software projects (Crowston & Howison, 2005). The main goal of this chapter is to introduce the time axis in these kinds of studies, focusing most notably on the contribution of volunteers.

We have therefore set up a list of research questions which we would like to answer for several large libre software projects. In the case of Debian, we have additional information that permits us to link the work done by a volunteer with a piece

of software, so we can, for example, study what has happened to packages from volunteers who have left the project. In the following, the set of questions that we are raising will be answered in detail for the Debian project.

The specific questions we aim to answer with this chapter are the following:

a. **How many volunteers does the project have, and how does this number change over time?** This will provide us with some basic data, useful when working with subsequent questions. When we started the study, we expected a steady increase of volunteers over time, as it is already known that the number of packages included in the system has been growing in that way (Gonzalez-Barahona et al., 2004). In addition, we will try to find out if the work ratio (measured as activity or output per developer) has increased over time or not.

b. **How many volunteers from previous releases remain active?** We want to measure the volatility of the volunteers in large libre software projects. That is, do volunteers join the project and work on it for short periods of time, or, on the contrary, do they stay for many years? Specifically, we want to calculate the half-life of contributors of the project. The half-life is defined as the time required for a certain population of maintainers to fall to half of its initial size. This figure could be easily compared with other libre software projects and, of course, with statistics from companies from software and other industries.

c. **What is the activity of volunteers who remain in subsequent releases?** Answering this question will allow us to know if "older" volunteers strengthen their contributions as time passes, contributing more to the project, or whether they become less active. There are two possible hypotheses one could propose. On the one hand, those volunteers who have been involved for a long time may be very experienced and therefore more efficient in their work than less experienced developers. On the other hand, young developers may have more time or energy to devote to the project and therefore contribute more. Both theories are possible and mutually compatible.

d. **What happens to packages maintained by volunteers who leave the project?** Our intention is to see if we can find a regeneration process in libre software projects that allows them to survive the loss of some of their human resources.
Since Debian maintainers are volunteers, they may quit the project at almost any time, leaving their packages unmaintained. There are two possible outcomes regarding the future of those packages: First, they can be taken over (adopted) by another maintainer. Alternatively, if nobody is interested in adopting them, they will eventually be removed from the archive and excluded from future stable releases. Such removals of unmaintained packages are part of Debian's Quality Assurance effort. Our intention was to know how this inherent characteristic of the voluntary contributors affects Debian, and how this is damped down by other (possibly new) maintainers.

e. **Are more "important" and commonly used packages maintained by more experienced maintainers?** It can be interesting to know whether packages which are considered crucial for the functioning of the system are maintained primarily by volunteers who have more experience. For this, we have considered the most used packages as the targets of the study. We have defined as crucial packages those which are usually installed on every system, as, for instance, the base system which in the case of the Debian GNU/Linux operating system is composed, among others, of the Linux kernel

and the GNU tools. This does not necessarily mean, of course, that crucial packages are more difficult to maintain than other packages, but as they are used by all users of the system and the rest of the software heavily depends on their proper functioning, these packages have to be maintained with special care. Data about the importance of each package will be obtained from the Debian Popularity Contest[2] that tracks how many people have installed a given package.

METHODOLOGY AND SOURCES OF DATA

Debian consists of four parallel versions (stable, testing, unstable, experimental) which can be downloaded from the Internet. The focus of this study is on the stable versions from Debian 2.0 onwards, up to version 3.1, which provide good snapshots of the history of the distribution. These releases comprise a period of time from July 1998 to June 2005. There have been releases of Debian before 1998 (Lameter, 2002), but they have not been taken into consideration for this study since the sources of data we have used in this study were not available for them. For each release, we have retrieved the corresponding Sources.gz file (see next section) from the Debian archive. We have then extracted information about packages and their maintainers from this file and stored the results in a database. After that, we performed some semi-automatic cleaning and massaging of the data that will be explained in more detail below. Final results were obtained through queries to the database, and correlations that have been implemented by another set of scripts.[3]

In addition to the analysis of official releases, we have enriched the findings by additionally taking a more fine-grained data source into account. While releases are only done occasionally, in the case of Debian with years between releases, uploads to the Debian archive are done on a continuous basis. We have analyzed the activity related to these uploads to clarify some of the findings of the paper on which this chapter is based (Robles et al., 2005). The estimations of the size of the releases have been done using a software that counts source lines of code and avoids double-counting the code included in various packages (this methodology is described in detail (Gonzalez-Barahona et al., 2001)), using previously published data (Gonzalez-Barahona et al., 2004), except for release 3.1, which was calculated specifically for this study. The data related to the importance of packages has been retrieved from the Debian Popularity Contest (see section on this topic).

Debian Sources File

Since version 2.0, the Debian repository contains a Sources.gz file for each release, listing information about every source package in it. Every source package contains the name and version, list of binary packages built from it, name and e-mail address of the maintainer, and some other information which is not relevant for this study. As an example, see an excerpt of the entry for the mozilla source package in Debian 2.2 below. It can be seen how it corresponds to version M18-3, provides four binary packages, and is maintained by Frank Belew.

```
[...]
Package: mozilla
Binary: mozilla, mozilla-dev, libnspr4, libnspr4-dev
Version: M18-3
Priority: optional
Section: web
Maintainer: Frank Belew (Myth) <frb@debian.org>
Architecture: any
Standards-Version: 3.2.0
Format: 1.0
Directory: dists/potato/main/source/web
Files: 57ee230b97ccc69444ccccd0bc66908a 719
mozilla_M18-3.dsc
```

532934635ad426255036ee070bad03c8 28642415 mozilla_M18.orig.tar.gz

3adf83de7e74bf940ee02c0deca20372 18277 mozilla_M18-3.diff.gz

[...]

Debian Popularity Contest

The Debian Popularity Contest is an attempt to map the usage of Debian packages. Its main goal is to know what software packages are actually installed and used. Information from the Popularity Contest is used by Debian, for example, to decide which software to put on the first CD.

The system functions as follows: Debian users may install the popcon package, which sends a message every week with the list of packages installed on the machine as well as the access time of some files which may give hints regarding the last usage of these packages. Of course, privacy issues are considered in a number of ways: Upon installation, users are explicitly asked if they want to send this information to Debian, and the server which collects the data anonymizes it as much as possible.

The resulting statistical information of all users participating in this scheme is publicly available on the Web site of the project. For every package, it includes the number of machines on which it is installed (inst), the number of machines which make regular use of that package (vote), the number of machines with old versions of the package (old), the number of recent updates (recent), the number of machines where not enough information is available (no-file), and the maintainer of the package. Below is an excerpt of the available data, in this case the top ten packages ordered by installations as of December 4, 2004. The first 66 packages are installed on all machines, with 6881 installations.

Debian Developer Database

From June 1999 onwards, Debian has held a database (http://db.debian.org) with data related to members of the project. Some information, such as the full name and user name, can be retrieved publicly through the Internet. This database also contains information about the digital keys used by a developer. Debian makes use of digital signatures through the use of the tools PGP (Pretty Good Privacy) and GPG (GNU Privacy Guard) to approve uploads to their software archive. The use of digital signatures provides two guarantees. First, the signature will show that the package comes from a trusted source, that is, from an of-

Table 1. Debian Popularity Contest statistics

rank	name	inst	vote	old	recent	no-files	(maintainer)
1	adduser	6881	6471	94	316	0	(Adduser Development)
2	debianutils	6881	6517	50	314	0	(Clint Adams)
3	diff	6881	6425	261	195	0	(Santiago Vila)
4	e2fsprogs	6881	5448	825	608	0	(Theodore Y. Ts'o)
5	findutils	6881	6449	233	199	0	(Andreas Metzler)
6	grep	6881	6436	126	319	0	(Ryan M. Golbeck)
7	gzip	6881	6558	245	78	0	(Bdale Garbee)
8	hostname	6881	6112	715	54	0	(Graham Wilson)
9	login	6881	6407	56	418	0	(Karl Ramm)
10	ncurses-base	6881	56	143	6	6676	(Daniel Jacobowitz)

ficial Debian developer whose PGP or GPG key is stored in the Debian keyring and this developer database. Second, by verifying the signature on the package, it can be ensured that the package has not been tampered with during the process of uploading it to the Debian archive.

Package Uploads

While the main data source of this chapter is the Sources.gz files from the last official Debian releases, we have additionally taken uploads to the archive into account to answer some of the research questions with more detail. As mentioned above, Debian's archive is separated into various branches. The official releases are known as the stable branch, whereas Debian's development tree is known as "unstable." Even though it is said that the development tree is usually fairly stable, it is where major development occurs and as such, major bugs are introduced from time to time.

When an upload is made to unstable, a summary of the changes is automatically sent to a mailing list known as "debian-devel-changes." By extracting data from the archives of this list, the uploads made to the Debian archive over the last few years can be studied. The archive of this mailing list starts towards the end of August 1997. Because August is not complete, we use the data starting with September, since having data for full months allows for better comparisons. We then divided the whole time period into periods of three months each, leading to 34 periods which cover 8.5 years (102 months), from September 1997 leading up to February 2006. In total, slightly more than 181,500 uploads have been observed in this period, leading to a mean number of uploads of around 1,800 each month.

The method used to extract this information is as follows: first, the archives of the "debian-devel-changes" mailing list are downloaded from Debian. These are then parsed, leading to one message for each upload. These messages are signed by the developer's PGP or GPG key as described above. The information from these digital signatures is then used to map each upload to a unique user id corresponding to the developer who made the upload. The mapping from PGP or GPG key to user name is obtained from Debian's developer database and missing entries for old developers are manually supplemented. After this information is obtained, information about the developer (user name) and the date of each upload is stored in a database together with the message id from the posting stored in the archive.

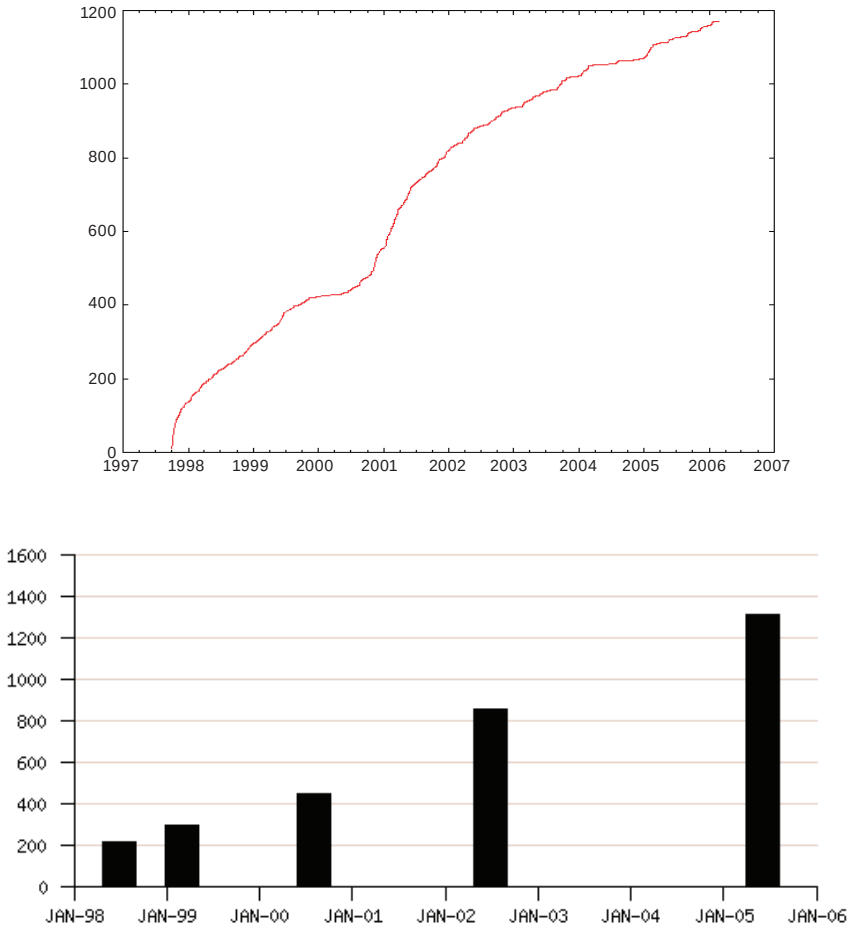
The extraction of this data leads to fairly precise results but there are some limiting factors. First of all, it is important to take into account that uploads are not a measure of effort. We use the data as an indication of activity of a developer but the information is not rich enough to give specific information about how much effort was involved with a specific upload. Second, for the last few years Debian had the concept of "sponsorship," whereby an official Debian developer would upload a package created by a prospective developer (who is not part of Debian yet and whose GPG key is therefore not recognized). In such cases, the main effort was done by the prospective developer but the signature shows the name of the official Debian developer. Since we are concerned with activity of developers and not with effort, this is not an obstacle but it has to be considered during the interpretation of the data. Finally, while the archive software used by Debian now sends automatic notifications of new uploads to the "debian-devel-changes" list, this was done manually in the past. Therefore, data from the past can be slightly unreliable. For example, we observed a number of messages which were not signed by PGP or GPG. Out of about 185,000, we have only detected about three thousand uploads for which no information about the developer could be extracted automatically. We believe that these are not significant and that data from uploads greatly enriches the findings from studies using the Source.gz file from official Debian releases.

EVOLUTION OF THE NUMBER OF DEBIAN MAINTAINERS

The information of the evolution of the number of Debian maintainers will provide us with some basic data useful when working with subsequent research questions. When we started the study, we expected a steady increase of maintainers over time, as it is already known that the number of packages included in the system has been growing linearly (Gonzalez-Barahona et al., 2004). In fact, we expected the packages-to-maintainer ratio to be nearly constant, since it seems reasonable to consider that volunteers devote similar amounts of effort over time, which would lead to a constant number of packages per maintainer.

Figure 1 shows on the left side the evolution of the number of Debian maintainers for the latest five stable releases. As we have expected, the number of Debian individual maintainers has been growing over time. Debian 2.0 (July 1998) was put together by 216 individual maintainers, while the number of maintainers for later releases are 859 for 3.0 (July 2002) and 1,314 for 3.1 (June 2005). This shows a growth of about 35 percent every year. The right side of the figure shows the cumulative

Figure 1. Number of maintainers over time. The left chart is based on Debian releases while the right one is based on continuous uploads.



number of developers who have made uploads to Debian (starting as of September 1997 and finishing with February 2006). This chart gives more precise information as to the growth over time but it does not include some information which the other chart captures. As mentioned above, only official Debian developers are considered and therefore the numbers are lower than in the chart on the left side which considers all maintainers of packages in Debian, regardless of their official status. The more detailed information the chart on the right conveys is very interesting, though. A remarkable stagnation of the growth can be observed. This is because the New Maintainer process, Debian's admission process, was stopped for several months at the end of 1999. The growth continues in the middle of 2000 and, interestingly enough, the pause in the admission of the developers did not have any significant effect on the overall growth of the project.

Based on the data from official releases, we have conducted a small statistical analysis; the results are shown in Table 2. The ratio of packages per maintainer (see column "Pkg/Maint") grows over time, contrary to our initial hypothesis. The growth of packages is actually bigger than that of volunteers who contribute to the project. There are some possible explanations for this finding. First, it is possible that improvements of the development tools or in the practices employed have led to an increase in the efficiency of developers. Second, due to increased interest in libre software, the development speed in general has accelerated and volunteers are more committed.

Interesting enough, the median does not vary (with the exception of Debian 3.0) over time in these last years. Half of the maintainer population does not have more than three packages to maintain. Furthermore, the mode shows that the most frequent situation is a maintainer who is in charge of one package. In brackets we can find the number of developers who actually maintain only one package, which is around one forth of the total population of Debian maintainers.

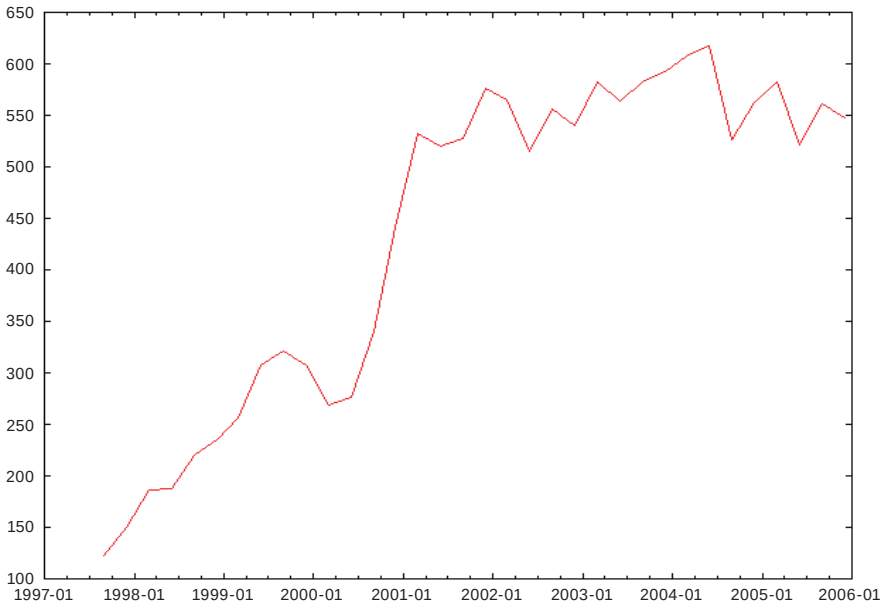
The next three values (the standard deviation, the Gini coefficient,[4] and the maximum number of packages maintained by a single maintainer) strengthen the idea that the distribution of work tends to be distributed in a more unequal way, with a small number of maintainers maintaining more and more packages while the number of packages the vast majority is in charge of does not change much. Compared to other libre software applications and, in general, to other studies which have looked at the distribution of work in libre software projects (Ghosh &d Prakash, 2000; Koch & Schneider, 2002; Hunt & Johnson, 2002; Mockus et al., 2002; Ghosh et al., 2002), we can see that, unlike other projects, Debian is far away from a Pareto distribution. In terms of the Gini coefficient, Debian shows values from roughly 0.5 to 0.6 while studies of the activity in CVS repositories of other projects have found Gini to be in the range between 0.7 and 0.9 (Robles, 2006).

Finally, in Figure 2 we see the number of individual developers making uploads to the Debian

Table 2. Statistical analysis of the growth in number of Debian maintainers

Date	Release	Maint	Packages	Pkg/Maint	Median	Mode	Std. Dev	Gini	Max
Jul 98	2.0	216	1,101	5.1	3	1 (52)	5.8	0.492	50
Mar 99	2.1	296	1,559	5.3	3	1 (76)	6.5	0.521	55
Aug 00	2.2	453	2,601	5.7	3	1 (122)	7.4	0.535	69
Jul 02	3.0	859	5,119	6.0	4	1 (208)	8.2	0.539	79
Jun 05	3.1	1,314	7,989	6.1	3	1 (386)	9.1	0.577	127

Figure 2. Mean number of developers making uploads per each three month period



archive for each three month period. A significant growth in the number of contributors can be seen in the first few years of the observed period. Since roughly the beginning of 2001, a fairly constant number of individuals makes contributions to Debian. There are about 550 unique contributors, even if they change over time.

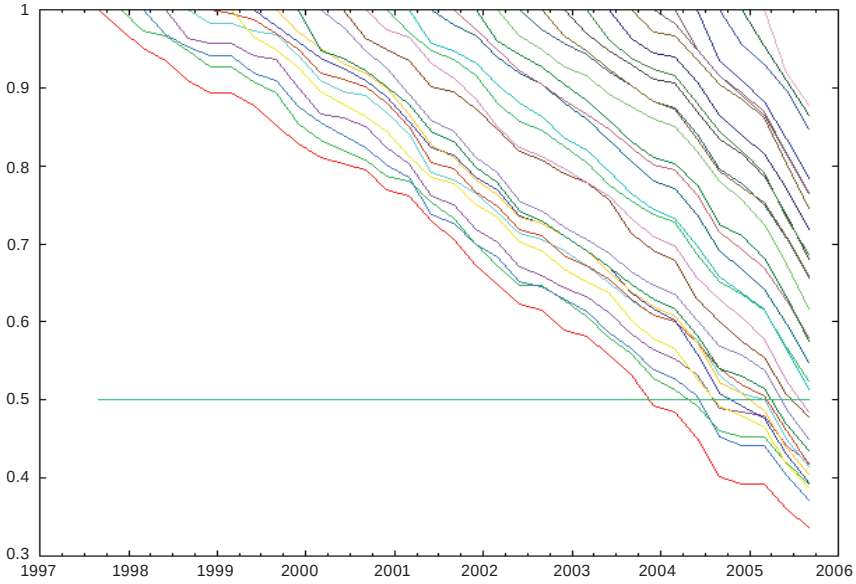
The first column gives the date of the release specified in the second one. "Maint" is the number of maintainers that maintain at least a package, "Packages" the number of total packages for that release, "Pkg/Maint," the mean number of packages per maintainer, "Median" the median number of packages, "Mode" gives the most frequent contribution in number of packages and in brackets the number of maintainers who contribute to it, "Std. Dev," the standard deviation of our sample," "Gini" the Gini coefficient, and "Max" the maximum number of packages that a unique maintainer is responsible for.

TRACKING REMAINING DEBIAN MAINTAINERS

At the time of the release of Debian 2.0 in July 1998 there were 216 voluntary developers contributing to Debian. We have studied how the involvement of these 216 contributors to Debian 2.0 has changed over time. Table 3 gives an overview of the number of contributors from the original group left at each release, as well as the number of packages maintained by them. As the figure shows, the number decreases steadily, with only 117 of the original 216 contributors (54.2%) still possessing ownership of a package in June 2005. Based on these figures, we concluded in a previous paper that the half-life value had not been reached after six and a half years and estimated that the half-life value would be around 7.5 years (or 90 months) (Robles et al., 2005).

Taking the more fine-grained information from uploads into account, we can now revise these findings. Taking package ownership as an indication for activity is error prone, since it

Figure 3. Half-life of Debian maintainers: How populations shrink over time. The horizontal line at 0.5 shows when a population reaches half-life



has been shown that a number of "maintainers" are actually inactive and do not maintain their packages (Michlmayr, 2004). It can take several months or longer until the situation is resolved, in particular if maintainers are busy but do not want to admit to themselves that they do not actually have enough time anymore. Uploads are therefore a much better measure in this case since they show activity. While this measure does not show the effort done by a maintainer it shows that they are still active, which is the question being asked here.

Based on data from uploads, we can see that as of the three-month period starting in June 1998, only 187 out of the original 216 contributors (which still possess packages in the release done in July 1998) are still active. Taking these 187 developers as the new population, we find that the group reaches their half-life in the periods between June and September 2004. This leads to a half-life value of less than 78 months (6.5 years). This is in line with the originally estimated value of 7.5 years, a slight over-prediction due to the fact that it takes some time until inactivity is reflected in the maintainer field of a package. The value obtained from this population is also in line with those obtained from other populations observed as part of the investigation of uploads to Debian, as can be seen in Figure 3. They all show a half-life of between 75 and 90 months.

It would be interesting to perform further analysis about which factors influence how long volunteers remain active. There is already evidence that some volunteers face feelings of burn-out (Hertel et al., 2003), but further studies into human-resource management and motivation in libre software projects could have positive effects on extending the half-life of volunteer contributions.

The number of packages for which these developers are responsible is also interesting. The initial number of packages maintained by the 216 contributors of Debian 2.0 was 1,101. The corresponding number of packages in Debian 2.1 (around nine months later) for the developers remaining rose to 1,351 and then to 1,457 for

Debian 2.2, where the maximum number of packages was achieved. Then it decreased to 1,305 for Debian 3.0, and in the last Debian version it had similar figures as in their first release, although now with half of the maintainers.

This data shows that there has been a continuous increase in the mean number of packages that maintainers are responsible for. While the number of packages per maintainer was slightly above five for the 2.0 release, this number has grown to nine packages per maintainer in release 3.1. It also seems that the mean value keeps on growing, although at a lower pace, and that it has a tendency towards a value around nine as we can see from the last two releases. Given the large amount of time between the last two releases, we can assume that this observed pattern is stable. We have already discussed possible explanations for this behavior with the data about the evolution in number of Debian maintainers (see Table 2).

Regarding the involvement of maintainers, we can see from the median that there is a general shift towards maintaining more packages, as the median value starts with three packages and raises up to five for Debian 2.2 and Debian 3.0. The mode, on the other hand, shows that the number of maintainers who only maintain one package decreases over time more quickly than the number of total maintainers (the total number of maintainers drops from 216 to 117, a drop of 46%, while the number of maintainers who maintain only one package decreases from 52 to 20, a 62% drop; this means that the number of maintainers with more than one package shrinks from 164 to 97, which is only 34% less, almost half of the drop for maintainers in charge of a single package). The cause for this may be twofold: On one hand those maintainers could have left the project, and on the other they could have gotten more involved in it by maintaining more packages. In any case, maintaining one package could be seen as a "hot" zone in which nobody stays for a long time and where a decision has to be taken: to get more involved in the project or to leave.

The standard deviation and the Gini coefficient give an idea of the distribution of work. Both values show that there is tendency to have a less equally distributed load of work. Of particular interest is the Gini coefficient, which starts at almost 0.5 and grows up to 0.574. The maximum number of packages that a single maintainer is in charge of grows consequently, from 50 packages in Debian 2.0 to 83 in Debian 3.1. It should be noted that the maximum number of packages of the first three Debian versions under study correspond to a different person than the last two.

INVESTIGATING MAINTAINER EXPERIENCE

In the previous paragraphs we tracked maintainers from Debian version 2.0 over time to see how their contributions evolved. In the following we are going to do the opposite; we will take the last Debian release (Debian 3.1) and will try to track when maintainers first started participating in the project. This will allow us to have a measure of experience in the project. Maintainers who entered in the same release will be grouped and analyzed together.

Figure 4 shows when currently active maintainers got involved in the project. For every maintainer of a package in the latest release, we have investigated in which release their first contribution can be found. In addition to the 117 developers who have made steady contributions since July 1998 (release 2.0), 55 participants got involved before Debian 2.1, and 106 arrived with Debian 2.2. In the last two stable releases, 384 and 652 new maintainers have been identified.

The evolution of the number of packages per maintainer given in Table 4 provides evidence about the impact of experience on the number of packages maintained. We can see that for the first three versions considered, the values range around 9.0 up to 11.5 packages per maintainer, while in the last two the number of packages

Table 3. Packages maintained by the Debian 2.0 maintainers

Date	Release	Maint	Packages	Pkg/Maint	Median	Mode	Std. Dev	Gini	Max
Jul 98	2.0	216	1,101	5.1	3	1 (52)	5.8	0.492	50
Mar 99	2.1	207	1,351	6.5	4	1 (38)	7.3	0.501	55
Aug 00	2.2	188	1,457	7.8	5	1 (33)	9.2	0.515	69
Jul 02	3.0	147	1,305	8.9	5	2 (20)	10.6	0.540	65
Jun 05	3.1	117	1,055	9.0	4	1 (20)	12.1	0.574	83

Figure 4. First stable release to which Debian 3.1 maintainers have contributed

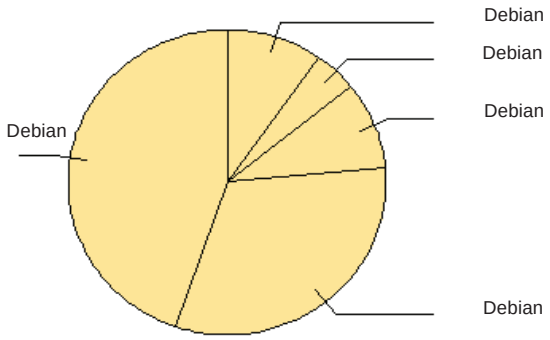


Table 4. First release as maintainer for maintainers in Debian 3.1

Date	Releale	Maint	Packages	Pkg/Maint	Median	Mode	Std. Dev	Gini	Max
Jul 98	2.0	117	1,055	9.0	4	1 (20)	12.1	0.574	83
Mar 99	2.1	55	631	11.5	6	5 (8)	15.1	0.544	81
Aug 00	2.2	106	1,008	8.8	6	1;2 (16)	9.7	0.515	55
Jul 02	3.0	384	2,835	7.2	5	1 (63)	9.5	0.511	121
Jun 05	3.1	652	2,221	4.0	2	1 (246)	7.4	0.570	106

is lower. In general, the tendency is that older maintainers have more packages than those who joined later. The exceptions are maintainers who joined with Debian 2.1. For this version, we can see a statistical distortion in the mode as it has a value of 5 while the values for all other versions is 1 (or 1 and 2 in the case of Debian 2.2).

With regard to the median value, we can see that it is also higher for more experienced maintainers, although in this case it is not that clear as we have seen with the mean number of packages. The different behavior of the last two releases is interesting: While Debian 3.1 has a median of two with many (up to 246) maintainers only in charge of one package, those maintainers who entered in Debian 3.0 and who are still active, have a median value of five and a smaller proportion of them maintain a single package. Again, this supports our previous conclusion that maintaining a single package is only a temporary situation.

The standard deviation of the sample does not give us much information in this case. Maybe it stresses the distorted behavior of Debian 2.1 with such a high value; interestingly enough it shows

that the data is more homogeneous as we come nearer to Debian 3.1. This is an expected effect, as "younger" maintainers should have a similar (smaller) involvement while "older" ones may vary more. Nonetheless, the Gini coefficient does not necessarily support this finding as the values show no clear tendency over time (the highest value is for Debian 2.0 followed closely by Debian 3.1). This is also the case for the maximum number of packages maintained by a single person which fluctuates from version to version without any predictable direction. In any case, we can see that very active maintainers enter any time in the project, some of them with a surprisingly high involvement. For instance, from July 2002 to June 2005 one new maintainer became in charge of 106 packages! Obviously, the effort needed for the maintenance of a package can vary widely. Further exploration is needed to estimate the effort associated with the maintenance of these packages.

PACKAGES OF MAINTAINERS WHO LEFT THE PROJECT

When maintainers leave the project, their packages become unmaintained (the Debian project uses the expression "orphaned"). These packages may be taken up by others ("adopted" in the Debian jargon), or they will not be present in the next stable release. In Table 5 the ratios and numbers of orphaned and adopted packages between any pair of the studied releases are shown.

The table should be read as follows: The first column shows the number of packages, 15, for which their maintainers have left the project (column "Orphaned") and that have been adopted, 14, by other (possibly new) maintainers (column "Adopted") from Debian 2.0 to Debian 2.1. This means that the percentage of orphaned packages that have been adopted is 93.3% (column "Adopt/Orph"), so in this case few packages got lost. The last two columns help situating the amount of orphaned packages we are talking about, giving the share of orphaned packages in comparison to the total number of packages for each release.

Looking at the rest of the rows in the table, we can see that the percentage of adopted packages is very high: more than 60% for all releases considered. This happens even for releases with a very high portion of orphaned packages (for instance, between version 2.0 and 3.1). In other words, even though maintainers who left Debian between July 1997 and June 2005 were responsible for 35.0% of the packages in Debian 2.0, 65.7% of these packages can still be found in version

Table 5. Orphaning and adoption of packages

Release 1	Release 2	Orphaned	Adopted	Adopt/Orph	Orph/Total1	Orph/Total2
2.0	2.1	15	14	93.3%	1.3%	1.0%
2.0	2.2	61	40	65.6%	5.5%	1.5%
2.0	3.0	231	171	74.0%	21.0%	4.5%
2.0	3.1	385	253	65.7%	35.0%	4.8%
2.1	2.2	47	31	66.0%	3.0%	1.8%
2.1	3.0	302	220	72.8%	19.4%	5.9%
2.1	3.1	516	332	64.3%	33.1%	6.5%
2.2	3.0	281	207	73.7%	10.8%	5.5%
2.2	3.1	685	433	63.2%	26.3%	8.6%
3.0	3.1	685	435	63.5%	13.4%	8.6%

3.0. We can thus affirm that Debian counts on a natural "regeneration" process for its voluntary contributors and that there is a high probability that the packages of a maintainer who leaves the project will be adopted by others.

Another interesting fact is that the ratio of adopted to orphaned packages is decreasing in later releases. This means that the number of orphaned packages grows more quickly than that of adopted, i.e., there are some packages missing in every new release. If a package is unmaintained and falls off the next release, it will probably not enter a future one. In this study we have only considered removed packages from maintainers who left the project, but it is likely that some software will also be abandoned by maintainers who still remain active and are therefore not covered by this study.

In any case, it should be noted that users are left unsupported when a package (maybe providing a unique functionality) from a previous release is not present in subsequent ones. It may therefore be beneficial to establish mechanisms to ensure that only packages which can be supported in the long term will not be introduced in the first place, or that at least that they be introduced only in a section of the Debian repository which is clearly marked as being less supported.

EXPERIENCE AND IMPORTANCE

We have used data from the Debian Popularity Contest (presented in detail in the section Debian Popularity Contest) to find out whether more "important' packages are maintained by more experienced volunteers. Table 6 shows the data corresponding to installations and use of packages by developers which are still in the project, and which were already present in the studied releases. In it we can see, for instance, that Debian 2.0 and 3.1 have 117 common maintainers, who are responsible for 1,091 packages which have been installed 1,305,907 times and 576,991 that are regularly used.

The CMaint column shows how many maintainers Debian 3.1 had in common with the release in the first column, while the CPkg shows the number of packages maintained by them. Columns Installations and Votes give the sum of the packages installed and voted (used regularly) for those packages maintained by common maintainers. The last two columns show the ratios of both to common maintainers.

If we take the number of installations per maintainer and the number of regularly used packages per maintainer ("Votes/Maint') we can answer the question we proposed in the section Research Objectives and Goals. According to our hypothesis, these ratios decrease over time, which would mean that more experienced volunteers maintain packages which are installed and used more often. In fact, this can be observed through all Debian

Table 6. Installations and regular use of packages

Release	CMaint	CPkg	Installations	Votes	Inst/Maint	Votes/Maint
2.0	121	1,091	1,305,907	576,991	10792.6	4768.5
2.1	176	1,722	1,584,413	673,236	9002.3	3825.2
2.2	290	2,730	2,217,199	885,448	7645.5	3053.3
3.0	683	5,565	3,923,753	1,405,322	5744.9	2057.6
3.1	1315	7,989	5,248,869	1,711,496	3991.5	1301.5

releases. An alternative (or complementary) explanation to our initial hypothesis is that many of the essential components of the Debian system were introduced in the first releases, and that new packages are mostly add-ons and software that are not installed and used that often.

CONCLUSION AND FURTHER WORK

We have conducted a quantitative study of the evolution of the Debian maintainership over the last six-and-a-half years. We have retrieved and analyzed publicly available data in order to find out how Debian handles the volatility of the volunteers who made it happen.

Some of the most interesting findings are:

- Both the number of Debian maintainers and the number of packages per maintainer grow over time.
- The number of maintainers from previous releases who remain active is very high, with an estimated half-life of around 6.5 years (78 months). Slightly less than half of the maintainers from Debian 2.0 still contribute to the current release after more than 7 years.
- Developers tend to maintain more and more packages as they gain experience in the project.
- However, this does not mean that maintainers who have been in the project for more time maintain more packages than newer maintainers. In fact, in the latest release, the highest packages per maintainer ratio is shown by those entering the project around the year 2000.

From these facts, it can be said that Debian maintainers tend to commit to the project for long periods of time. However, there is a worrisome trend towards a higher and higher ratio of packages per maintainer, which could imply scalability problems as the number of packages in the distribution increases, if the project doesn't admit a proportional number of developers.

Another issue on which we have focused is what happens to those packages that were maintained by developers who left the project. Most of them are taken over by other maintainers, so that we can state that a natural "regeneration" exists. Based on the data we have researched, those packages which are not adopted by other maintainers in the next release, and are therefore not present in it, are unlikely to be re-introduced in future releases.

Finally, we have also found that more experienced maintainers are responsible for packages which are installed more often and used more regularly.

In addition to the new insights gained in this investigation, we have proposed a number of further studies to elaborate on the findings of the present chapter. In particular, team maintenance and its impact on the quality of packages would be interesting to research. It is also not clear why there is an increase in the ratio of packages per maintainer. Possible explanations are that better tools and practices lead to more efficiency, or that with the success of libre software, new volunteers show more motivation and commitment, but more data is needed before these explanations can be conclusive.

From a more general point of view, this study explores the behavior of volunteers in libre software projects and provides some answers as to why these kinds of voluntary contributions are capable of producing such large, mature and stable systems over time, even when the project has no means for forcing any single developer to do any given task and when members may leave the project during important development phases. It is impossible to infer the behavior of volunteer developers just from the study of a single project, but given the size and relevance of the Debian project, at least some conclusions can be exposed as hypotheses

for validating in later research efforts.

One of them is the stability of volunteer work over time. The mean life of contributors in the project is probably longer than in many software companies, which would have a clear impact on the maintenance of the software (it is likely that developers with experience in a module are available for its maintenance over long periods of time). Another one is that volunteers tend to take over more work with the passing of time if they remain in the project: In other words, they voluntarily increase their responsibilities in the project. Whether this is because it is easier for them because of their experience, or because they devote more effort to the project, is for now an open question. Yet a third one is the stability of the voluntary effort when some individuals leave the project: Most of their work is taken over by other developers. Therefore, despite being completely based on volunteers, the project organizes itself rather well with respect to drop-outs, which is an interesting lesson about how the project can survive in the long term.

As a final summary, we have found that given that there are no formal ways of forcing a developer to assume any given task, voluntary efforts seem to be more stable over time, and more reliable with respect to individuals leaving the project than we had initially expected.

ACKNOWLEDGMENT

The work of Martin Michlmayr has been funded in part by Google, Intel, and the EPSRC. The work of Gregorio Robles and Jesus M. Gonzalez-Barahona has been funded in part by the European Commission under the CALIBRE CA, IST program, contract number 004337. We would also like to thank the anonymous reviewers for their extensive comments.

REFERENCES

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday, 10*(2).

Debian Social Contract. (2006). Debian social contract. Retrieved from http://www.debian.org/social_contract

Dinh-Trong, T. T., & Bieman, J. M. (2005). The FreeBSD project: A replication case study of Open Source development. *IEEE Transactions on Software Engineering, 31*(6), 481-494.

Garzarelli, G., & Galoppini, R. (2003). *Capability coordination in modular organization: Voluntary FS/OSS production and the case of Debian GNU/Linux.*

German, D. (2004). The GNOME project: A case study of open source, global software development. J*ournal of Software Process: Improvement and Practice*, *8*(4), 201-215.

Ghosh, R. A., Glott, R., Krieger, B., & Robles, G. (2002). Survey of developers (Free/libre and open source software: Survey and study). Technical report, International Institute of Infonomics, University of Maastricht, The Netherlands. Retrieved from http://www.infonomics.nl/FLOSS/report

Ghosh, R. A., & Prakash, V. V. (2000). The orbiten free software survey. *First Monday, 5*(7). Retrieved from http://www.firstmonday.dk/issues/issue5_7/ghosh/

Gonzalez-Barahona, J. M., Ortuno Perez, M. A., de las Heras Quiros, P., Centeno Gonzalez, J., & Matellan Olivera, V. (2001). Counting potatoes: The size of Debian 2.2. *Upgrade Magazine*, *II*(6), 60-66.

Gonzalez-Barahona, J. M., Robles, G., Ortuno Perez, M., Rodero-Merino, L., Centeno Gonzalez, J., et al., (2004). Analyzing the anatomy of GNU/Linux distributions: Methodology and case

studies (Red Hat and Debian). In S. Koch (Ed.), *Free/open source software development* (pp. 27-58). Hershey, PA: Idea Group Publishing.

Herbsleb, J. D., Mockus, A., Finholt, T. A., & Grinter, R. E. (2001). An empirical study of global software development: Distance and speed. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering* (pp. 81-90).

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel. *Research Policy, 32*(7), 1159-1177.

Hunt, F., & Johnson, P. (2002). On the Pareto distribution of open source projects. In *Proceedings of Open Source Software Development Workshop*, Newcastle, UK.

Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12*(1), 27-42.

Lameter, C. (2002). *Debian GNU/Linux: The past, the present and the future.* Retrieved from http://telemetrybox.org/tokyo/

Michlmayr, M. (2004). Managing volunteer activity in free software projects. In *Proceedings of the USENIX 2004 Annual Technical Conference, FREENIX Track*, Boston (pp. 93-102).

Michlmayr, M., & Hill, B. M. (2003). Quality and the reliance on individuals in free software projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, OR (pp. 105-109).

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Monga, M. (2004). From bazaar to kibbutz: How freedom deals with coherence in the Debian project. In *Proceedings of the 4th Workshop on Open Source Software Engineering*, Edinburg, Scotland, UK.

O'Mahony, S. (2003). Guarding the commons: How community managed software projects to protect their work. *Research Policy, 32*, 1179-1198.

Robles, G. (2006). *Empirical software engineering research on libre software: Data sources, methodologies and results.* PhD thesis, Universidad Rey Juan Carlos.

Robles, G., Gonzalez-Barahona, J. M., & Michlmayr, M. (2005). Evolution of volunteer participation in libre software projects: Evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, Genoa, Italy (pp. 100-107).

ENDNOTES

1. In this chapter we will use the term "libre software" to refer to any software licensed under terms compliant with the Free Software Foundation definition of "free software," and the Open Source Initiative definition of "open source software," thus avoiding the controversy between those two terms.
2. http://popcon.debian.org
3. All the code used has been released as libre software, and can be obtained from http://libresoft.dat. escet.urjc.es/index.php?menu=Tools
4. The Gini coefficient is a normalized measure of inequality; values near 0 point out equal distributions while values close to 1 are indicative for high inequalities.

This work was previously published in Emerging Free and Open Source Software Practices, edited by S. Sowe; I. Stamelos; and I. Samoladas, pp. 1-24, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.6
Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects

Luis López-Fernández
Universidad Rey Juan Carlos, Spain

Gregorio Robles
Universidad Rey Juan Carlos, Spain

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain

Israel Herraiz
Universidad Rey Juan Carlos, Spain

ABSTRACT

Source code management repositories of large, long-lived libre (free, open source) software projects can be a source of valuable data about the organizational structure, evolution, and knowledge exchange in the corresponding development communities. Unfortunately, the sheer volume of the available information renders it almost unusable without applying methodologies which highlight the relevant information for a given aspect of the project. Such methodology is proposed in this article, based on well known concepts from the social networks analysis field, which can be used to study the relationships among developers and how they collaborate in different parts of a project. It is also applied to data mined from some well known projects (Apache, GNOME, and KDE), focusing on the characterization of their collaboration network architecture. These cases help to understand the potentials of the methodology and how it is applied, but also shows some relevant results which open new paths in the understanding of the informal organization of libre software development communities.

INTRODUCTION

Software projects are usually the collective work of many developers. In most cases, and especially in the case of large projects, those

developers are formally organized in a well defined (usually hierarchical) structure, with clear guidelines about how to interact with each other, and the procedures and channels to use. Each team of developers is assigned certain modules of the project, and only in rare cases do they work outside that realm. However, this is usually not the case with libre software[1] projects, where only loose (if any) formal structures are acknowledged. On the contrary, libre software developers usually have access to any part of the software, and even in the case of large projects, they can move freely to a certain extent from one module to other, with only some restrictions imposed by common usage in the project and the rules on which developers themselves have agreed to.

In fact, during the late 1990s some voices started to claim that the success of some libre software projects was rooted in this different way of organization, which was referred to as the "bazaar development model," described by Eric Raymond (1997) and later complemented by some more formal models of nonhierarchical coordination (Elliott & Scacchi, 2004; Healy & Schussman, 2003). Some empirical studies have found that many libre software projects cannot follow this bazaar-style model, since they are composed of just one or two developers (Healy & Schussman, 2003; Krishnamurthy, 2002), but the idea remains valid for large projects, with tens or even hundreds of developers, where coordination is obviously achieved, but (usually) not by using formal procedures. These latter cases have gained much attention from the software engineering community during the last years, in part because despite apparently breaking some traditional premises (hard-to-find requirement studies, apparently no internal structure, global software development, etc.) final products of reasonable quality are being delivered. Large libre software projects are also *suspicious* of breaking one of the traditional software evolution *laws*, showing linear or even superlinear growth even after reaching a size of several millions of lines of code (Godfrey & Tu, 2000; Robles, Amor, Gonzalez-Barahona, & Herraiz, 2005a). The *laws* of software evolution state that the evolution of a system is a self-regulating process that maintains its organizational stability. Thus, unless feedback mechanisms are appropriately introduced, the effective global activity tends to remain constant, and incremental growth declines. The fact that several studies on some large libre software projects show evidence that some of these *laws* are disobeyed may be indicative of an efficient organizational structure.

On the other hand, the study of several large libre software projects has shown evidence about the unequal distribution of the contributions of developers (Dinh-Trong & Bieman, 2005; Koch & Schneider, 2002; Mockus, Fielding, & Herbsleb, 2002). These studies have identified roles within the development community, and have discovered that a large fraction of the development work is done by a small group of about 15 persons, which has been called the "core" group. The number of developers is around one order of magnitude larger, and the number of occasional bug reporters is again about one order of magnitude larger than that of developers (Dinh-Trong & Bieman, 2005; Mockus et al., 2002). This is what has been called the *onion* structure of libre software projects (Crowston, Scozzi, & Buonocore, 2003). In this direction, it has also been suggested that large projects need to adopt policies to divide the work, giving rise to smaller, clearly defined projects (Mockus et al., 2002). This trend can be observed in the organization of the CVS[2] repository of really large libre software projects, where the code base is split into modules with their own maintainers, goals, and so forth. Modules are usually supposed to be built maintaining the interrelationships to a minimum, so that independent evolution is possible (Germán, 2004a).

In this article, a new approach is explored in order to study the informal structure and organization of the developers in large libre software

projects. It is based on the application of well known social networks analysis (SNA) techniques to development data obtained from the versioning system (CVS). According to the classical Conway's *law*, organizations designing systems are constrained to produce designs which are copies of their communication structures (Conway, 1968). Following this line of reasoning, the relationships among modules will be studied, and the dual case of those among developers. Our target is the advancement of the knowledge about the informal coordination structures that are the key to understanding how these large libre software projects can work in the apparent absence of formalized structures, and where the limits are of those ways of coordinating and exchanging information. We have designed a methodology following this approach, and have also applied it to some well known projects. Although the aim of our approach is mainly descriptive, not proposing novel models for project evolution or agent behavior, just trying to describe in as much detail as possible the organizational structure of libre software projects, our work is illustrative of the power of the SNA techniques. To attain this goal, our approach is similar to that presented in Madey, Freeh, and Tynan (2002) and Xu, Gao, Christley, and Madey (2005): we consider libre software projects as complex systems and characterize them by using mathematical formalisms. As a result, some interesting facts related to the organizational structure of libre software projects have been uncovered.

The remainder of this article is organized as follows. The next section contains a basic introduction to SNA, and how we intend to apply its techniques to the study of libre software projects based on the data available in their CVS repositories. The third section specifies in detail the methodology for such a study, followed by the fourth section with a brief introduction to a set of classical social network analysis parameters. After that, the fifth section presents the main characteristics of the networks corresponding to the three projects used as case examples: Apache, GNOME, and KDE. This serves as an introduction to the more detailed comments on several aspects of those projects, presented in the sixth, seventh, eighth, ninth, and tenth sections. The final section offers some conclusions, comments on some related work, and discusses further lines of research.

APPLICATION OF SNA TO LIBRE SOFTWARE PROJECTS

The study and characterization of complex systems is a fruitful research area, with many interesting open problems. Special attention has been paid recently to complex networks, where graph and network analysis play an important role. This approach is gaining popularity due to its intrinsic power to reduce a system to its single components and relationships. Network characterization is widely used in many scientific and technological disciplines, such as neurobiology (Watts & Strogatz, 1998), computer networks (Albert, Barabási, Jeong, & Bianconi, 2000), or linguistics (Kumar, Raghavan, Rajagopalan, & Tomkins, 2002).

Although some voices argue that the software development process found in libre software projects is hardly to be considered as a new development paradigm (Fuggetta, 2003); without doubt, the way it handles its human resources differs completely from traditional organizations (Germán, 2004b). In both cases, traditional and libre software environments, the human factor is of key importance for the development process and how the software evolves (Gîrba, Kuhn, Seeberger, & Ducasse, 2005), but the volunteer nature of many contributors in the libre software case makes it a clearly differentiated situation (Robles, González-Barahona, & Michlmayr, 2005b).

Previous research on this topic has both attended to technical and organizational points of view. Germán used data from a versioning repository in time to determine feature-adding

and bug-correcting phases. He also found evidence for developer territoriality (software artifacts that are mainly, if not uniquely, *touched* by a single developer) (Germán, 2004a).

The intention of other papers has been to uncover the social structure of the underlying community. The first efforts in the libre software world are due to Madey et al. (2002), who took data from the largest libre software projects repository, SourceForge.net, and inferred relationships among developers that contributed to projects in common. A statistical analysis of some basic social network parameters can also be found by López, Gonzalez-Barahona, and Robles (2004) for some large libre software projects. Xu et al. (2005) have presented a more profound topological analysis of the libre software community, joining in the same work characteristics from previous papers: data based on the SourceForge platform and a statistical analysis of some parameters with the goal of gaining knowledge on the topology of the libre software phenomenon. This has also been the intention of González-Barahona, López-Fernández, and Robles (2004), where a structure-finding algorithm was used to obtain the evolution in time of the organization of the Apache project. Wagstrom, Herbsleb, and Carley (2005) propose to use the knowledge acquired from analyzing libre software projects with SNA for the creation of models that help understand the underlying social and technical process.

METHODOLOGY

The first problem to solve when using SNA is getting the information to construct the network to analyze. One especially interesting kind of data sources is the records maintained by many computer-based systems. For instance, Guimera, Danon, Diaz-Guilera, Giralt, and Arenas (2003) analyze informal networking on organizations using tracks of e-mail exchanges. Therefore, from the many kinds of records available about the activity of a libre software project, those provided by the CVS system where source code is stored have been the ones chosen. Those records offer information about who modified the code, and when and how, in many cases from the very beginning of the project, in some cases over a total period of time above 10 years.

The information in the CVS repository of a project includes an accurate and detailed picture of the organizational structure of the software, and of the developers working on it. When two developers work on the same project module, they have to exchange (directly or maybe indirectly) information and knowledge to coordinate their actions and produce a working result. It seems reasonable to assume that the higher their contributions to the module, the higher the strength of their informal connection.

Based on this assumption, a specific kind of social network has been considered, those called affiliation networks. They are characterized by showing two types of vertices: *actors* and *groups*. When the network is represented with actors as vertices, each one is usually associated with a particular person, and two of them are linked together when they belong to the same group. When the network is represented with groups as vertices, two groups are connected when there is, at least, one actor belonging, at the same time, to both groups. In our case, actors will be identified as developers, and groups as software modules. The "belong to" relationship will be in fact "has contributed to." This approach will result in a dual view of the same organization: as a network of modules linked by common developers, and as a network of developers linked by common modules. Similar approaches have been used for analyzing other complex organizations, like the network of scientific authors (Newman, 2001a, b) or the network of movie actors (Albert & Barabasi, 2002).

To finish the characterization of our networks, weighted edges are being considered. This means that it is not only taken into account whether a

node has some relationship with any other, but also the strength of that relationship. In our case, the weight will be related to the size of contributions to common modules (in the case of developers) and to the size of contributions by common developers (in the case of modules). It should be noted that from the methodological point of view, the use of weights is a major contribution of this article in comparison with previous works describing SNA techniques applied to libre software (Madey et al., 2002; Wagstrom et al., 2005; Xu et al., 2005). As we will see in this article, the use of weights is indicated as the distribution of work follows a very unequal distribution, in the range of a Pareto distribution[3] (Ghosh & Prakash, 2000). Our assumption at this point is that considering a link between two major contributing developers that equals the one between two random chosen developers, introducing an important bias in the results regarding the distribution of work observed in libre software environments.

Once we have identified how we want to use SNA for libre software projects, a well defined methodology is proposed in order to apply those ideas to any libre software with a public CVS repository. The process begins by downloading the relevant information from the CVS repository.[4] This information includes, for each commit (modification in a file in the repository): the date, the identifier of the developer (commiter), and the number of lines involved. Using all those records, the following networks are defined for characterizing the organization of the project:

- **Modules network.** Each vertex represents a particular software module (usually a directory in the CVS repository) of the project. Two modules are linked together by an edge when there is at least one commiter who has contributed to both. Those edges are weighted using a *degree of relationship* between the two modules, defined as the total number of commits performed by common commiters.
- **Commiters network.** In this case, each vertex represents a particular commiter (developer). Two commiters are linked by an edge when they have contributed to at least one common module. Again, edges are weighted by a *degree of relationship* defined as the total number of commits performed by both developers on modules to which both have contributed.

The definition being used for the *degree of relationship* is an attempt to measure the *closeness* of two vertices. The higher this parameter, the stronger the relationship between those vertices. In this sense, *cost of relationship* between any two vertices can also be defined as the inverse of their *degree of relationship*. In this sense, the *cost of relationship* defines a distance between vertices: the higher it is, the more difficult it is to reach one of them from the other. More formally, given a (connected) graph G and a pair of vertices i and j, we define the distance between them as $d_{ij}=\sum_{e\in P_s} c_r$, where e are all the edges in the shortest path P_s from i to j, and c_r is the *cost of relationship* of any of those edges.

Parameters

Once the networks are constructed based on the previous definitions, and the degrees and costs of relationship have been calculated for linked nodes, standard SNA concepts can be applied in order to define the following parameters of the network (the interpretation of the main implications of each parameter is also offered):

- **Degree.** The degree, k, of a vertex is the number of edges connected to it. In SNA, this parameter reflects the popularity of a vertex, in the sense that most popular vertices are those maintaining the highest number of relationships. More revealing than the degree of single vertices is the distribution

degree of the network (the probability of a vertex having a given degree). This is one of the most relevant characterizations because it provides essential information to understand the topology of a network (and if longitudinal data is available, the evolution of the topology). For example, it is well known that a random network follows a Poisson's distribution, while a network following a preferential attachment growth model presents a power law distribution (Albert & Barabasi, 2002). In our context, the degree of a commiter corresponds to the number of other commiters sharing modules with that committer, while the degree of a module is the total number of modules with which it shares developers.

- **Weighted degree.** When dealing with weighted networks, the degree of a vertex may be tricky. A vertex with a high degree is not necessarily well connected to the network because all its edges may be weak. On the other hand, a low degree vertex may be strongly attached to the network if its entire links are heavy. For this reason the weighted degree of a vertex, *w*, is defined as the sum of the weights of all the edges connected to it. The weighted degree of a vertex can be interpreted as the maximum capacity to receive information of that vertex. It is also related to the effort spent by the vertex in maintaining its relationships.
- **Clustering coefficient** (Watts & Strogatz, 1998). The clustering coefficient, *c*, of a vertex measures the transitivity of a network. Given a vertex *v* in a graph *G*, it can be defined as the probability that any two neighbors of *v* are connected (the neighbors of *v* are those vertices directly connected to *v*). Hence

$$c(v)=\frac{2E(v)}{k_v\ (k_v-} \qquad (1)$$

where k_v is the number of neighbors of k_v and $E(v)$ is the number of edges between them. The intuitive interpretation of the clustering coefficient is somehow subtle. If the total number of neighbors of v is k_v, the maximum number of edges than can exist within that neighborhood is k_v (k_v -1)/2 . Hence, the clustering coefficient represents the fraction of the number of edges that really are in a neighborhood. Therefore it can be considered as a measurement of the tendency of a given vertex to promote relationships among its neighbors. In a completely random graph, the clustering coefficient is low, because the probability of any two vertices being connected is the same, independently on them sharing a common neighbor. On the other hand, it has been shown that most social networks present significantly high clustering coefficients (for instance, the probability of two persons being friends is not independent from the fact that they share a common friend) (Albert & Barabasi, 2002; Watts, 2003).

  From an organizational point of view, the clustering coefficient helps to identify hot spots of knowledge exchange on dynamic networks. When this parameter is high for a vertex, that vertex is promoting its neighbors to interact with each other. Somehow it is fostering connections among its neighborhood. High clustering coefficients in networks are indicative for *cliques*. Besides, the clustering coefficient is also a measurement of the redundancy of the communication links around a vertex.
- **Weighted clustering coefficient** (Latora & Marchiori, 2003). The clustering coefficient does not consider the weight of edges. We may refine it by introducing the weighted clustering coefficient, c_w, of a vertex, which is an attempt to generalize the concept of clustering coefficient to weighted networks.

Given a vertex v in a weighted graph G it can be defined as:

$$c_w(v)=\sum_{i\neq j\in N_G(v)} w_{ij}\frac{1}{k_v(k_v-} \qquad (2)$$

where $N_G(v)$ is the neighborhood of v in G (the subgraph of all vertices connected to v), w_{ij} is the degree of relationship of the link between neighbor i and neighbor j (w_{ij}=0 if there are no links), and k_v is the number of neighbors. The weighted clustering coefficient can be interpreted as a measurement of the local efficiency of the network around a particular vertex, because vertices promoting strong interactions among their neighbors will have high values for this parameter. It can also be seen as a measurement of the redundancy of interactions around a vertex.

- **Distance centrality** (Sabidussi, 1996). The distance centrality of a vertex, D_c, is a measurement of its proximity to the rest. It is sometimes called *closeness centrality* as the higher its value the closer that vertex is (on average) to the others. Given a vertex v and a graph G, it can be defined as:

$$D_c(v)=\frac{1}{\sum_{t\in G}} \qquad (3)$$

where $d_G(v,t)$ is the minimum distance from vertex v to vertex t (i.e., the sum of the costs of relationship of all edges in the shortest path from v to t). The distance centrality can be interpreted as a measurement of the influence of a vertex in a graph because the higher its value, the easier for that vertex to spread information through that network. Observe that when a given vertex is "far" from the others, it has a low degree of relationship (i.e., a high cost of relationship) with the rest. So, the term $\sum_{t\in G} d_G(v.t)$ will increase, meaning that it does not occupy a central position in the network. In that case, the distance centrality will be low.

Research has shown that employees who are central in networks learn faster, perform better, and are more committed to the organization. These employees are also less likely to turn over. Besides, from the point of view of information propagation, vertices with high centrality are like "hills" on the plain, in the sense that any knowledge is put on them is rapidly seen by the rest and spreads easily to the rest of the organization.

- **Betweenness centrality** (Anthonisse, 1971; Freeman, 1977). The betweenness centrality of a vertex, c, is a measurement of the number of shortest paths traversing that particular vertex. Given a vertex v and a graph G, it can be defined as:

$$B_c(v)=\sum \frac{}{\sigma_{st}} \qquad (4)$$

where $\sigma_{st}(v)$ is the number of shortest paths from s to t going through v, and σ_{st} is the total number of shortest paths between s and t. The betweenness centrality of a vertex can be interpreted as a measurement of the information control that it can perform on a graph, in the sense that vertices with a high value are intermediate nodes for the communication of the rest. In our context, given that we have weighted networks, multiple shortest paths between any pair of vertices are highly improbable. So, the term $\sigma_{st}(v)$ /σ_{st} takes usually only two values: 1, if the shortest path between s and t goes through v, or 0 otherwise. So, the betweenness centrality is just a measurement of the number of shortest paths traversing a given vertex. In the SNA literature vertices with high betweenness centrality are known to cover "structural holes." That is, those vertices glue together parts of the organization that would be otherwise far away from each other. They receive a diverse combination of information available to no one else in the network and have therefore a higher prob-

ability of being involved in the knowledge generation processes.

High values of the clustering coefficient are usually a symptom of *small world* behavior. The small world behavior of a network can be analyzed by comparing it with an equivalent (in number of vertices and edges) random network. When a network has a diameter (or average distance among vertices) similar to its random counterpart but, at the same time, has a higher average clustering coefficient, it is defined as a small world. It is well known (Watts, 2003) that small world networks are those optimizing the short and long term information flow efficiency. Those networks are also especially well adapted to solve the problem of searching knowledge through their vertices.

Table 1 summarizes the various SNA parameters that have been presented in this section, their meanings, and the information they provide. These parameters, and their distributions and correlations will characterize the corresponding networks. From their study, a lot can be learned about the underlying organization and structure that those networks capture. An attempt to illustrate this is found in the following sections

Table 1. Summary of the SNA parameters described in this article, their meaning and their interpretation

Parameter	Meaning	Interpretation
Degree of relationship	Common activity among two entities (measured in commits)	How strong the relationship is
Cost of relationship	Inverse of the degree of relationship	Gives the cost of reaching one vertex from the other
Degree	Number of vertices connected to a node	Popularity of a vertex
Distribution degree	Probability of a vertex having a given degree	Topology of the network (Poisson or power law distributions)
Weighted degree	Degree considering weights of the links among vertices	Maximum capacity to receive information for a vertex. Effort in maintaining the relationships
Clustering coefficient	Fraction of the total number of edges that could exist for a given vertex that really exist	Transitivity of a network: tendency of a vertex to promote relationships among its neighbors. Helps identifying hot spots of knowledge interchange in dynamic networks
Weighted clustering coefficient	Generalization of the clustering coefficient concept to weighted networks	Local efficiency of the network around a vertex. Redundancy of interactions around a vertex
Distance centrality	Measurement of the proximity of a vertex to the rest	Gives the influence of a vertex in a graph. The higher the value the easier it is for the vertex to spread information through the network
Betweenness centrality	Number of shortest paths traversing a vertex	Measurement of the information control. Higher values mean that the vertex is an intermediate node for the communication of the rest. Vertices with high values are known to cover “structural holes”
Small world	Diameter (or average distance among vertices) similar but higher average clustering coefficient than random network	Optimizes short and long term information flow efficiency. Especially well adapted to solve the problem of searching knowledge through their vertices

by studying several cases on real libre software projects.

CASE STUDIES: APACHE, KDE, AND GNOME

Apache,[5] KDE, and GNOME are all well known libre software projects, large in size (each one well above the million of lines of code), in which several subprojects (modules) can be identified. They have already been studied from several points of view (Germán, 2004a; Koch & Schneider, 2002; Mockus et al., 2002). Here, they will be used to show some of the features of our proposed methodology for applying SNA to software projects.

The use of versioning systems is fortunately the case for most large libre software projects. Some approaches on how to gather information from versioning repositories, in particular CVS (Germán, 2004a; Germán & Hindle, 2006; Zimmermann & Weißgerber, 2004; Zimmermann, Weißgerber, Diehl & Zeller, 2005), have been presented, and are used in this study. Therefore, focus is set on what to do once that information is available, and not on how to gather it.

Tables 2 and 3 summarize the main parameters of both. In the case of commiter networks the GNOME case has been omitted.

By comparing the data in both tables some interesting conclusions can already be drawn. It may be observed, for instance, that the average number of commiters per module is greater in KDE (12.5) than in Apache (4.3), meaning more people being involved in the average KDE subproject. It can also be highlighted that the average degree on the commiters networks is in general larger than in the modules ones. This is especially true for KDE, which rises from a value of 21.4 in the latter case to 225 in the former. In the case of Apache it only raises from 14.2 to 31.1. Therefore, we can conclude that in those cases, commiters are much more linked than modules. The percentage of modules linked gives an idea of the synergy (in form of sharing information and experience) in a network as many modules have commiters in common. It can be assumed that this happens because of the technical proximity between modules. Regarding our case studies, KDE and GNOME show percentages near 30%, while the average Apache module is only linked to 8% of the other modules in the versioning system. So, Apache is specially fragmented in several module

Table 2. Number of vertices and edges of the module networks in the Apache, GNOME, and KDE projects

Project name	Modules (Vertices)	Edges		% of edges (avg)
Apache	175	2491	14.23	8.13
KDE	73	1560	21.37	29.27
GNOME	667	121,134	181.61	27.23

Table 3. Number of vertices and edges of the commiter networks in the Apache and KDE projects

Project name		Edges	Commiters per module	Avg Number of edges
Apache	751	23,324	4.3	31.06
KDE	915	205,877	12.5	225.00
GNOME	869	N/A	1.3	N/A

families that have no commiters in common. KDE and GNOME have a higher cohesion, while there is more dispersion in Apache.

In the following sections some specific aspects of all those networks will be studied, with the idea of illustrating both how the methodology is applied and which kind of results can be obtained from it.

DEGREE IN THE MODULES NETWORK

Table 4 shows that the number of modules for Apache (175), KDE (73), and GNOME (667) differ significantly. These projects are similar in software size (at least in order of magnitude), so the number of modules depends mainly on the various strategies that the projects follow when creating a new module. KDE has a structured CVS; applications that belong together are usually grouped into one module (so, for instance, there exists the *kdenetwork* module for many network applications or the *koffice* module for the various office suite programs). Apache has modules at the application level. Finally, GNOME follows a more *chaotic* approach, resulting in many more modules. Almost every application, even components (there are almost a dozen different GIMP add-ons with their own module) can be found to be a module in themselves.

The most popular characterization of network degree is the distribution degree $P(k)$, which measures the probability of a given vertex having exactly k edges. However, the representation of $P(k)$ in networks of a small size like ours is usually messy.[6] In these cases, the specialized literature prefers to use an associated parameter called the *cumulative distribution degree*, $CP(k)$, which is defined as $CP(k) = \sum_{k}^{\infty} P(i)$ and is usually represented in a log-log scale.

Table 4. Small world analysis for the module networks

Project name	<*d*> / <*rd*>	<*cc*> / <*rcc*>
Apache	2.06 / 1.47	0.73 / 0.19
KDE	1.31 / 1.11	0.88 / 0.65
GNOME	1.46 / 1.10	0.87 / 0.54

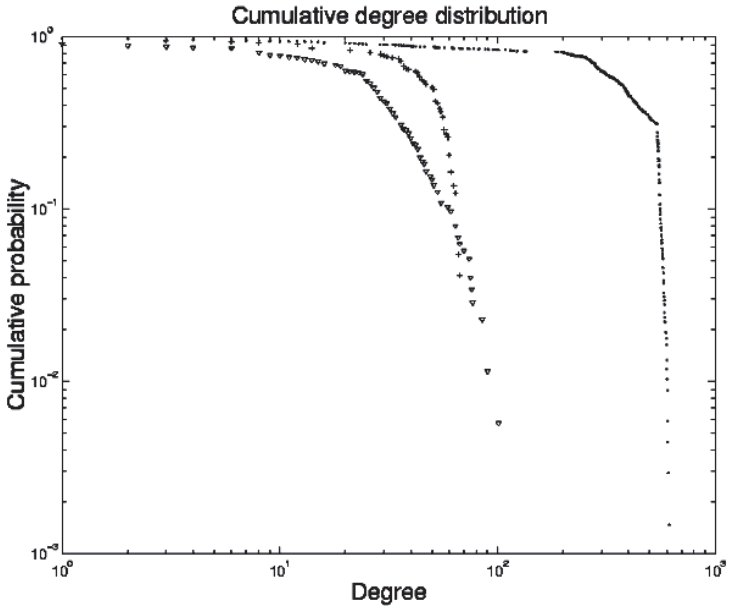
Figure 1 shows the cumulative distribution degree for our three networks. As it can be observed, all of them present a sharp cut off, which is a symptom of an exponential fall of the distribution degree tail. From a practical point of view, this means that none of our networks follow a power law distribution. This is quite a remarkable finding, because the specialized literature has shown that most social networks present power laws for this parameter. This implies that the growth of the network does not follow the traditional random preferential attachment law. Thus, it is difficult to come to any conclusions at this point; maybe by using a weighted network approach, as shown later, we could infer more information about the network topology.

Starting with the degree of the vertices, an analysis of assorts of the networks can also be carried out. The assorts measure the average degree of neighbors of vertices having a particular degree. For this reason it can also be called the *degree-degree distribution*.

Figure 2a represents this parameter for our networks. As can be observed, all three networks are elitist, in the sense that vertices tend to connect to other vertices having a similar degree ("rich" with "rich" and "poor" with "poor"). This is especially clear in the case of GNOME, where the curve approaches a linear equation with slope 1. Apache project deviates slightly from that behavior, showing some higher degree modules connected to other modules of a lower degree.

The previous analysis assumes unweighted networks. If weighted edges are considered now, similar conclusions are obtained. Figure 2b represents the cumulative weighted distribution degree of the networks. Comparing this picture with Figure 1, it may be remarked that the sharp exponential cut-offs have disappeared. This is

Figure 1. Cumulative degree distribution for Apache (▽), KDE (+), and GNOME (·)

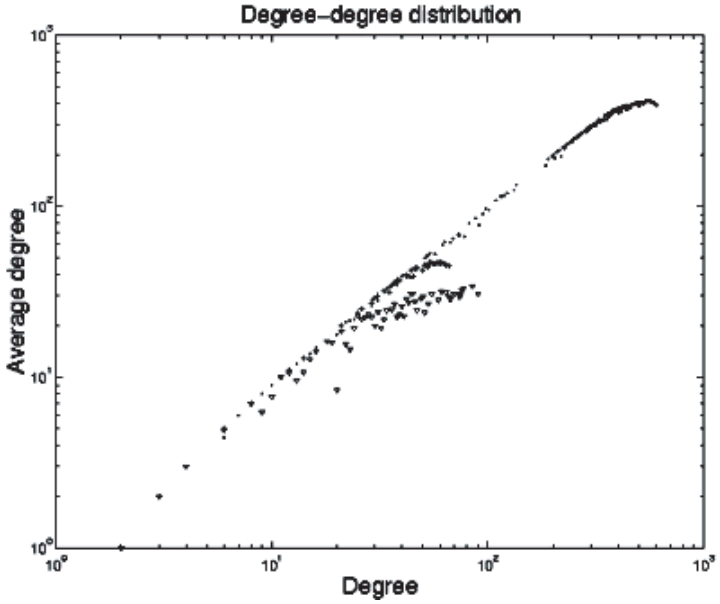


especially clear in the case of GNOME, where the curve tail can be clearly approximated by a power law. The interpretation for this finding is that the growth of that network could be driven by a preferential attachment law based on weighted degrees. This means that the probability of a new module to establish a link with a given vertex is proportional to the weighted degree of that vertex. That is, the commiters of new modules are, with high probability, commiters of modules which are well connected (have high weighted degree) in the network. It should be noted that the use of weights has given a more realistic picture.

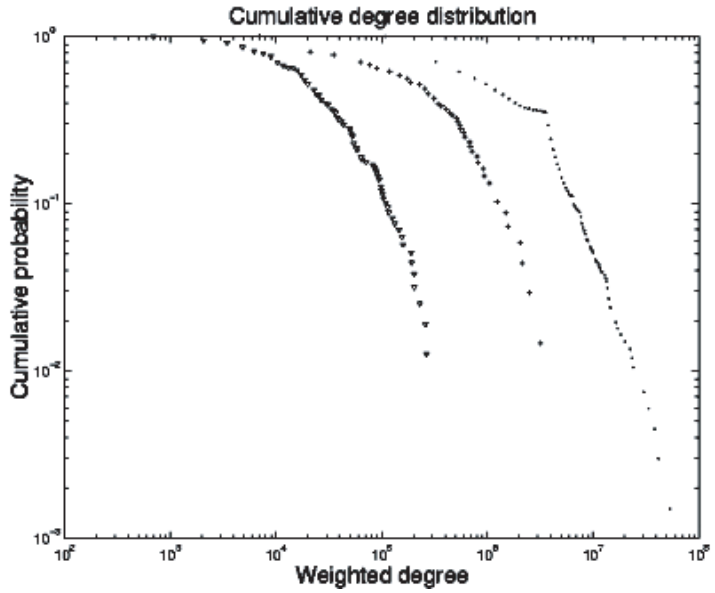
From Figure 1, it can be remarked that the sharp cut offs for Apache and KDE are close to each other. This means that the maximum numbers of relationships in both projects are similar. Nevertheless, observing Figure 2b, it can be seen that the KDE tail is clearly over the Apache tail. This fact implies that KDE weighted links are, on average, stronger than those of Apache. This can be quantitatively verified: we have calculated the average edge weight for the three projects obtaining 1,409.27 for Apache, 11,136.82 for KDE, and 7,661.18 for GNOME.

If multiplied, the average edge weight and the number of modules, the figures obtained are the total amount of commits performed by developers that contribute to at least two modules: 105,695 for Apache, 812,988 for KDE, and 5,110,007 for GNOME. This gives an idea of the modularity of the modules as a lower number of commits is indicative for developer work being more focused on a low number of modules. While the figures for Apache are not surprising (we have already noticed with previous parameters that is a high level of structure in the Apache project), the difference between KDE and GNOME is astonishing. The organization of the KDE CVS repository yields in more independent modules than the ones found in the one for GNOME.

Figure 2. Assortativity (degree - degree distribution) for Apache (), KDE (+) and GNOME (·). Cumulative weighted degree distribution for Apache (∇), KDE (+) and GNOME (·)



(a) Assortativity



(b) Cumulative weighted degree

CLUSTERING COEFFICIENT IN THE MODULES NETWORK

For the analysis of the clustering coefficient, we have represented its distribution in Figure 3a.

In Table 4 the average distance <*d*> among vertices are represented, together with the average clustering coefficients <*cc*> for our three networks and their equivalent random counterparts (<*rd*> is the random average distance and <*rcc*> is the random average clustering coefficient). As can be observed, the three networks satisfy the small world condition, since their average distances are slightly above those of their random counterparts; but the clustering coefficients are clearly higher.

As can be observed, the average random clustering coefficients for KDE and GNOME are very close to the real ones, due to the high density of those networks. This could be an indication of over-redundancy in their links. That would mean that the same efficiency of information could be obtained with fewer relationships (i.e., eliminating many edges in the network without significantly increasing the diameter or reducing the clustering coefficient). In this sense, the Apache network seems to be more *optimized*. To interpret this fact, the reader may remember that links in this network are related to the existence of common developers for the linked modules. It should be noted that redundancy is probably a good characteristic of a libre software project as it may lose many developers without being affected heavily. It may be especially interesting to have over-redundancy in projects with many volunteers, as in those environments, turnover may be high. Future research should focus on investigating whether over-redundancy is a good or bad parameter in the case of libre software projects. On the other side, how much of this redundancy is due to the taking of a static picture of the project should be researched; it may well be that the redundancy we have observed is the result of different generations of developers working on the same file in different periods of time.

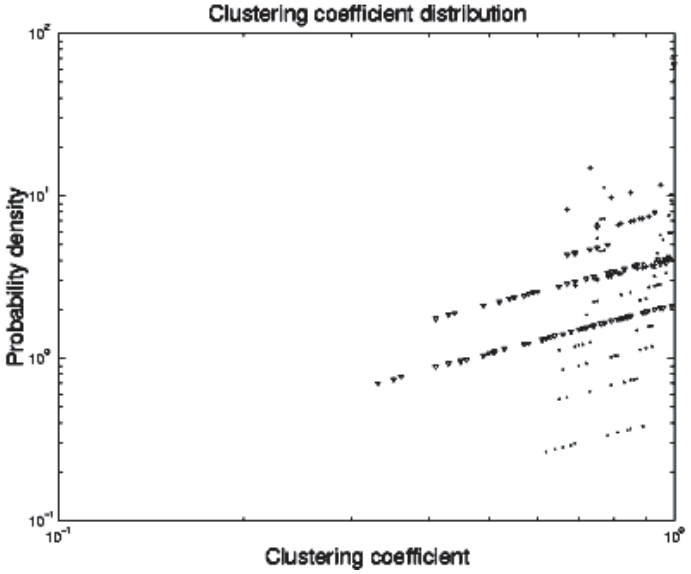
Some interesting conclusions can also be obtained by looking at the weighted clustering coefficient. In Figure 3b we can observe the average weighted clustering coefficient as a function of the degree of vertices (the weighted degree-degree distribution). As we have already noticed, the KDE and GNOME networks have a similar local redundancy, which is higher than the one of Apache. High redundancy implies more fluid information exchanges in the short distances for the first two projects. Besides, the weighted clustering coefficient lowers with the degree in all cases, according to a power law function. We can infer that highly connected vertices cannot maintain their neighbors as closely related as poorly connected ones. This happens typically in most social networks because the cost of maintaining close relationships in small groups is much lower than the equivalent cost for large neighborhoods.

DISTANCE CENTRALITY IN THE MODULES NETWORK

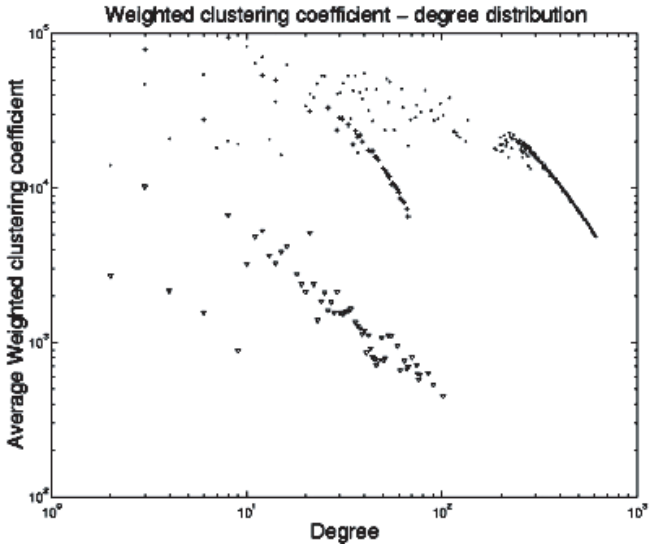
The analysis of the distance centrality of vertices is relevant because this parameter measures how close a vertex is to the rest of the network. In Figure 4a, the distance centrality distribution for our three networks can be observed. They follow multiple power laws, making higher values of the parameter most probable. This is an indication of well structured networks for the fast spread of knowledge and information.

We can also analyze the average distance centrality as a function of the degree (average distance centrality-degree distribution), which is shown if Figure 4b. It can be observed that in all three cases the average distance centrality grows with the degree following, approximately, a power law of low exponent. This means that, in terms of distance centrality, the networks are quite *democratic*, because there is not a clear advantage of

Figure 3. Clustering coefficient distribution for Apache (∇), KDE (+) and GNOME (·). Average weighted clustering coefficient as a function of the degree of vertices for Apache (∇), KDE (+) and GNOME (.)

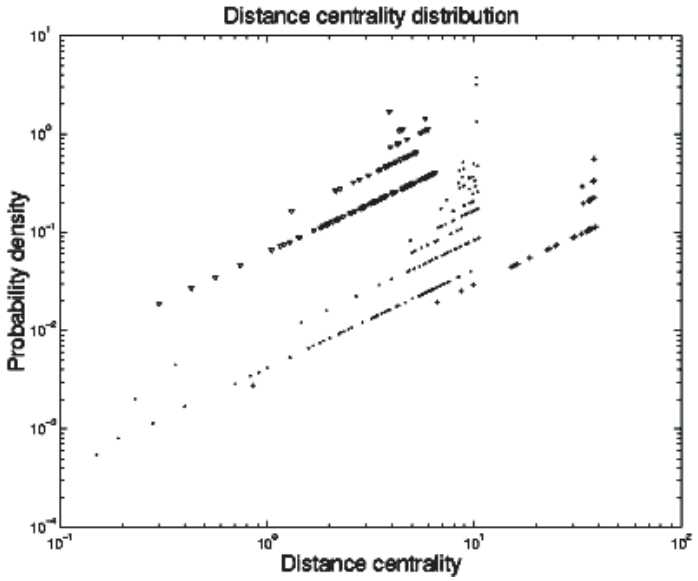


(a) Clustering coefficient distribution

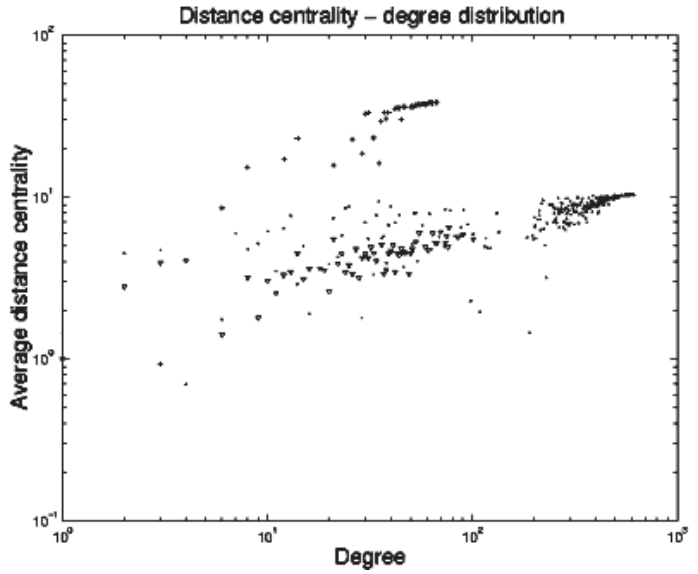


(b) Average weighted clustering coefficient

Figure 4. Distance centrality distribution for Apache (∇), KDE (+), and GNOME (·); Average distance centrality as a function of the degree of vertices for Apache (∇), KDE (+), and GNOME (·)



(a) Distance centrality distribution



(b) Average distance centrality

well connected nodes compared to the rest. Curiously enough, the Apache and GNOME curves are quite similar, while the KDE one is clearly an order of magnitude over the rest. This could be an effect of the lower size of this network, but is also an indication of an especially well structured network in terms of information spread. So, even if KDE showed to be more modular as has been seen for a previous parameter, its structure seems to maximize information flow.

BETWEENNESS CENTRALITY IN THE MODULES NETWORK

The distance centrality of a vertex indicates how well new knowledge created in a vertex spreads to the rest of the network. On the other hand, betweenness centrality is a measurement of how easy it is for a vertex to generate this new information. Vertices with high betweenness centrality indexes are the crossroads of organizations, where information from different origins can be intercepted, analyzed, or manipulated. In Figure 5a, the betweenness centrality distribution for our three networks can be observed. In the same way, this was the case for distance centrality, as it grows following a multiple power law. Nevertheless, there is a significant difference between the distributions of these two parameters. Although the log-log scale of the axis of Figure 5a does not allow visualizing it, the most probable value of the betweenness centrality in all three networks is zero. Just to show an example, only 102 out of 677 vertices of the GNOME network have a nonzero betweenness centrality. So, the distance centrality is a common good of all members of the network, while the betweenness centrality is owned by reduced elite. This should not be surprising at all, as projects usually have modules (i.e., applications) which have a more central position and attract more development attention. Surrounding these modules, other minor modules may appear.

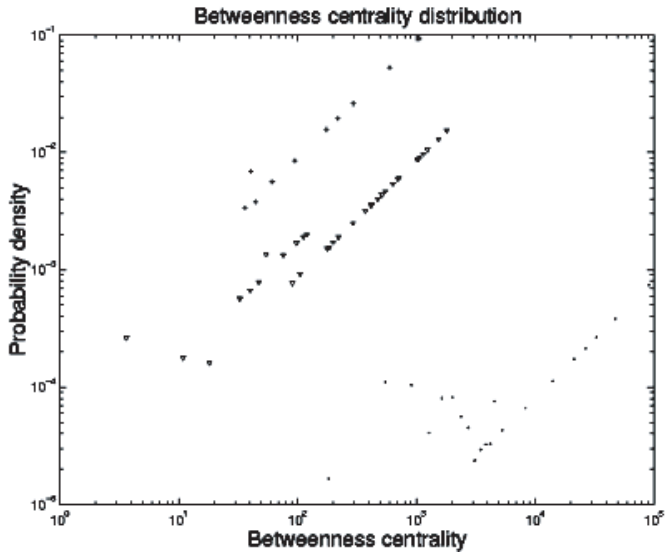
This fact can also be visualized in Figure 5b, where we represent the average betweenness centrality as a function of the degree. It can be clearly seen that only vertices of high degree have nonzero betweenness centralities.

COMMITER NETWORKS

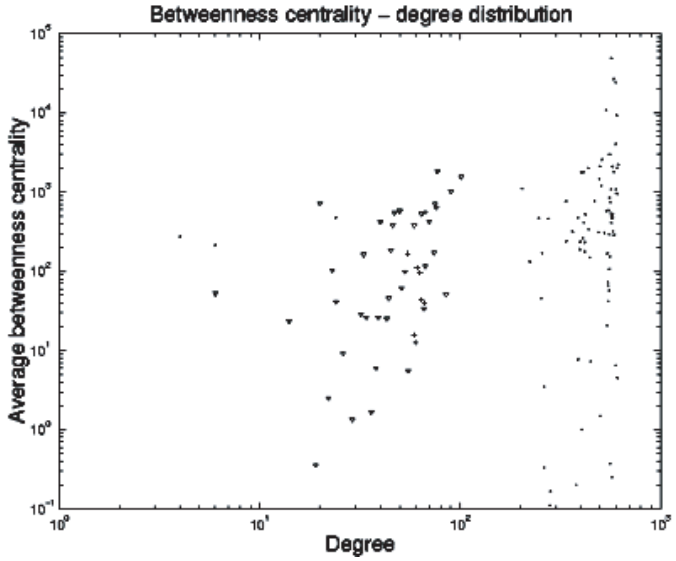
The analysis of commiter networks draw similar conclusions to those shown for module networks, and therefore they are not going to be commented on in detail. For instance, the cumulative degree distribution for the two commiter networks is shown in Figure 6a, which has clearly the same qualitative properties than for this parameter for the module networks shown in Figure 1. The same holds true for the commiter cumulative weighted degree distribution depicted in Figure 6b, or for the average degree as a function of degree, depicted in Figure 7a, where it can be noticed how both networks maintain the elitist characteristic also observed in the case of modules.

An interesting feature of commiter networks can be seen in Figure 7b. The average weighted degree of authors remains more or less constant for low values of the degree. Nevertheless, in the case of KDE, it increases meaningfully for the highest degrees. The implication is that commiters with higher degrees not only have more relationships than the rest, but also their relationships are much stronger than the average. This indicates that authors having higher degrees are more involved in the project development and establish stronger links than the rest. At the same time, as we observed in Figure 7a, they only relate to other commiters that are involved in the project to the same degree as they are. If this behavior is found in other large libre software projects, it could be a valid method to identify the leading “core” group of a libre software project. On the other hand, the Apache project seems to promote a single category of developers, given that the weighted degree does not depend so clearly on the degree of vertices. It

Figure 5. Betweenness centrality distribution for Apache (∇), KDE (+) and GNOME (·). Average betweenness centrality distribution for Apache (∇), KDE (+) and GNOME (·)

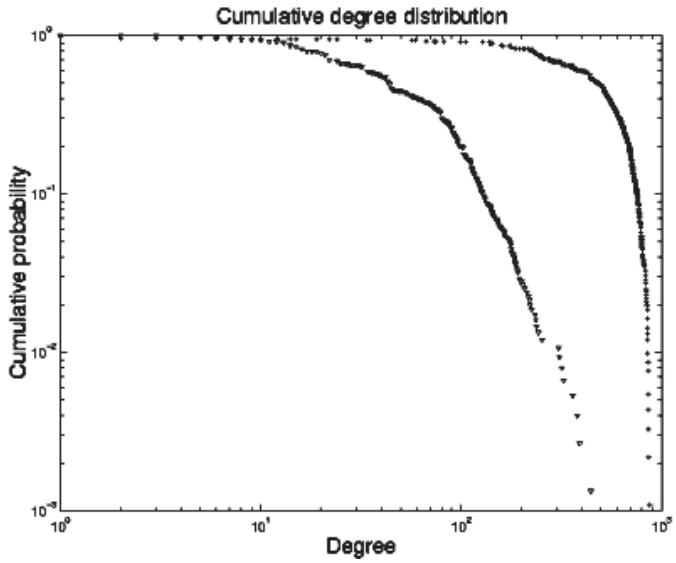


(a) Betweenness centrality distribution

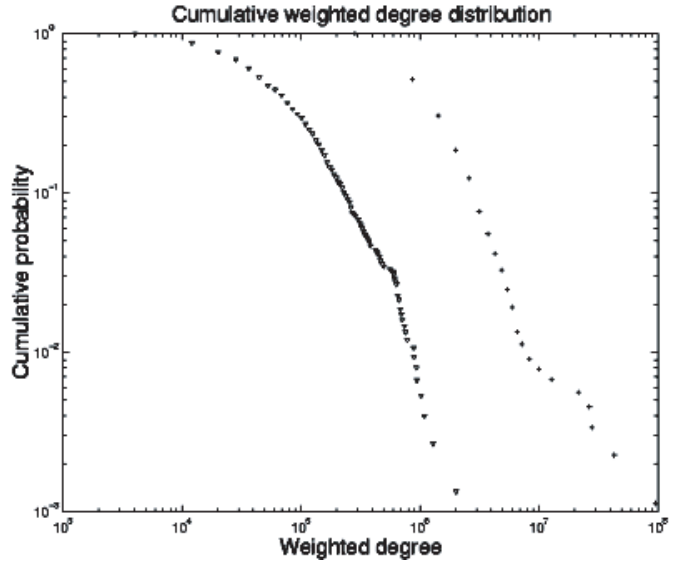


(b) Average betweenness centrality distribution

Figure 6. Cumulative degree distribution for Apache (∇) and KDE (+); Cumulative weighted degree distribution for Apache (∇) and KDE (+)

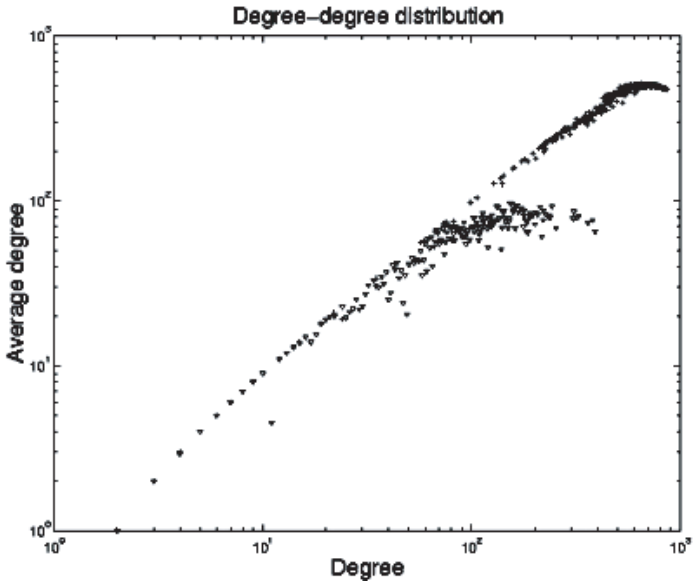


(a) Cumulative degree distribution

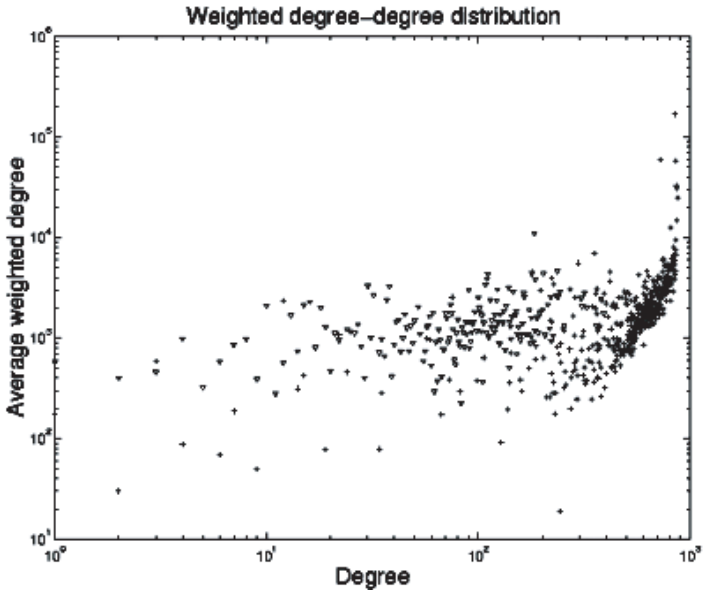


(b) Cumulative weighted degree distribution

Figure 7. Degree - degree distribution for Apache (∇) and KDE (+); Average weighted degree as a function of the degree for Apache (∇) and KDE (+)



(a) Degree - degree distribution



(b) Average weighted degree as a function of degree

may be also that because of the fragmentation of the Apache project in many *families* of modules, it is easy to developers to reach a point where they do not have the possibility to get to know more developers.

Table 5 digs into the small-world properties of commiter networks. As we can observe, both networks can still considered to be small world. The Apache case is especially interesting, because an increase in the average distance is observed. This characteristic plus the large value of the clustering coefficient may indicate that the network is forming cliques.

CONCLUSIONS, LESSONS LEARNED, AND FURTHER WORK

In this article an approach to the study of libre (free, open source) software projects has been presented, based on the quantitative and qualitative application of social networks analysis to the data retrieved from source code management repositories. Since most libre software projects maintain such repositories, and allow for public read-only access to them, this analysis can be repeated for many of them. However, given its characteristics, it will be most useful for large projects, well above the hundreds of thousands of lines of code and dozens of developers.

We have designed a detailed methodology which applies this SNA-based approach to the study of CVS data, and which can be automated. It starts by downloading the required information from CVS, and produces as an output several graphs and tables which can be interpreted to gain knowledge about the informal organization of the studied project. It is important to highlight a set of parameters in the output that are suitable for characterizing several aspects of the organization of the studied project, which makes it possible to gain a lot of insight on how a group of developers is managing coordination and information flow within the project. In addition, it has been shown that the introduction of weights in the relationships gives more realistic information about the projects under study.

It has also been shown how our methodology is applied to some important and well known projects: KDE, Apache, and GNOME. Although these studies are sketched just as case examples, some relevant results can also be extracted from them. For instance, it has been shown how all the networks that have been studied fulfill the requirements to be a small world. This has important consequences on their characterization, since small worlds have been comprehensively studied and are well understood in many respects. We have also not only found that the growth of the studied networks cannot be explained by random preference attachment (something that could be previously suspected). Moreover, it matches pretty well the pattern of preference attachment related to the weight (amount of shared effort) of links. Some other relevant results are the elitist behavior found in these projects with respect to the connectivity of modules and developers, which are indicators of an over-redundancy of links, and of a good structure for the flow of knowledge, and the absence of centers of power (in terms of information flux). All of these conclusions should be validated by studying more projects, and by analyzing with detail their microimplications before being raised to the category of characteristics of libre software projects, but that so far are good lines of further research.

There are some other studies applying similar techniques to libre software projects. For instance, Crowston and Howison (2003) suggests that large projects are more modular than

Table 5. Small world analysis for commiter networks

Project name	<*d*> / <*rd*>	<*cc*> / <*rcc*>
Apache	2.18 / 1.60	0.84 / 0.08
KDE	1.47 / 1.10	0.86 / 0.52

small ones. However, to our knowledge the kind of comprehensive analysis shown in this article has never been proposed as a methodology for characterizing libre software projects and their coordination structure. In fact, after using it in the study of some projects, we believe that it has a great potential to explore informal organizational patterns, and uncovering nonobvious relationships and characteristics of their underlying structure of coordination.

REFERENCES

Albert, R., & Barabasi, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics, 74*, 47–97.

Albert, R., Barabási, A.-L., Jeong, H., & Bianconi, G. (2000). Power-law distribution of the World Wide Web. *Science, 287*.

Anthonisse, J. (1971). The rush in a directed graph (Tech. Rep.). Amsterdam: Stichting Mathemastisch Centrum.

Conway, M. (1968). How do committees invent? *Datamation, 14*(4), 28–31.

Crowston, K., & Howison, J. (2003). The social structure of open source software development teams. In *Proceedings of the ICIS*.

Crowston, K., Scozzi, B., & Buonocore, S. (2003). An explorative study of open source software development structure. In *Proceedings of the ECIS*, Naples, Italy.

Dinh-Trong, T.T., & Bieman, J.M. (2005). The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering, 31*(6), 481–494.

Elliott, M., & Scacchi, W. (2004). Mobilization of software developers: The free software movement (Tech. Rep.). Retrieved June 16, 2006, from http://opensource.mit.edu/papers/elliottscacchi2.pdf

Freeman, C. (1977). A set of measures of centrality based on betweenness. *Sociometry, 40*, 35-41.

Fuggetta, A. (2003). Open source software: An evaluation. *Journal of Systems and Software, 66*(1), 77–90.

Germán, D. (2004a). An empirical study of fine-grained software modifications. In *International Conference in Software Maintenance*.

Germán, D. (2004b). Decentralized open source global software development, the GNOME experience. *Journal of Software Process: Improvement and Practice, 8*(4), 201-215.

Germán, D.M., & Hindle, A. (2006). Visualizing the evolution of software using softChange. *Journal of Software Engineering and Knowledge Engineering, 16*(1), 5-21.

Ghosh, R.A., & Prakash, V.V. (2000). The Orbiten free software survey. *5*(7). Retrieved from http://www.orbiten.org/ofss/01.html

Gîrba, T., Kuhn, A., Seeberger, M., & Ducasse, S. (2005). How developers drive software evolution. In *Proceedings of the International Workshop on Principles in Software Evolution* (pp. 113–122), Lisbon, Portugal.

Godfrey, M.W., & Tu, Q. (2000). Evolution in open source software: A case study. In *Proceedings of the International Conference on Software Maintenance* (pp. 131–142), San Jose, California.

González-Barahona, J.M., López-Fernández, L., & Robles, G. (2004). Community structure of modules in the Apache project. In *Proceedings of the 4th Workshop on Open Source Software*.

Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., & Arenas, A. (2003). Self-similar community structure in a network of human interactions. *Physical Review E 68, 065103(R)*.

Healy, K., & Schussman, A. (2003). *The ecology of open-source software development*. (Tech. Rep.). University of Arizona.

Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12*(1), 27–42.

Krishnamurthy, S. (2002). Cave or community? An empirical investigation of 100 mature open source projects. *First Monday, 7*(6).

Kumar, R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (2002). The Web and social networks. *IEEE Computer, 35*(11), 32–36.

Latora, V., & Marchiori, M. (2003). Economic small-world behavior in weighted networks. *Euro Physics Journal, B32,* 249-263.

Lopez, L., Gonzalez-Barahona, J.M., & Robles, G. (2004). Applying social network analysis to the information in cvs repositories. In *Proceedings of the International Workshop on Mining Software Repositories, 26th International Conference on Software Engineering,* Edinburg, Scotland.

Madey, G., Freeh, V., & Tynan, R. (2002). The open source development phenomenon: An analysis based on social network theory. In *Proceedings of the Americas Conference on Information Systems (AMCIS2002)* (pp. 1806–1813), Dallas, Texas.

Mockus, A., Fielding, R.T., & Herbsleb, J.D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309–346.

Newman, M.E.J. (2001a). Scientific collaboration networks: I. Network construction and fundamental results. *Physical Review, E64, 016131.*

Newman, M.E.J. (2001b). Scientific collaboration networks: Ii. Shortest paths, weighted networks, and centrality. *Physical Review, E64, 016132.*

Raymond, E.S. (1997). The cathedral and the bazar. *First Monday, 3*(3).

Robles, G., Amor, J.J., Gonzalez-Barahona, J.M., & Herraiz, I. (2005a). Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles in Software Evolution* (pp. 165–174), Lisbon, Portugal.

Robles, G., González-Barahona, J.M., & Michlmayr, M. (2005b). Evolution of volunteer participation in libre software projects: Evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems* (pp. 100–107), Genoa, Italy.

Robles, G., Koch, S., & Gonzalez-Barahona, J.M. (2004). Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), 26th International Conference on Software Engineering,* Edinburg, Scotland.

Sabidussi, G. (1996). The centrality index of a graph. *Psychometrika, 31,* 581-606.

Wagstrom, P.A., Herbsleb, J.D., & Carley, K. (2005). A social network approach to free/open source software simulation. In *Proceedings of the 1st International Conference on Open Source Systems* (pp. 100–107), Genoa, Italy.

Watts, D.J. (2003). *Six degrees*. New York: W.W. Norton & Company.

Watts, D.J., & Strogatz, S. (1998). Collective dynamics of small-world networks. *Nature, 393*, 440-442.

Xu, J., Gao, Y., Christley, S., & Madey, G. (2005). A topological analysis of the open source software development community. In *Proceedings of the 38th Hawaii International Conference on System Sciences,* Hawaii.

Zimmermann, T., & Weißgerber, P. (2004). Processing CVS data for fine-grained analysis. In *Proceedings of the International Workshop on Mining Software Repositories* (pp. 2–6), Edinburg, Scotland.

Zimmermann, T., Weißgerber, P., Diehl, S., & Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering, 31*(6), 429–445.

ENDNOTES

* This work has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337. Israel Herraiz has been funded in part by Consejeria de Educación of Comunidad de Madrid and European Social Fund under grant number 01/FPI/0582/2005.

1 In this paper the term "libre software" will be used to refer to any software licensed under terms that are compliant with the definition of "free software" by the Free Software Foundation, and the definition of "open source software" by the Open Source Initiative, thus avoiding the controversy between those two terms.

2 Concurrent Version System (CVS) is the source code management (also known as versioning) system used in most libre software projects, although lately a new generation of tools, including for instance Subversion, are gaining popularity. In those projects, the CVS repository is usually freely readable over the Internet.

3 A Pareto distribution is known to be given when the 20% most active is responsible for 80% of the output.

4 For downloading this information we have used the CVSAnalY tool described in Robles, Koch, and Gonzalez-Barahona (2004).

5 Throughout this article, references to Apache cover all projects lead by the Apache Foundation and not just the HTTPd server, usually known as the Apache Web server.

6 Social network analysis has been applied to networks with hundreds of thousands, sometimes millions of vertices. In this sense, our network is of a small size even if we are handling large libre software projects.

This work was previously published in International Journal of Information Technology and Web Engineering, Vol. 1, Issue 3, edited by E. Damiani; and G. Succi, pp. 27-48, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 5.7
Open Source Software Business Models and Customer Involvement Economics

Christoph Schlueter Langdon
Center for Telecom Management, University of Southern California, USA

Alexander Hars
Inventivio GmbH, Bayreuth, Germany

ABSTRACT

This chapter is focused on the business economics of open source. From a strategic perspective, open source falls into a category of business models that generate advantages based on customer and user involvement (CUI). While open source has been a novel strategy in the software business, CUI-based strategies have been used elsewhere before. Since the success of e-commerce and e-business, CUI-based strategies have become far more prevalent for at least two reasons: Firstly, advances in information technology and systems have improved feasibility of implementation of CUI strategies and secondly, CUI-based economics appear to have often become a requirement for e-business profitability. This chapter presents a review of CUI-based competition, clearly delineates CUI antecedents and business value consequences, and concludes with a synopsis of managerial implications and a specific focus on open source.

INTRODUCTION

Open source software applications and source code are developed cooperatively in an Internet-based peer-to-peer network or community of programmers (Hars, 2002). Some call the open source development process, therefore, also as peer-to-peer production (Wikipedia.org, 2006). The open source model has caught the attention of business strategists and financial analysts (and executives and shareholders of software firms), because open source developers devolve most property rights to the public, including the right to use, redistribute and modify the software free of charge. Some industry observers argue that this approach will emerge as the prevalent way to design and write software; others have been more cautious seeing open source as a niche model (Hars & Ou, 2001; *The Economist*, 2006).

Open source is new and old at the same time. It is a new concept in the software industry. However, the attractiveness of open source is rooted in mechanisms and economics that have fueled

business success in many other areas before. From a business strategy perspective open source fits into a broader category of business models based on customer and user involvement (CUI) that can provide superior economics.

A very visible example of this category of business models is Ikea, the Swedish furniture maker and retailer. Among consumers Ikea is known for its stylish yet affordable furniture. Among some business strategists and researchers Ikea is a prominent example of the economics of customer involvement, which has emerged as a key source of competitive advantage, particularly in the e-commerce area. Broadly speaking customer or user involvement describes a strategy that emphasizes engaging customers and user in business operations.

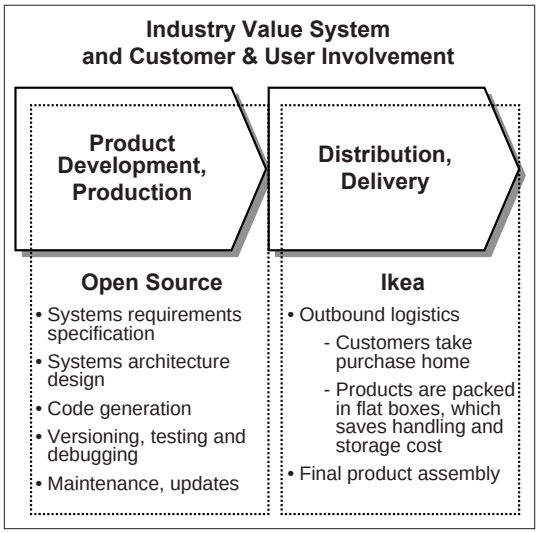
BACKGROUND

"Ikea Economics"

In the case of the Swedish furniture maker and retailer, Ikea, customer involvement is integral to doing business and creating economic advantage. Ikea customers are involved in business operations in that they pick their purchase off the Ikea warehouse shelf, drive it home and assemble it themselves.

Figure 1 depicts a two-tier industry system following Porter's value chain schematic (Porter, 1985). A product has to be developed, made, distributed, sold, and delivered. In the case of Ikea outbound logistics or delivery and final assembly are "outsourced" to the customer (see Figure 1). This saves Ikea cost compared to the competition that sells assembled pieces, which are bulky and, therefore, have to be home delivered. Furthermore, because Ikea furniture is assembled at the final destination, a customer's home, products can be shipped in flat boxes without negative space, which further saves handling and storage cost throughout the entire supply chain and channel system. But the advantage of customer involvement doesn't stop here with merely lower cost. Customer involvement economics can be an enabler of other economic advantages. In the Ikea example, the cost advantage due to customer involvement is used or leveraged by splitting savings with the customer, effectively lowering product prices, often below the price of the competition. The lower sticker price makes stylish design affordable for a larger market, which increasing Ikeas market potential. This larger footprint, in turn, allows Ikea to benefit from another economic advantage, the one that has been the main economic engine of mass production, namely scale economies. In other words, at Ikea customer involvement has worked as a starter to ignite an economies of scale engine. This combination of customer involvement economics and scale economies have helped Ikea become the world's largest furniture maker and retailer with 221 stores in 34 countries as of Spring 2006 (http://franchisor.ikea.com, 3/31/06). It also turned its founder, Ingvar Kamprad, into a multi-billionaire. *Forbes* magazine recently estimated Mr. Kamprad's fortune at $28

Figure 1. Open source and Ikea: Two examples of customer and user involvement



billion—trailing only Microsoft co-founder Bill Gates, U.S. investor Warren Buffett and Mexican industrialist Carlos Slim (Forbes, 2006).

Theory of Customer and User Involvement (CUI)

The literature defines customer involvement as the extent to which a customer is engaged as a participant in business operations, specifically in service production and delivery, including, for example, order processing and account management (Schlueter Langdon, 2003a, 2006). A first research construct has been developed and integrated into a broader theoretic model (see Figure 2; Schlueter Langdon, 2003a, 2006).

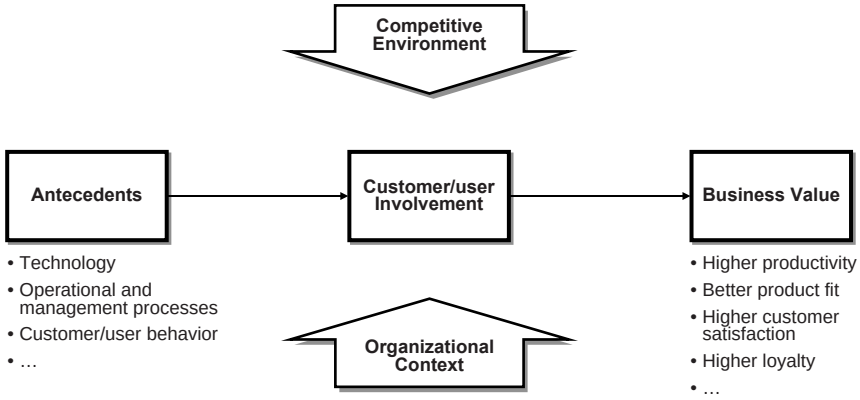
The customer involvement construct and its definition are rooted in several streams in the literature: "customer integration" and "customer relationship management" in marketing, "co-production" and "service encounter management" in service operations research, and "citizen participation" in the public policy literature.

In 1980, Whitaker introduced the notion of "co-production" in public service delivery in the field of public policy management (1980). At the same time Hakansson appears to have introduced the notion of a "customer integration strategy" within the context of marketing strategies in industrial markets, defining it as the ability to adapt to specific customer needs to increase business benefits (Hakansson, 1980, p. 370).

Brown, Raymond, and Bitner (1994) have first explicitly used the phrase "customer involvement" in their categorization of research on service encounters. Brown et al. divided research on service encounters into three primary types, the second of which is focused "on customer involvement in service encounters and the customer's role in service production and delivery" (1994, p. 34). Chase (1978) first discussed a customer's role in the service delivery process. This perspective has been expanded in the service operations literature to also consider the customer as a partial employee (Czepiel, 1990; Bowen, 1986; Kelley, Donnelly, & Skinner, 1990; Mills & Moberg, 1982; Mills & Morris, 1986).

In the marketing literature Sheth and Parvatiyar posited that "relationship marketing attempts to involve and integrate customers, suppliers, and other infrastructural partners into a firm's development and marketing activities. Such involvement results in close interactive relationships with […] customers […]" (Sheth & Parvatiyar, 1995, p. 399). Furthermore, "consumers are increasingly becoming co-producers. […] In many instances,

Figure 2. Theoretic customer and user involvement model



market participants jointly participate in design, development, production, and consumption of goods and services" (Sheth & Parvatiyar, 1995, p. 413). Gruen, Summers, and Acito (2000, p. 36) called this phenomenon "co-production."

The notion of customer integration is presented in the marketing literature as an extension of manufacturer-distributor relationships (Andersen & Narus, 1984, 1990). The theory base that underlies the marketing literature on manufacturer-distributor relationships and, therefore, the argument that customer involvement can enhance business value (see Figure 2) is a synthesis of exchange theory (Kelley & Thibaut, 1978; Thibaut & Kelley, 1959) and transaction cost economics (Williamson, 1975, 1985). Exchange theory states that parties transfer resources in relationships to enhance self-interest, while transaction cost economics reveals conditions under which certain organizational choices can maximize self-interest in the exchange relationship.

Specifically, the literature points to several consumer and seller benefits from tight customer integration. Lovelock and Young (1979) discussed the customer as source for increasing a service firm's productivity. Sheth and Parvatiyar (1995) indicate that consumers benefit from products and services that suit their needs better and sellers from higher customer satisfaction. Higher customer satisfaction in turn is positively related with customer loyalty and market share (Anderson, Fornell, & Lehmann, 1994; Anderson, Fornell, & Rust, 1997).

MAIN FOCUS OF THE CHAPTER

Customer and User Involvement and Business Value Categories

Since the success of the Internet in business, CUI-based strategies have become more prevalent. For one, advances in information technology and many open standards have increased information systems capabilities at lower cost to make CUI-based strategies feasible. For another, CUI economics are often required in the first place in order to make e-business operations profitable, because in electronic commerce companies have become expected to do more for less.

Analysis based on industrial organization theory clearly highlights this more-for-less dilemma. Tracking value chain activities, such as product search, reveals that the Internet-enabled change in the interaction between a consumer (demand side) and vendor (supply side) has led to an extension of the traditional value system (Schlueter Langdon & Shaw, 2000, 2002). In electronic commerce vendors are often doing more than in traditional commerce. Online vendors are supporting activities, which consumers have to perform manually in traditional channel systems. For example, instead of driving to multiple stores, walking up and down the aisles to search for a product and find a low price, shoppers can enter key words and at the push of a button, they can evaluate competing price quotes. Doing more is costly as online sellers resort to "softwarization," the wholesale automation of business transactions and processes using information systems (Schlueter Langdon, 2003b, 2003c). While labor cost may be saved, online vendors have to invest in the design, building and implementation of sophisticated information systems, and they continue to spend money on operations, maintenance and updates. The cost of selling goods online may be cheaper but consumers also expect lower sticker prices online. In order to turn a profit, online vendors often rely on CUI economics. Table 1 provides a systematic overview of major, generic CUI business value categories.

High customer involvement may allow for mass-customization of products and services using customer data or user profiles, which, in turn, may facilitate both—lower cost and higher revenue. To take a real-world example, Dell can leave customization of products (e.g., choice of microprocessor) and product bundling (e.g., PC

Table 1. Major, generic CUI business value categories

Cost	Revenue
Customer or user operates business process activities	
• Company saves employee time and expense • Likely higher fixed cost for IS that can be operated by many customers instead of a few employees only	• Goods can be purchased anytime and from anywhere -> Higher quality, better product fit -> Better customer data
Higher quality, better fit	
• Less inventory in entire channel system • Less slow moving and obsolete items • Less discounts	• Customer likes the feeling of being in control -> Higher customer satisfaction • Monopolistic competition pricing opportunities
Higher customer satisfaction	
• Lower churn saves customer acquisition cost • Positive word of mouth may save marketing expenses	• Higher loyalty • Higher lifetime customer value
Better customer data or "profiles" (behavior, wants and needs)	
• Data mining improves accuracy of targeting customers and saves marketing and sales cost • Lower marketing research cost	• Up-selling opportunities • Better next generation product • User lock-in and higher switching cost

with ink jet or laser printer) to individual preferences, which can increase up-/cross-selling opportunities and customer satisfaction (see Table 1). At the same time Dell can save inventory cost and write-offs, because customers trigger of manufacturing and assembly activities (see Table 1). Instead of Dell pushing products into the market, customers are pulling the product through the system, turning a made to stock system into a made-to-order flow.

CUI is not limited to specific industries, such as consumer products (Ikea and Dell). The auto industry has discovered CUI economics as a source of business advantage. For example, BMW, the German maker of luxury cars, has designed information systems so that European buyers can custom-design their own cars with any change possible until five days before production. As a result, 80% of European BMW buyers custom-design their vehicles and most last minute changes of orders are reportedly upgrades to bigger engines and more luxurious interiors, which tend to be more lucrative for the firm (*Business Week*, 2003). Another CUI example in the auto industry is the emerging area of vehicle relationship management (VRM). Automakers have begun to install black boxes into vehicles that often work similar to flight tracking devices in airplanes. The box is valuable in two ways. Firstly, it provides vehicle usage data, which is a function of vehicle model, the driver and its environment. Secondly, it provides a new, interactive channel system with every customer. The data and the channel can be exploited to better manage customer and vehicle relationships, hence VRM. Vehicle usage data can be exploited for diagnostics purposes to improve uptime. The new channel can be used to interact with customers to improve buyer satisfaction and loyalty. All it takes to unlock the value is user participation.

In the software industry open source software has emerged as an important implementation of CUI economics. Many applications are created by an open source community. Figure 1 illustrates that all essential software development activities of requirements specification, architecture design, code generation, debugging and testing as well

as ongoing maintenance are left to a community of users.

FUTURE TRENDS

Discussion and Managerial Implications

In open source software advantages accrue along all three major dimensions of business performance: cost, time and quality. Figure 3 reveals how an average IT implementation project using open source software compares with a project that is being built traditionally. Results are based on a convenience sample of expert assessments. The business value parameters have been defined at a very high level: cost measures project cost including maintenance, time measures initial implementation as well as downstream modifications, and quality rates the degree of excellence and customer satisfaction.

In terms of cost, open source saves at least the profit or profit margin associated with a brand name product, brand name systems integration service and brand name maintenance contract. (The software business "has an exceedingly high gross [profit] margin of 90%, [...] a net profit margin of 27%. This shows that its marketing and administration costs are very high, while its cost of sales and operating costs are relatively low (McClure, 2004).)

Open source can save time, because documentation is public and exposed to public scrutiny, just like the source code itself (Hars, 2002). Furthermore, customer support is not limited to a vendor's office hours or a particular maintenance subscription level but open source documentation and expertise tends to be available online and anytime.

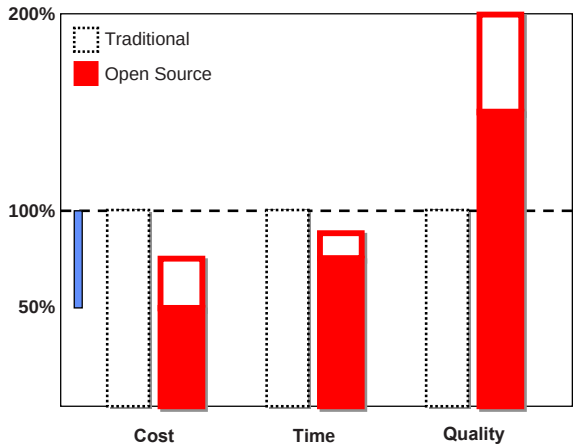
Quality can be better, firstly, because of transparency of the process and secondly, because of transparency of qualification and achievements of contributors (Hars, 2002). This mirrors a key lesson of a free market system, namely that transparency tends to increase buyer value. Also, problems are fixed when a problem exceeds users' willingness to cope and not when decided by a vendor's corporate strategy or business policy.

Figure 3 summarizes the assessment of our convenience sample of experts, which includes senior developers and architects, and information technology executives of Fortune 500 companies (chief information officers, CIOs, and vice presidents).

Figure 3 compares an open source implementation with a traditional, branded solution along the aforementioned and defined business value categories of cost, time and quality. Results reflect a consensus among our experts that open source software beats a traditional solution in any category. The extent of this advantage can vary. First, there is variability within each category. Consider cost: some experts see an OSS implementation at 50% of the cost of a traditional solution. Others see it more at 75%. Second, there is variability across business value dimensions. Higher quality appears to be the most significant advantage, followed by lower cost.

By nature, a high level comparison, such as the one presented in Figure 3, is constrained and

Figure 3. CUI business value assessment: The open source example

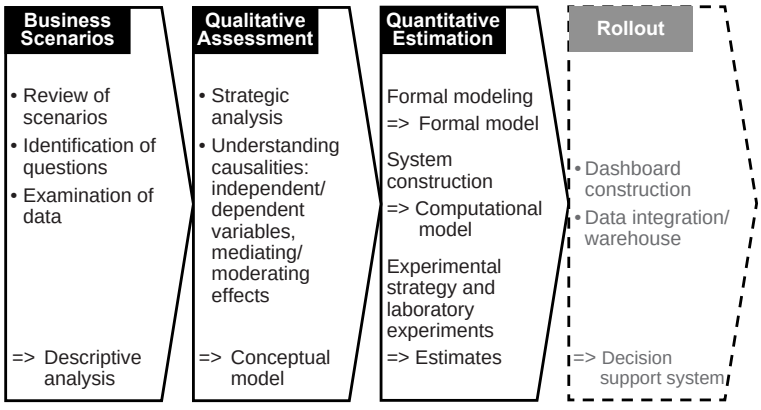


implications are limited. First, this comparison is limited to an implementation that utilizes open source software instead of a commercial package (e.g., installing, configuring, integrating, testing, and maintenance). It does not include the writing of application source code. Furthermore, the comparison is focused on situations in which open source is a true alternative. Second, the business value parameters—cost, time, and quality—can be interdependent and, therefore, difficult to isolate. For example, in order to speed up a project the quality of the code may be compromised; to save money less qualified engineers are used who need more time to write the code, and so on. Expert interviews were conducted in a way that such effect would be additive to the assessments presented in Figure 3. Third, an average project is considered and, therefore, results aim to reflect a *central tendency,* which is useful as a guideline but it obscures the variance in size and complexity of information systems projects. Furthermore, the distribution may be skewed and in this case average values can be easily misinterpreted.

It is understood that a specific evaluation would require a dedicated analysis in order to properly compare alternatives quantitatively. In order to conduct such analysis, a multi-step approach would have to be devised. Figure 4 depicts an exemplary business intelligence analytics schematic derived from research theory (Schlueter Langdon, 2005, 2007).

Central to any business value assessment—and open source is no exception—is the identification of a causal model that underlies everything that follows (see Figure 4, phase two: qualitative assessment -> conceptual model). A causal model represents the most relevant variables and a set of logical relationships between them. It prevents confusing cause and consequences. The business practice of jumping straight into a spreadsheet to calculate a conclusion is a common mistake. No patient would accept treatment without prior diagnosis. By the same token, any reliable and robust business value assessment requires careful separation of independent and dependent variables as well as moderating effects grounded in theory and best practice. While medical doctors are trained extensively to administer diagnosis-based treatment many managers jump straight to actions, often based on gut instinct only. If key variables and cause-result relationships cannot be clearly identified and delineated on a single sheet of paper then it is not plausible that jumping to some spreadsheet-based calculation would suddenly solve the problem.

Figure 4. Business model evaluation method



At the conceptual modeling stage for assessing CUI benefits it is important to:

- Understand under which circumstances customer involvement can create benefits:
  - What are specific customer involvement antecedents?
  - What are key moderating effects (see Figure 2)?
  - Avoid reinventing the wheel and instead use existing theory, best practice and literature in information systems, management and marketing, for example.
- Understand how to leverage CUI economics:
  - Can CUI be leveraged to generate other advantages (see Table 1)?
  - Would it take partners to increase advantages?
- Understand how an incumbent business model may become vulnerable to CUI-based competition from either old rivals or new entrants or both.

Once a model has been designed it can be implemented. Typically, this means constructing a spreadsheet (see Figure 4, phase three: quantitative estimation -> estimates). This is also an opportunity to verify measurements concepts before collecting the required data. Finally, results would have to be evaluated.

CONCLUSION

Aforementioned issues can only be exemplary. Experience suggests that it is often misleading to suggest a generic solution. The business model evaluation method presented in Figure 4 distinguishes between major analytical phases. It would have to be adapted, modified and specified for a given decision problem. However, while actual outcomes may vary, Figure 3 suggests that an open source solution may in any case be an economical choice. This outcome coincidences with the observation that brand name software vendors increase the attractiveness of products that compete with open source packages and/or even offer products in an open source way.

ACKNOWLEDGMENT

The manuscript has benefited from thoughtful suggestions and comments of the many contributors and anonymous reviewers of the Special Interest Group on Agent-Based Information Systems (SIGABIS) of the Association for Information Systems (AIS, www.agentbasedis.org). The authors particularly acknowledge discussions with and advice from Steve Davis, Omar El Sawy, Mark Hayes, Jörg Heilig, Bob Josefek, Ann Majchrzak, Steffen Neumann, Kim Spenchian, and Ed Trainor.

REFERENCES

Anderson, E. W., Fornell, C., & Lehmann, D. R. (1994, July). Customer satisfaction, market share, and profitability. *Journal of Marketing, 56*, 53-66.

Anderson, E. W., Fornell, C., & Rust, R. T. (1997). Customer satisfaction, productivity, and profitability. *Marketing Science, 2*, 129-145.

Anderson, J. C., & Narus, J. A. (1984). A model of the distributor's perspective of distributor-manufacturer working relationships. *Journal of Marketing, 48*, 62-74.

Anderson, J. C., & Narus, J. A. (1990). A model of distributor firm and manufacturer firm working relationships. *Journal of Marketing, 54*, 42-58.

Bowen, D. E. (1986). Managing customers as human resources in service organizations. *Human Resource Management, 25*(3), 371-383.

Brown, S. W., Raymond, P. F., & Bitner, M. J. (1994). The development and emergence of service marketing thought. *International Journal of Service Industry Management, 5*(1), 21-48.

Business Week. (2003, June 9). *BMW's labor practices are cutting-edge, too.* Retrieved August 6, 2003, from http://www.businessweek.com

Chase, R. (1978, November-December). Where does the customer fit in a service operation? *Harvard Business Review*, 138-139.

Czepiel, J. A. (1990). Service encounter and service relationships: Implications for research. *Journal of Business Research, 20*(1), 13-21.

The Economist. (2006, March 18). Open, but not as usual. *Special Report: Open-source business*, 73-75.

Forbes. (2006). *The world's billionaires.* Retrieved March 31, 2006, from http://www.forbes.com/billionaires

Gruen, T. W., Summers, J. O., & Acito, F. (2000, July). Relationship marketing activities, commitment, and membership behavior in professional associations. *Journal of Marketing, 64*, 34-49.

Hakansson, H. (1980). Marketing strategies in industrial markets: A framework applied to a steel producer. *European Journal of Marketing, 14*(5,6), 365-378.

Hars, A. (2002). Open source software. *WISU, 4*, 542-551.

Hars, A., & Ou, S. (2001). Working for free? Motivations for participating in open source projects. *International Journal of Electronic Commerce, 6*(2), 25-39.

Kelley, H. H., & Thibaut, J. W. (1978). *Interpersonal relations: A theory of interdependence*. New York: John Wiley & Sons.

Kelley, S. W., Donnelly, J. H., & Skinner, S. K. (1990). Customer participation in service production and delivery. *Journal of Retailing, 66*(3), 315-335.

Lovelock, C. H., & Young, R. F. (1979, May-June). Look to consumers to increase productivity. *Harvard Business Review*, 168-178.

McClure, B. (2004, April 28). The bottom line on margins. *Investopedia.com*. Retrieved March 31, 2006, from http://www.investopedia.com

Mills, P. K., & Morris, J. H. (1986). Clients as partial employees of service organizations: Role development in client participation. *Academy of Management Review, 11*(4), 726-735.

Mills, P. K., & Moberg, D. J. (1982). Perspectives on the technology of service operations. *Academy of Management Review, 7*(3), 467-78.

Porter, M. E. (1985). *Competitive advantage: Creating and sustaining superior performance*. New York: The Free Press.

Schlueter Langdon, C. (2007). Instrument validation for strategic business simulation. In V. Sugumaran (Ed.), *Application of agent and intelligent information technologies* (pp. 108-120). Hershey, PA: Idea Group Publishing.

Schlueter Langdon, C. (2003a). Linking IS capabilities with IT business value in channel systems: A theoretical conceptualization of operational linkages and customer involvement. In *Proceedings of WeB December 2003*, Seattle, WA (pp. 259-270).

Schlueter Langdon, C. (2003b, June). IT matters. In Does IT Matter? An HBR Debate. *Harvard Business Review*, 16. Retrieved from www.hbr.org

Schlueter Langdon, C. (2003c). Information systems architecture styles and business interaction patterns: Toward theoretic correspondence. *Journal of Information Systems and E-Business, 1*(3), 283-304.

Schlueter Langdon, C. (2005). Assessing economic feasibility of e-business investments [White Paper Version 3.0]. Redondo Beach, CA: Pacific Coast Research.

Schlueter Langdon, C. (2006). Designing information systems capabilities to create business value: A theoretical conceptualization of the role of flexibility and integration. *Journal of Database Management, 17*(3), 1-18.

Schlueter Langdon, C., & Shaw, M. J. (2000). The online retailing challenge: Forward integration and e-backend development. In *Proceedings of ECIS July 2000 Conference*, Vienna, Austria (pp. 1025-1028).

Schlueter Langdon, C., & Shaw, M. J. (2002). Emergent patterns of integration in electronic channel systems. *Communications of the ACM, 45*(12), 50-55.

Sheth, J. N., & Parvatiyar, A. (1995). Relationship marketing in consumer markets: Antecedents and consequences. *Journal of the Academy of Marketing Science, 23*(4), 255-271.

Thibaut, J. W., & Kelley, H. H. (1959). *The social psychology of groups.* New York: John Wiley & Sons.

Williamson, O. E. (1975). *Markets and hierarchies: Analysis and antitrust implications.* New York: The Free Press.

Williamson, O. E. (1985). *The economic institutions of capitalism.* New York: The Free Press.

Whitaker, G. (1980, May-June). Co-production: Citizen participation in service delivery. *Public Administration Review,* 240-242.

KEY TERMS

Business Intelligence Analytics: Summarizes models and methods used to analyze data for the purpose of helping executives make better, more precise decisions.

Business Model: Describes how profit is generated; captures business logic by separating independent/dependent variables and mediating/moderating effects.

Co-Production: Has evolved to describe a situation in which people outside paid employment, such as customers, contribute to business value-added.

Customer and User Involvement: Describes the extent to which a customer is engaged as a participant in business operations, specifically in service production and delivery, including, for example, order processing and account management.

Customer Relationship Management (CRM): A broad term to cover concepts, methods, and procedures, and enabling information technology infrastructure that support an enterprise in managing customer relationships.

IT Business Value: Captures the business value derived from investments in information technology components and systems. Generic IT business value categories include cost, revenue, and quality.

Peer-to-Peer Production: Describes work performed and organized through the free cooperation of equals.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St. Amant; and B. Still, pp. 522-531, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 5.8
Investing in Open Source Software Companies:
Deal Making from a Venture Capitalist's Perspective

Mikko Puhakka
Helsinki University of Technology, Finland

Hannu Jungman
Tamlink Ltd., Finland

Marko Seppänen
Tampere University of Technology, Finland

ABSTRACT

This chapter studies how venture capitalists invest in open source-based companies. Evaluation and valuation of knowledge-intensive companies is a challenge to investors, and while many methods exist for evaluating traditional knowledge-intensive companies, the rise of open source companies with new hard-to-measure value propositions such as developer communities brings new complexity to deal-making. The chapter highlights some experiences that venture capitalists have had with open source companies. The authors hope that the overview of venture capital process and methodology as well as two case examples will provide both researchers and entrepreneurs new insights into how venture capitalists work and make investments.

INTRODUCTION

In the traditional view, the evolution of a technology-based new company is seen through separate consecutive stages. Business is based on creating tangible real assets; and in the end, the value of a company is also based on real assets. First, the technology is developed, which is followed by setting up the organization. Once the organization has reached a sufficient scale, internationalization is started. Finally, the value of the company is estimated with potential venture investment or through realization either through an initial public offering (IPO) or a trade sale.

However, due to the increased complexity of products and services, time-to-market tends to lengthen. In order to maintain sufficient resources until the company reaches profitability, external financing is needed. The time needed to turn a

company's cash flow positive varies considerably. A long product development phase and slow market penetration prolong the period of negative cash flow. Simultaneous internationalization drains resources at an even higher rate. Since start-ups do not usually have collateral to secure bank loans, equity financing is the most evident form of financing. Venture capital funding is usually sought in order to get business development support in addition to plain financing.

New business ideas are increasingly more knowledge intensive, driven in part by the application of ICT as an enabling technology across industrial sectors. Also, the nature of business has changed: times-to-market are faster, development stages are no longer consecutive but can be simultaneous or even skipped, and companies are born global. Distinct from yesterday's industrial companies, today's knowledge-intensive companies' values are not based on their real assets but rather on their intangible assets such as knowledge, networks, and brand. Needless to say, intangible assets are considerably more challenging to value. The previous is even truer in the case of open source software (OSS) companies, since part of their business (and value) relies on open source (OS) communities in which people contribute their time and knowledge voluntarily into projects. Contributions are real but take place without formal contracts or incentive mechanisms, and people can easily abandon the community.

Furthermore, OSS companies that build their businesses on OS products (e.g., Google, JotSpot) have huge savings in time and licensing fees; they get to market faster and cheaper. This sets even greater challenges for those valuating OSS companies. In theory, these free contributions should yield in higher valuations. On the other hand, the uncertainties involved should have the opposite effect.

The mission of this study is to compare traditional IT companies and their valuations and evaluations to those of OSS companies from the viewpoint of the venture capitalist. This is further divided into several subquestions:

- What are the special issues to be taken into account when evaluating OSS companies?
- Do venture capitalists assign a positive, negative, or no value to OSS companies and their communities when compared to traditional IT companies?
- Is there hype around OS?

Data for recent valuations of OSS and traditional IT companies were gathered from the VentureOne database. VentureOne (2005) is one of the leading venture capital research firms offering information on the venture capital industry. To better understand the investment decisions made and valuations paid for OS companies, and in order to get insights into what are the specialties in evaluation of OSS companies, two case studies were carried out. When designing the case study, based on the authors' initial understanding of the issues at hand, a pattern of interview questions was constructed. In addition to these semistructured interviews, data were gathered from publicly available sources. The interviewees were key managers of the case companies. Both of the cases present seed/early-stage venture capitalists that have been active in investing in OS companies. In addition, the case studies were backed up with several interviews with venture capitalists and entrepreneurs as well as feedback gathered from Internet online communities (for the questionnaire used, see Puhakka & Jungman, 2005).

BACKGROUND

Earlier Research on Evaluation and Valuation Theory

Venture capitalists evaluate their investment opportunities based on certain criteria. It is widely accepted that the three key investment decision

criteria are management team, market projections, and product (Tyebjee & Bruno, 1981, 1984; MacMillan, Siegel, & Narasimha 1985).

In addition, venture capitalists have preferences, such as a venture's stage of development, its location, its industry or technology, and size of the investment required, which vary among one another (Seppä, 2000). These criteria and preferences are related to evaluation of an investment opportunity: does the venture have potential? Is it worth our time and money? Does it fit our investment strategy? Venture capitalists base their evaluation on business plans, meetings with the entrepreneurial team, and various researches.

Only after positive results from evaluation is it time to think about the value of the company. The process of valuation resembles business negotiation. Herein, "valuation means the process of placing a monetary value on an investment opportunity" (Seppä, 2003, p. 6). Venture capital valuations are not as straightforward as public market valuations or share prices. "Because of the fluctuations in the supply and demand of venture capital, investment valuations are not always determined according to the rules of efficient markets" (Seppä, 2003, p. 11). Valuation also can refer to venture capital funds' periodic valuations of investments (Association Française des Investisseurs en Capital [AFIC], British Venture Capital Association [BVCA], & European Private Equity and Venture Capital Association [EVCA], 2005).

Valuation of high-tech companies by venture capitalists theoretically has been studied extensively (e.g., Lockett, Wright, Sapienza, & Pruthi, 2002; Seppä, 2003). The value of a new venture is derived by discounting predicted future cash flows to the present. The discounting factor depends on the probability of returns. Even if a company has significant potential future cash flows, the risk of failure decreases its net present value.

Different methodologies exist in the valuation, but all aim at answering the same question: what is the present value of expected future earnings or the exit value of a company? The methods fall into the following four categories (Lockett et al., 2002):

1. Liquidation value-asset-based methods
2. Discounted cash-flow-based methods
3. Options-based valuation methods
4. Rule-of-thumb valuation methods (comparator valuations)

The concepts of present value and net present value (NPV) form the basis for the valuation of real assets and investment decision-making. Essentially, the method makes a comparison between the cost of an investment and the net present value of uncertain future cash flows generated by the venture. There are at least four major steps in a discounted cash flow for a proposed venture.

First, assuming that the venture is all equity financed (i.e., all necessary capital is provided by the shareholders), forecasts are needed for what the expected incremental cash flows would be to the shareholders if the venture were accepted.

Second, an appropriate discount rate should be established that reflects the time value and risks of the venture, which, therefore, can be used for the calculation of the present value of expected future cash flows. The concept of present value includes the notion of the opportunity cost of capital. The appropriate discount rate, or the cost of capital, first must compensate shareholders for the foregone return they could achieve on the capital market by investing in some risk-free assets. It also has to compensate them for the risk they are undertaking by investing in this project rather than in a risk-free financial asset. Thus, the cost of capital is determined by the rate of return investors could expect from an alternative investment with a similar risk profile. Fortunately, the rich menu of traded financial assets provides venture fund managers with the opportunity to estimate the right price.

Third, based on the value additives of present values, the NPV of the venture is to be calculated.

Once the cash flow forecasts are finalized and the appropriate discount rate is established, the calculation of the venture's NPV is a technical matter. When all future cash flows that need to be discounted arrive at their present values, and by adding them to the present value of the necessary capital outlay, the NPV of the venture is achieved.

Finally, a decision has to be made whether to go ahead with the venture or not. As the company proceeds toward profitability, the likelihood of success grows, and the value of the company grows. Thus, it can be argued that every step a company takes toward its goals increases its value.

Exit valuations of technology companies are dependent on the prevailing market situation. Because the presumed exit valuation is the most important measure when considering the value of a company at the last venture capital round before an IPO, it is obvious that exit valuations have significant effects on valuations at all investment rounds, although the effect diminishes toward the founding stage. Due to dramatic changes in exit valuations (e.g., during 1999-2000), there has been a wide variation in valuations at various venture capital rounds as well.

Hype and Uncertainties Vitiate the Theory

Every now and then, things get out of hand. In the 1990s, it was argued that revenues and earnings were neither sufficient nor relevant ways to put value to emerging e-businesses or dot-coms that had no revenues and actually no existing mechanisms of extracting payments from customers. A way to assign value to a member in a Web community was proposed: a so-called "lifetime value of a customer" or a "price-to-eyeball multiple," an estimate of how much on average a customer would end up paying to a company (Valliere & Peterson, 2004).

Emerging OS companies face a similar challenge since part of their businesses (and values) relies on OS communities in which people contribute their time and knowledge voluntarily. Contributions are real but take place without formal contracts or incentive mechanisms, and people can easily abandon the community. The question rises how one should value community contributions like these. The International Private Equity and Venture Capital Valuation Guidelines (AFIC, BVCA, & EVCA, 2005) provide no aid on this. On the other hand, venture capitalists certainly have some views, since there are already several cases in which they have invested in OS companies.

Every venture capital (VC) investment is difficult to value due to the high degree of uncertainty in the performance. The valuation of OS companies is even more challenging, as there is yet neither history nor guidelines due to the uncertainties, for example, in the following:

- Profitability of business model
- Revenue streams
- Market acceptance
- Community commitment
- Competitive reactions
- Quality of software
- New General Public License (GPL) version in 2007

The list includes similar uncertainties that were involved in the dot-com bubble (Valliere & Peterson, 2004). Indeed, one can see the signs of hype in OS as well. Signs of hype surround certain companies (company hype), the OS market (market hype), and the activity of other investors as a group (investor hype) (Valliere & Peterson, 2004). In 2005, OS was getting increasing attention in the press, and venture capitalists were announcing OS strategies. However, it is too early to say whether this will lead to unreasonable valuations of OS companies.

MAIN FOCUS OF THE CHAPTER

Done Deals and Given Valuations

So far, the OS experience has not been a happy one for venture capitalists. According to the research firm VentureOne, some $714 million was invested in 71 OS companies in 1999-2000, and most of those projects collapsed (VentureOne, 2005). One of the biggest successes that is left of those experiences is RedHat Inc., which went public in 1999 and makes money selling enhancements and maintenance services to corporations using Linux OS operating systems. However, it still has some ways to go before reaching $200 million in revenues (RedHat, 2005) and is a relatively mild success with market value less than $5,000 million and earnings per share (EPS) of $0.33 (NASDAQ, 2005). So this is certainly no Google with market value just under $70,000 million (EPS $5.02) or eBay with market value just under $60,000 million (EPS $0.78) (NASDAQ, 2005) that aggressive venture capitalists often use as a reference as companies they want to fund as the "next big thing."

The biggest success so far with OS ventures, as they traditionally have been viewed, has been IBM's Linux service business that the company has grown as a separate emerging business opportunity unit and has managed to grow it from $0 to more than $2 billion in revenues in just 5 years. Still, there is no public record on how much IBM has invested in this venture to realize that growth (IBM, 2005).

Several studies have pointed out that Linux, Apache, and MySQL, for example, have reached the maturity in which the technology or code is comparable or even superior to the existing proprietary ones. Furthermore, for example, Firefox has managed to take the market by storm extremely quickly without any significant marketing budget. In other words, early evidence seems to point to the OS approach, at least some cases, as an efficient way to develop technology and take that to market. However, at least the experiences from the first round financings of OS companies indicate that it is not necessarily the best way to do business.

After a few years of trying to figure out whether money can be made by OS companies, the answer from venture capitalists seems again to be a reluctant yes. Twenty OS businesses raised $149 million in venture money in 2004 in the United States alone (VentureOne, 2005). There are no numbers available for the rest of the world, but in Europe, several investments took place. Looking at that total, it would seem that most of the investments are still on a seed or first-round level (compared to an average level on various rounds of realized investments); if distributed evenly among companies, the amount would be $7.45 million.

Case: BlueRun Ventures

BlueRun Ventures was originally launched as Nokia Venture Partners in 1998 with $150 million initial invested capital from Nokia Corporation. Even with money from Nokia, it was designed right from the start to act independently of its only investor. It raised a second fund of $500 million in 2000, which then already included other investors besides Nokia, such as Goldman Sachs.

In 2005, Nokia Venture Partners raised its third fund of $350 million and changed its name to BlueRun Ventures. Today, BlueRun Ventures has offices in nine locations globally and manages $1 billion making investments into IT, mobile, and consumer technologies at seed- and early-stage levels (BlueRun Ventures, 2005). In looking at OS investment opportunities, the key issue identified by BlueRun Ventures is a strong community close to the company. In their view, OS is a transformation force that is forcing a unit price down and the only realistic counterforce to big incumbent companies such as Oracle or BEA systems.

Still, they consider the market being at an early stage of deployment since after the bubble there have been no notable initial public offerings by OS companies. From the investment point of view, they consider two uncertainties in OS: the size of the market and the fragile business attachment of dealing with a community.

BlueRun Ventures has quite a bit of experience dealing with OS companies and has looked at about a hundred companies from 2002 to 2005. However, in early 2006, it had just completed its first investment in this space. Seed and early-stage investments are tricky, as typically there is very little or no historical numbers to look at. As the company's partner noted, it really is not a science but rather a very subjective opinion of opportunities. The questions are typically, "Do I like this opportunity? How much money is needed to make it happen? Does it fit with the funds strategy?" After that, the actual valuation is actually based on negotiations, which rely more on people skills than anything else (A. Kokkinen, personal communication, November 11, 2005).

From BlueRun Ventures' perspective, valuations in the long run should be the same for both traditional and OS startups. However, the nature of seed-investments is different since communities in a way have taken care of development that is typically done with seed money, resulting in a technology but not in protected intellectual property rights (IPRs).

As the market is still developing, BlueRun Ventures has not been able to identify any OS-dedicated venture funds, even though it expects several of those to be formed. A prerequisite for an OS fund may be that first there should be four to five initial public offerings, which would give enough evidence to the managers of funds in order to go to their investors and propose an OS fund (A. Kokkinen, personal communication, November 11, 2005). How this will turn out remains to be seen. Either OS will remain part of existing funds' investment targets, or OS-dedicated funds will be seen in the future. The latter would obviously result in more sophisticated ways of evaluating OS; otherwise, the competition for investors' money will continue to be played out between traditional software companies and OS companies in mutually accepted terms.

Case: Nexit Ventures

Nexit Ventures is a Finnish-based traditional venture capital company. It raised its Euro 100 million fund in 2000, which was later reduced to Euro 66.3 million. The investors are private institutions with 50% of their commitments outside Finland. The initial focus was seed and early-stage companies both in the Nordic and North America; later this was modified to early- and later-stage companies in the same geographical regions. The technology focus of mobile and wireless communication, from core components and enabling middleware to applications and services, has remained the same.

Nexit Ventures does not consider a pure OS company to be a viable investment opportunity. Rather, it sees the OS approach of collaborative effort to solve various issues to be an enabler for various things of potentially great value. For example, Apple's iPod makes it easy for consumers to utilize music downloaded from Web, whether the music is from legitimate sources or not. Still, the idea was that the closer one gets to the core of OS (the community), the harder it is to make money. Nexit Ventures considers OS be at every level of deployment from early adoption to maturity; it just is not always very visible, and there are legal uncertainties.

From Nexit Ventures' point of view, it is somewhat isolated in Finland about what is taking place globally (a bit paradoxical, as most things are said to take place on the Internet), and it has not yet seen a rise of OS-based businesses, often referred as Web 2.0 companies. There has been little discussion on the public media, especially compared to the United States. In the United States, where valuations are very high again,

due diligence in follow-up rounds is quite weak, according to Nexit Ventures, as there is pressure to do hard sought deals.

Regardless, looking at opportunities and valuating them, Nexit Venture's comments corroborate those of BlueRun Ventures. The markets for venture capital investments are imperfect and always will be. Therefore, the valuations are not made with transparent scientific methods but rather are results of negotiations. In other words, it can be argued that the potential of one's business idea opens the door to negotiations with the venture capitalist, but the valuation that will take place with the investment is determined "by one's skills as a negotiator, that are impossible to quantify or to break down into a scientific model" (A. Tarjanne, personal communication, November 30, 2005).

FUTURE TRENDS

It might be that in the end, the biggest successes to financing community come from and to companies that are not really OS companies as such but rather use OS components to build new businesses; for example, Google, which, like most Web companies, was built on top of OS). From $1 million initial seed capital in 1998 and an injection of $25 million growth capital in 1999, the company realized the value to its investors in 2004 by going public, and by spring 2005, it had surpassed the Finnish pride Nokia with more than $80 billion in market capitalization, compared to just less than $80 billion for Nokia (Google, 2005).

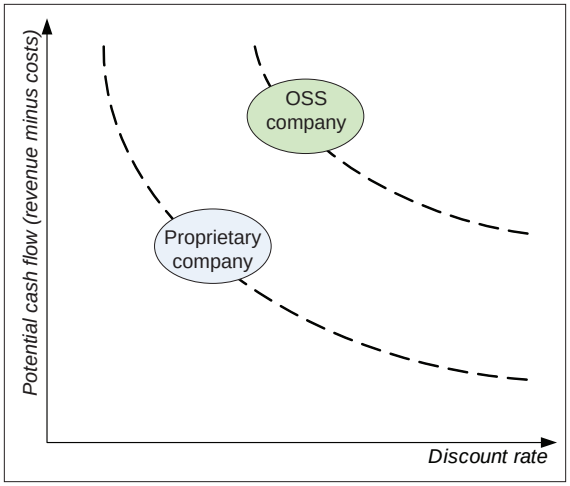
As stated earlier, lareg amounts of money are invested into OS businesses, and we expect dedicated OS funds to be formed in the near future. The key driver will be successful exits from OS investments. However, the first bets (i.e., seed round investments) to potential future successes have just been made, and how successful those will be can only be known in the coming years. Once we can get significant amounts of data, interesting quantitative comparisons can be made between investments in OSS and traditional IT companies.

CONCLUSION

Venture capitalists do not seem to put special value on OS companies. However, some of them recognize that there are distinctly different elements in evaluating OS companies. For instance, expected cash flows are likely to be bigger in businesses built on OS software than in similar traditional software companies, due to the savings, for instance, in licensing fees. Concurrently, the uncertainties in OS should increase the discount rate (see Figure 1).

In interviewing the selected experts and looking at the selected cases, it seems that rather than putting effort into further understanding valuation methodologies, entrepreneurs should seek help in learning better negotiation skills. However, in the academic world, more complex approaches have been taken in valuating a company. It might be

Figure 1. Potential cash flow and risk measured by discount rate of the companies using OS or proprietary software (Source: Adapted from W. Cardwell, personal communication, November 15, 2005)



appropriate to ask whether the academics are really serving the industries if these methodologies are not actually used by the people in the venture capital industry.

The good news for entrepreneurs looking to launch new OS ventures is that money is available, and investors are making their bets again on OS. Still, the basic dilemma remains: while the venture capitalist is looking to become a shareholder as cheaply as possible, the entrepreneur, of course, is trying to retain as much ownership as possible. This would not be an issue if there were a transparent, objective way to estimate the value of the venture. However, as one interviewee said, this is not likely to happen, as the venture capital market remains imperfect. Unfortunately, there are many unknown factors affecting the present value of a startup that have to be estimated, and thus, objectivity is hard to maintain.

ACKNOWLEDGMENT

The authors wish to express their gratitude to interviewed investors and other persons who have expressed their valuable opinions in various forums. An earlier version of this chapter (Puhakka & Jungman, 2005) was presented at the eBRF 2005 Conference in Tampere, Finland, September 26-28, 2005, and published in the *Conference Proceedings, Frontiers of e-Business Research (FeBR 2005).*

REFERENCES

Association Française des Investisseurs en Capital (AFIC), British Venture Capital Association (BVCA), & European Private Equity and Venture Capital Association (EVCA). (2005). *International private equity and venture capital valuation guidelines.* Retrieved September 16, 2005, from http://www.privateequityvaluation.com

Lockett, A., Wright, M., Sapienza, H., & Pruthi, S. (2002). Venture capital investors, valuation and information: A comparative study of the U.S., Hong Kong, India and Singapore. *Venture Capital*, *4*(3), 237-252.

MacMillan, I. C., Siegel, R., & Narasimha, P. N. S. (1985). Criteria used by venture capitalists to evaluate new venture proposals. *Journal of Business Venturing*, *1*, 119-128.

NASDAQ. (2005). *The NASDAQ stock market.* Retrieved February 14, 2006, from http://www.nasdaq.com

Nexit Ventures. (2005). *Nexit Ventures Web site*. Retrieved November 28, 2005, from http://www.nexitventures.com

Puhakka, M., & Jungman, H. (2005). Evaluation and valuation of open source software companies: A venture capitalist' perspective. In M. Seppä, M. Hannula, A.-M. Järvelin, J. Kujala, M. Ruohonen, & T. Tiainen (Eds.), *Frontiers of e-business research 2005* (pp. 855-865). Tampere, Finland: Tampere University of Technology & University of Tampere.

RedHat Inc. (2005). *RedHat Inc. Web site*. Retrieved December 3, 2005, from http://www.redhat.com

Seppä, M. (2000). Strategy logic of the venture capitalist. *Jyväskylä Studies in Business and Economics 3.* Jyväskylä: University of Jyväskylä.

Seppä, T. (2003). *Essays on the valuation and syndication of venture capital investments.* Doctoral dissertations. Helsinki University of Technology, Helsinki, Finland.

Tyebjee, T., & Bruno, A. (1984). A model of venture capitalist investment activity. *Management Science*, *9*, 1051-1066.

Tyebjee, T., & Bruno, A. (1981). *Venture capital decision making: Preliminary results from*

three empirical studies. Wellesley, MA: Babson College.

Valliere, D., & Peterson, R. (2004). Inflating the bubble: Examining dot-com investor behaviour. *Venture Capital, 6*(1), 1-22.

VentureOne. (2005). *VentureOne Web site*. Retrieved September 16, 2005, from http://www.ventureone.com

KEY TERMS

Evaluation: Subjective and qualitative assessment of an investment opportunity.

Proprietary: Belonging to or controlled by an individual or organization that has the ability to share that item (in this case, software code) with others.

Seed Company: Company in a stage of research, assessment, and development of an initial concept before reaching the start-up phase (FVCA Yearbook, 2004).

Startup Company: Company in a product development stage requiring further funds to initiate commercial manufacturing and sales (FVCA Yearbook, 2004).

Valuation: Process of placing a monetary value on an investment opportunity (Seppä, 2003).

Venture Capital: Equity investments made for the launch, early development, or expansion of a business (EVCA, 2005, www.evca.com).

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St. Amant; and B. Still, pp. 532-540, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 5.9
Open Source Software:
A Developing Country View

Jennifer Papin-Ramcharan
The University of the West Indies – St. Augustine Campus, Trinidad and Tobago

Frank Soodeen
The University of the West Indies – St. Augustine Campus, Trinidad and Tobago

ABSTRACT

This chapter presents issues that relate to developing countries' use of open source software (OSS) and the experience of these countries with OSS. Here the terms open source software (OSS), free/libre and open source software (FLOSS) and free software (FS) are used interchangeably. It describes the benefits of FLOSS including its superior quality and stability. Challenges to FLOSS use particularly for developing countries are described. It indicates that despite the greater benefits to developing countries of technology transfer of software development skills and the fostering of information and communication technology (ICT) innovation, the initial cost of acquiring FLOSS has been the key motivation for many developing countries adopting FLOSS solutions. It illustrates this by looking at the experience of a university in a developing country, The University of the West Indies, St. Augustine Campus in Trinidad and Tobago. Strategies for developing countries to benefit "fully" from FLOSS are presented including the implementation of formal organized programmes to educate and build awareness of FLOSS. The authors hope that by understanding some of the developing country issues that relate to OSS, solutions can be found. These countries could then fully benefit from OSS use, resulting in an increase in size of the global FLOSS development community that could potentially improve the quality of FLOSS and indeed all software.

INTRODUCTION

Open source software (OSS) is understood by many to mean software or computer programs where the source code is distributed and can be modified without payment of any fee by other programmers. The term OSS first came into use in 1998 and is attributed to Eric Raymond (Feller & Fitzgerald, 2002). The Open Source Initiative (OSI) has been formed to promote the use of OSS in the commercial world (www.opensource.org/).

The terminology related to software that is released with its source code and is modifiable and distributable without payment and then developed by a group of users or community can be confusing. For example the literal meaning of *open*

source implies access to the source code, without necessarily implying permission to modify and distribute. Also, the Free Software Foundation (FSF) (founded in 1985) which predates the OSI refers to these programs not as OSS but as *free software* (www.fsf.org/). The term *free software* was created by Richard Stallman where *free* refers to the freedoms to use, modify, and distribute the programs and does not have anything to do with the cost of acquiring the software. Therefore, *free* software does not necessarily mean *zero cost*, and *open source* does not just mean access to the source code.

The differences between free software and OSS have been well documented (Fuggetta, 2003). Stallman (2002) gives details about these differences at www.gnu.org/philosophy/free-software-for-freedom.html. What really then defines software as OSS? Generally, OSS is a software product distributed by license, which conforms to the Open Source Definition[1]. The best known of these licenses are the GNU General Public License (GPL) and the Berkeley Software Distribution (BSD) license. Unlike traditional commercial or proprietary software (e.g., Microsoft Word, Windows XP, or Internet Explorer), these licenses permit OSS to be freely used, modified, and redistributed. The source code for these programs must also be freely accessible.

The term *free/libre open source software* (FLOSS) is used to refer to both free and open source software and was first coined in 2002 by Rihab Ghosh in a study undertaken for the University of Maastricht (Ghosh, Glott, Kreiger, & Robles, 2002). Libre here is the French word for liberty making clear the *free* as in freedom and not free as in "no cost." It is also common to use the term FOSS (free/open source software) for such programs.

Many countries do not have the luxury of debating the philosophical differences between the OSS or free software movement and so are content to use the all encompassing term of FLOSS (see e.g., www.floscaribbean.org/). For the purposes of this discussion the terms Free Software, Open Source Software and FLOSS are used interchangeably[2].

The development of FLOSS has often been contrasted to that of proprietary software. FLOSS has primarily been developed by individuals who volunteer their time to work on FLOSS projects. Normally, modified versions of the source code are posted on the Internet and are available for free to anyone who wants to use or modify it further. Thus a community of developers is created all working on modifications, bug fixing, and customizations of the initial code. An extensive explanation and analysis of OSS development can be found in Feller and Fitzgerald (2002).

The number of open source software projects can be gleaned by visiting http://sourceforge.net/index.php where there are over a hundred thousand such projects registered to date. Thus, FLOSS is not a fad or fringe phenomenon. It is important to note that FLOSS has penetrated major markets in countries worldwide. Indeed, some open source products like Linux and Apache are market leaders globally, and major ICT companies like IBM, Sun, and Oracle have adopted the open source model (Bruggink, 2003). In some countries, governments have even made the decision to support the use of FLOSS (Brod, 2003; Evans & Reddy, 2003).

Because of its free cost and its freedoms, FLOSS should be an obvious choice for widespread use in developing countries. In fact, these countries should be virgin territory for FLOSS deployment. What do the developing countries themselves say and what has been their experience? This chapter presents the point of view and experience of developing countries with FLOSS.

BACKGROUND

Developing Countries

As is well-known, the term *developing country* applies to most African, Latin American, Caribbean, and Asian countries, as well as some countries in the Middle East and Eastern Europe. The definition of a developing country is generally based on that country's annual per capita income. Indeed, developing countries are most often defined following the World Bank classification (World Bank, 2006). According to the World Bank, the developing country's annual per capita Gross National Income (GNI) can range from:

- US$875 or less (low income)
- US$ 876-3,465 (middle income)
- US$ 3,466-10,725 (upper middle income)

Thus developing countries are not as homogeneous a group as some may think. Yet there are some common problems in all developing countries. For example, many such countries have unreliable electricity supplies (Ringel, 2004). Additionally, ownership of computers and access to the Internet is low when compared to developed countries (Table 1).

In simple terms, how useful is FLOSS without hardware or electricity or trained and skilled personnel? The vision of developing countries being able to leapfrog from the use of proprietary software into using FLOSS and benefiting from all its "freedoms" must be tempered with these realities (Steinmueller, 2001). This chapter presents many of these realities as they relate to FLOSS in developing countries. If the problems with FLOSS in developing countries could be solved, then such countries could fully participate in FLOSS development. This would increase the size of the global FLOSS development community thereby creating the potential for an increase in the quality of FLOSS and software in general.

Table 1. Telecommunications infrastructure for Internet access; comparison of selected countries for 2002 (Source: United Nations Statistics Division-Millennium Indicators [ITU estimates] from http://unstats.un.org/unsd/mi/mi_series_list.asp rounded to the nearest whole number)

Countries	Telephone Lines and Cellular Subscribers/100 Population	Personal Computers/100	Internet Users/100
Trinidad and Tobago	53	8	11
United Kingdom	143	41	42
United States	114	66	55
Singapore	126	62	50
Sweden	163	62	57
Venezuela	37	6	5
Brazil	42	8	8
Chile	66	12	27
China	33	3	6
Guyana	19	3	14
India	5	1	2
Nigeria	2	1	0

Benefits of FLOSS

The benefits of FLOSS are well-documented in the literature particularly by Raymond (2002). These benefits include:

- Free or small cost of acquisition; future upgrades are free
- Flexibility of its license vs. restrictive licenses of proprietary software; the General Public License (GPL) used to license most open source software is much more flexible than the End-User License Agreement (EULA) of proprietary counterparts, giving more freedom to users to customize and to install on as many computers as needed without incurring added costs
- Superior quality and stability of FLOSS; because the source code is open to full and extensive peer review, open source software is known for its superior quality and stability
- Effectiveness as a teaching tool vs. closed proprietary software; users of FLOSS learn team work; importance of intellectual property protection and ethical use of software in addition to programming skills (Rajani, Rekola, & Mielonen, 2003)
- Potential as a solution to the software crisis; the "software crisis" refers to "software taking too long to develop, costing too much, and not working very well when delivered" (Feller & Fitzgerald, 2000, p. 58)
- Reduces the dependence of public administration and international governments in particular on specific software providers (Fuggetta, 2003); according to Nolle (2004), internationally, where Microsoft is viewed with more alarm than it is in the United States, FLOSS is seen as a defense against U.S. and Microsoft domination
- Stimulates innovation; FLOSS encourages the mastering of the technology of software by enabling the development and expression of creativity in the modification of the software by its users
- Improves commercial software
- Develops and enables applications that leverage local knowledge; because it can be freely modified, FLOSS is easier to translate, or localize (Bruggink, 2003)
- Fosters the creation of local software industry and entrepreneurs; the potential exists for the creation of local companies and small businesses supplying services associated with FLOSS in training, support, customization, and maintenance (Ghosh, 2003; Rajani, et al., 2003)

FLOSS Challenges

There are those who question most of the stated benefits of FLOSS particularly its claim to be innovative (Boulanger, 2005; Evans & Reddy, 2003; Fuggetta, 2003). Those on the side of proprietary software suggest that FLOSS is less secure, not as high in quality, stable, or dependable as its advocates insist. The very model of development of FLOSS that results in its best qualities can also lead to concerns about lack of support (Lai, 2006), security, and possible intellectual property violations by incorporating FLOSS into proprietary software (Kramer, 2006).

Compatibility concerns are also common. For example, although most FLOSS runs on both Microsoft Windows and Mac OSX, some run only on the Linux operating system. FLOSS may not come with as complete documentation and ready support as proprietary alternatives. Fees may have to be paid for substantial technical support. It should also be noted that there are fewer trained people available to provide technical support since most ICT training programmes prepare students to work with the most commonly used proprietary software packages, such as those from Microsoft (Bruggink, 2003). Additionally, FLOSS may require more learning and training time as well as skill to deploy and maintain. Large scale

migration from proprietary software installations to FLOSS can be problematic, particularly if there is a lack of practical experience and support and ready information on migration issues (Bruggink, 2003; Van Reijswoud & Mulo, 2005).

Cost as the main driver for the adoption of FLOSS in developing countries cannot be ignored. Ghosh (2003) demonstrates this vividly by comparing license fees for proprietary software with the income per capita of selected countries. He concludes that in developing countries, "even after software price discounts, the price tag for proprietary software is enormous in purchasing power terms." This is further supported by the Free and Open Source Software Foundation for Africa (FOSSFA) (as cited in May, 2006) who report that countries in sub-Saharan Africa each year pay around US$24 billion to (mainly U.S.-based) software companies for the rights to use proprietary software.

Thus FLOSS provides an opportunity for developing country institutions to find cost effective solutions in many areas that could include electronic governance to online health and learning. But there is an even greater benefit of FLOSS to these countries. Following the old adage that it is better to teach a man to fish than to give him fish, there is some appreciation that OSS can be even more beneficial to developing countries because it can be a vehicle for technology transfer of software development skills, thus building local IT capacity and stimulating innovation (Camara & Fonseca, 2006; Ghosh, 2003). Yet, for many end-users and even institutions in these countries, the choice is not between FLOSS and proprietary software but between FLOSS and cheap pirated software. When faced with this choice there is very little incentive to consider FLOSS (Heavens, 2006).

Furthermore, limited Internet access and bandwidth may not allow regular interacting with FLOSS online communities for updates, documentation and help with problems (Heavens, 2006). In addition, jobs in the IT industry in these countries are often confined to large companies that place a high premium on skills in traditional proprietary software (e.g., Microsoft Certification and experience). Also for those uninformed about FLOSS in developing countries, there is much skepticism about its use since "free" is often equated with poor quality and expensive software with high quality and reliability. This is confirmed by Gregg Zachary (as cited in Fitzgerald & Agerfalk, 2005) in his personal communication about unsuccessful attempts to introduce FLOSS projects in Ghana.

Are these difficulties peculiar to some developing countries? As a contribution to the FLOSS debate it may be useful to present the experience of a major university in a developing country.

EXPERIENCE IN THE WEST INDIES

The University of the West Indies (UWI)

The University of the West Indies (UWI) was first established in 1948, as a college with a special relationship with the University of London to serve the British territories in the Caribbean area. There are three UWI Campuses, in three different West Indian islands: Mona in Jamaica, Cave Hill in Barbados and St. Augustine in Trinidad and Tobago.

FLOSS at the University of the West Indies – St. Augustine Campus

The St. Augustine campus of the UWI is located in the middle income developing country of Trinidad and Tobago. Rampersad (2003) gives a succinct description of FLOSS in Trinidad and Tobago and reports that "Proprietary software is used most in Trinidad and Tobago, and as such, Microsoft and its many applications have a strong grip on the IT market." The University of the West Indies just like other employers of IT personnel in Trinidad and Tobago places high value on

proprietary software certification (e.g., MCSE). Additionally, agreements have been made with computer manufacturers like Dell for the supply of computers campus wide and these are naturally shipped with proprietary software.

It is therefore not surprising that, like many similar developing country institutions, the UWI, St. Augustine campus has no formal institutional policy for the use or deployment of FLOSS. Individual IT personnel and other staff members at UWI who become aware of FLOSS solutions have tried using these in their various departments or units. The main motivation for this has been the cost of FLOSS versus proprietary software particularly when licensing per-seat costs are considered in deploying software in large computer labs. The FLOSS software used so far at the university is shown in Table 2.

Were the other vaulted outcomes of FLOSS use in developing countries experienced at the UWI? Modification of source code, customization, and so forth, implies that there exists a certain level of programming skills locally. In Trinidad and Tobago, practical computer programming skills are in very short supply and so FLOSS is sometimes seen as just a cheap alternative to the high cost of proprietary software, nothing more.

Also, as is the case with most developing countries, UWI has a small IT staff fully engaged at any time on a multiplicity of projects. There is often no time to invest in modifying source code.

Table 2. FLOSS used at UWI St. Augustine

FLOSS	Type
Linux	Operating System
Open Office/Star Office	Productivity Software
PHP, PERL	Middleware
MySQL	Database
Moodle	Courseware
DSpace	Institutional Repository
Apache	Web Server

A good example of how limited resources can affect the progress of FLOSS projects in particular is UWI, St. Augustine's Institutional Repository Project which is based on the open source DSpace software (www.dspace.org). The initial impetus for the implementation of an institutional repository at the UWI, St. Augustine campus was a need to expose the unique Caribbean resources housed in the West Indian collection of the library to the world via digitization.

DSpace was acquired in 2004 and was installed first on a test server at the UWI Main Library in early 2005. Yet the installation is still "ongoing" since it involves a steep learning curve for the staff charged with the technical implementation. Knowledge and skills in Linux, Apache, Tomcat, and Java programming required for a successful DSpace repository deployment are not readily available. Thus, progress on implementation of the repository has been slow (Papin-Ramcharan & Dawe, 2006). Like most developing countries which do not have in place a well developed IT infrastructure and highly skilled IT personnel, it has been found that the true total cost of ownership (TCO) of DSpace as a FLOSS institutional repository solution has been high.

FUTURE TRENDS

It seems clear that the initial cost of acquiring FLOSS has been the key motivation for many developing countries adopting FLOSS solutions. It is also clear that there are greater benefits that can be derived from FLOSS in terms of encouraging the development of local IT skills, the creation of jobs locally to support FLOSS, and the eradication of piracy of proprietary software. Independence from being hostage to a single proprietary vendor is also beneficial to such countries.

The benefits to a country and its citizens from FLOSS adoption can possibly be viewed along a spectrum. Some countries which are relatively new to FLOSS will take time to fully exploit its

potential, whereas those that are farther along will work on higher value FLOSS activities like customization. Eventually, developing country users could move from being just consumers of FLOSS to being equal participants in the global community by becoming initiators and creators of FLOSS projects (i.e., FLOSS developers). Further along the spectrum, local jobs and small businesses could be created to sell FLOSS support and maintenance services.

It also seems likely that for developing countries and others, there probably will never be a FLOSS-only or proprietary-only market. The future will be about choice, where both FLOSS and proprietary software will co-exist and decisions to acquire software will not be based on philosophy alone but follow the standard criteria used to select any software package.

CONCLUSION

The literature while emphasizing that FLOSS is obviously a cost effective solution for developing countries also extols its higher benefits. These include: its technology transfer potential, the creation of jobs, fostering of innovation and creativity, the reduction in piracy of proprietary software, the independence achieved from being hostage to a single proprietary vendor, and the ability to localize software products to local languages and conditions. These outcomes will not be achieved for most developing countries unless there are enhanced supporting mechanisms to foster FLOSS use. These can emanate from international agencies like those of the UN and World Bank whose interest lie (for example) in the sustainable development of developing countries. The mechanisms could include:

- Formal organized programmes to educate and build awareness of FLOSS in developing countries; this should not just be targeted to IT personnel but to common users, governments, and other decision makers
- International agencies working presently to upgrade ICT skills and infrastructure in developing countries should work closely with the FLOSS "movers and shakers" to ensure that training is provided in these countries on commonly used FLOSS with emphasis on programming skills.
- Sponsoring agencies that support nongovernmental organizations (NGO) or other community organizations should require that FLOSS be considered for use in their operations and projects.
- Procurement agencies of governments and other bodies should be educated about FLOSS so that it can be seen as a viable alternative when procurement decisions are made.
- Examination and other education bodies must be encouraged in an organized and targeted manner to change the computer studies and science programmes in these countries from being mostly Microsoft-centric to include the study and use of FLOSS.

REFERENCES

Boulanger, A. (2005). Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal, 44*(2), 239-248.

Brod, C. (2003). *Free software in Latin America: Version 1.2*. Retrieved August 18, 2006, from http://www.brod.com.br/file_brod//helsinki.pdf

Bruggink, M. (2003). *Open source software: Take it or leave it?* International Institute for Communication and Development (IICD) Report. Retrieved July 6, 2006, from http://www.ftpiicd.org/files/research/reports/report16.pdf

Camara, G., & Fonseca, F. (2006). Information policies and open source software in developing

countries. *Journal of the American Society for Information Science and Technology (JASIST)* (pre-print version). Retrieved August 26, 2006, from http://www.dpi.inpe.br/gilberto/papers/camara_fonseca_jasist.pdf

Evans, D. S., & Reddy, B. J. (2003). Government preferences for promoting open-source software: A solution in search of a problem. *Michigan Telecommunications and Technology Law Review, 9*(2). Retrieved August 22, 2006, from http://www.mttlr.org/volnine/evans.pdf

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source development paradigm. In *Proceedings of the 21st ACM International Conference on Information Systems,* Brisbane, Queensland, Australia (pp. 58-69). Atlanta, GA: Association for Information Systems .

Feller, J., & Fitzgerald, B. (2002).*Understanding open source software development*. Reading, PA: Addison Wesley.

Fitzgerald, B., & Agerfalk, P. J. (2005, January 3-6). The mysteries of open source software: Black and white and red all over? In R. H. Sprague (Ed.), *Proceedings of the 38th Hawaii International Conference on System Sciences*, Big Island, Hawaii [CD-ROM]. Los Alamitos, CA: IEEE Computer Society Press.

Fuggetta, A. (2003). Open source software––An evaluation. *The Journal of Systems and Software, 66*, 77-90.

Ghosh, R. A. (2003, December). Licence fees and GDP per capita: The case for open source in developing countries. *First Monday, 8*(12). Retrieved August 20, 2006, from http://firstmonday.org/issues/issue8_12/ghosh/index.html

Ghosh, R. A., Glott, R., Kreiger, B., & Robles, G. (2002). *Free/libre and open source software study: FLOSS final report*. International Institute of Infonomics, University of Maastricht. Retrieved August 16, 2006, from http://www.flossproject.org/report/

Heavens, A. (2006, July 10). Ubuntu in Ethiopia: Is free such a good deal? [Blog post]. *Meskel square*. Retrieved August 13, 2006, from http://www.meskelsquare.com/archives/2006/07/ubuntu_in_ethiopia_is_free_such_a_good_deal.html

Kramer, L. (2006, April). The dark side of open source. *Wall Street & Technology,* 43-44.

Lai, E. (2006). Lack of support slowing spread of open-source applications. *Computerworld, 40*(8), 20.

May, C. (2006). The FLOSS alternative: TRIPs, non-proprietary software and development. *Knowledge, Technology & Policy*, *18*(4), 142-163.

Nolle, T. (2004). Time to take open source seriously. *Network Magazine, 19*(4), 82-83.

Papin-Ramcharan, J., & Dawe, R. A. (2006). The other side of the coin for open access publishing—A developing country view. *Libri*, *56*(1), 16-27.

Rajani, N., Rekola, J., & Mielonen, T. (2003). *Free as in education: Significance of the free/libre and open source software for developing countries: Version 1.0.* Retrieved August 6, 2006, from http://www.itu.int/wsis/docs/background/themes/access/free_as_in_education_niranjan.pdf

Rampersad, T. (2003). Free- and open-source software in Trinidad and Tobago. *Linux Journal*. Retrieved August 22, 2006, from http://www.linuxjournal.com/article/6619

Raymond, E. S. (2002). The cathedral and the bazaar. In *The cathedral and the bazaar,* 23. Retrieved July 18, 2006, from http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Ringel, M. (2004). The interlinkage of energy and poverty: evidence from India. *International Journal of Global Energy Issues*, *21*(12), 2746.

Stallman, R. (2002). *Free software, free society: Selected essays of Richard M. Stallman* (J. Gay, Ed.). Boston: Free Software Foundation.

Steinmueller, W. E. (2001). ICTs and the possibilities for leapfrogging by developing countries. *International Labour Review, 140*(2),193-210.

Van Reijswoud, V., & Mulo, E. (2005, March 14-15). *Free and open source software for development myth or reality? Case study of a university in Uganda.* Paper presented at a seminar on Policy Options and Models For Bridging Digital Divides Freedom, Sharing and Sustainability in the Global Network Society, University of Tampere, Finland. Retrieved August 22, 2006, from http://www.globaledevelopment.org/papers/Artikel%20OSS-UMUv2%5B1%5D.1.pdf

World Bank. (2006). *Data and statistics: Country classification*. Retrieved August 24, 2006, from http://web.worldbank.org/WBSITE/EXTERNAL/DATASTATISTICS/0,,contentMDK:20420458~ menuPK:64133156~pagePK:64133150~piPK:64133175~theSitePK:239419,00.html

KEY TERMS

Developing Countries: Developing countries are those that have an annual per capita income (Gross National Income [GNI]) between US$875 and US$10,725.

Free/Libre Open Source Software (FLOSS): Used to refer to both free and open source software making no distinction between them.

Free Software (FS): Computer programs that are not necessarily free of charge but give access to the source code and permit users the freedom to freely use, copy, modify, and redistribute.

Open Source Software (OSS): Software that meets the terms of the Open Source Definition (www.opensource.org/docs/definition.php). To be open source, the software must be distributed under a license that guarantees users the right to read, redistribute, modify, and use freely.

Proprietary Software (PS): Software that is normally owned by a company that typically restricts access to the source code to protect the company's intellectual property. The software is distributed as the "compiled" source code or executable code (the binary form of the program). Its use, redistribution, or modification is prohibited or severely restricted (e.g., Microsoft Word, Norton Antivirus).

Source Code: The list of instructions that make up a computer program written in a high level programming language (like C, Java or PHP) that humans can read, understand and modify.

Total Cost of Ownership (TCO): The full cost of deploying, maintaining and using a system (or software) over the course of its lifespan.

ENDNOTES

1 Open Source Definition, Version 1.9. Retrieved July 15, 2006, from http://www.opensource.org/docs/definition.php

2 It is important that FLOSS is not confused with terms like freeware and shareware. These terms are usually used to describe software which is available at no cost, but its source code usually is closed. Internet Explorer is one example of freeware that is proprietary.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited byK. St. Amant; and B. Still, pp. 93-101, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 5.10
Open Source and Outsourcing:
A Perspective on Software Use and Professional Practices Related to International Outsourcing Activities

Kirk St.Amant
Texas Tech University, USA

ABSTRACT

This chapter examines the role of open source software (OSS) in international outsourcing practices that involve the transfer of knowledge work from one nation to another. Included in this examination are discussions of the benefits and the limitations of OSS use in outsourcing. The chapter also presents organization-specific and industry-wide strategies for effective OSS use in outsourcing situations. The chapter then concludes with a discussion of areas of international outsourcing where OSS might have important future applications or effects. The purpose of such an examination is to provide readers with the knowledge and the insights needed to make effective decisions related to the use of OSS in international outsourcing situations.

INTRODUCTION

International outsourcing now includes the distribution of knowledge-based work to employees in other countries. Much of this work, however, requires the use of software either to perform a task or to provide the technologies that allow clients and outsourcing providers to interact. Conventional "proprietary" software can, however, be prohibitively expensive to outsourcing employees in developing nations.

Open source software (OSS) might offer a solution to this problem, for OSS is often free to use and is relatively easy to modify or to update. Yet open source software also brings with it a new series of problems related to product consistency, user support, and digital piracy. While the relationship between outsourcing and software (particularly OSS) has been known for some time, it has received relatively little attention in terms of social and economic implications both for those who outsource work and for those who perform outsourced work. Knowledge of these issues, however, is essential to understanding both current and future outsourcing practices and the socio-economic development of nations that engage in outsourcing.

The purpose of this chapter is to provide readers with a foundational knowledge of how OSS use

could affect international outsourcing practices. After reading the chapter, individuals will understand the relationship between software and outsourcing practices in terms of the opportunities and the limitations it creates for client companies and for outsourcing employees in developing nations. This chapter also presents strategies organizations can employ to use OSS more effectively in international outsourcing situations. The chapter then concludes with an overview of how global computing and OSS use is poised for significant growth and the implications this growth could have for different organizations.

BACKGROUND

The Growth of International Outsourcing

In international outsourcing — or offshoring — situations, companies in one nation transfer the responsibility for completing a task to workers in another country (Bendor-Samuel, 2004). Originally, this transfer of responsibility focused on manufacturing and the production of physical products such as clothing or footwear. The global spread of online media, however, has given rise to a new kind of international outsourcing that involves the export of knowledge-based work. Known as business process outsourcing — or BPO — this practice encompasses everything from computer programming to call center staffing and medical transcription. While such BPO practices have existed on a relatively limited scale to date, they are poised to expand rapidly in the future.

The push to adopt BPO has to do with the perceived benefits related to such practices. Perhaps the most publicized of these benefits is savings related to the cost of skilled labor. Much of today's knowledge work is being outsourced to skilled employees in developing nations — employees who can perform most technical tasks for far less than what counterparts in industrialized nations would charge. For example, gaming developers in Russia earn roughly $100 U.S. a week, while middle managers in mainland China earn roughly $9,000 a year (Weir, 2004; Nussbaum, 2004).

Such wage-based savings, however, are not the only advantage related to offshoring. Rather, proponents of outsourcing note it also offers the benefits of

- Improved quality of service: Research indicates overseas outsourcing employees often provide better quality service than "domestic" workers who perform the same jobs (Reuters, July 182004; Farrell, 2004; Hagel, 2004). Certain call centers in the Philippines, for example, take 25% less time to handle incoming calls and receive higher rates of caller satisfaction than do U.S. counterparts (Hagel, 2004).
- Effective management practices: Because managers are paid less in developing nations, organizations can easily justify the use of more in-country managers to oversee outsourcing activities (Nussbaum, 2004; Hagel, 2004). This increase means managers have more time to answer employee questions and to provide employee training — factors that contribute to the improved quality of work or service perceived by many consumers (Hagel, 2004; Lewis, 2003; Hagel, 2004).
- Reduced employee turnover: Secure employment is often rare in many developing nations, and outsourcing jobs tend to be among the better paying ones. Therefore, outsourcing workers in developing nations tend to stay with employers for longer periods of time (Reuters, July 18, 2004; Farrell & Zainulbhai, 2004). As a result, these long-term employees tend to have more experience performing their jobs while reducing the need for and the cost of new employee training.
- Reduced production time: By using online media to distribute work to employees in

different nations and time zones, organizations can keep operations going 24 hours a day, seven days a week. Such continual production means that processes and products can be completed more quickly than if done exclusively in one country (Friedman, 1999; Baily & Farrell, 2004; Bierce, 1999; "America's pain," 2003).

- Increased access to international markets: In many developing nations, marketplace success is often a matter of knowing someone in that region (Rosenthal, 2001). Companies that provide outsourcing services can provide such an "in," as well as provide advice on how one should proceed with business interactions in a particular nation or region (Rosenthal, 2001).

The manifold benefits of offshoring have prompted many companies to adopt it as a part of their core business strategy. At present, it accounts for $10 billion U.S. in worth and engages the services of some 500,000 workers in India alone (Baily & Farrell, 2004; Rosenthal, 2004b). Moreover, some researchers believe offshoring will grow by 20% a year through 2008, and by 2015, some 3 million business processing jobs will be performed by outsourcing employees (Rosenthal, 2004b; Baily & Farrell, 2004). Some critics expect this number to be much higher and claim at least 5 million international outsourcing jobs will emerge in the next five to 10 years (Garten, 2004).

The perceived benefits of international outsourcing have resulted in such practices spreading to a number of countries and to a variety of industries. French companies, for example, have begun working with French-speaking developing nations such as Senegal and Morocco on a variety of outsourcing projects, and these relationships have met with good results (Reuters, July 18, 2004). Similarly, German firms have begun exploring outsourcing relationships with Easter European nations, while the Netherlands has begun using call centers located in South Africa where Dutch-based Afrikaans is spoken (Farrell, 2004; "Sink or Schwinn," 2004; Baily & Farrell, 2004; Rosenthal, 2004c). Additionally, markets in Spain, combined with the growth in the U.S. Spanish-speaking population, have meant more work is now outsourced to Mexico and to Latin America (Rosenthal, 2004a). Even India, the one time center for international outsourcing, is now outsourcing work to China and to Sri Lanka, where it can be done for less money (Reuters, September 2, 2004).

The majority of BPO practices, however, are reliant on software. Aspects related to software use can thus affect if and how organizations realize the advantages related to the offshoring of knowledge work. For this reason, decision makers in the public and the private sectors need to understand the differences between proprietary and open source software — as well as the limitations and advantages of each — in order to make informed choices related to international outsourcing. Only through such informed decision-making can organizations benefit from the advantages related to offshoring.

MAIN THRUST OF THE CHAPTER

Software, Cost, and International Outsourcing Activities

Software is essential to international BPO practices for two key reasons. First, it provides the mechanism for sharing materials and for interacting with others online (e.g., browsers and email systems). Without the software needed to engage with others quickly, easily, and directly via the Internet, many of the cost, time, and quality benefits associated with the international-outsourcing of knowledge work would not exist. Second, production-related software (e.g., Microsoft Word or Adobe Illustrator) is essential to performing most knowledge-based programming, customer

support, and IT tasks efficiently and effectively. Again, without the software needed to perform tasks, the time, cost, and quality benefits of international outsourcing cannot be realized.

Unfortunately, the makers of software products have traditionally produced proprietary programs that require individuals to purchase them in order to perform a given activity. The for-profit nature of proprietary software has, however, made it unavailable to large segments of the world's population — particularly in developing nations where high purchase prices are often associated with such materials (Warschauer, 2003). This restriction, in turn, affects online access and thus outsourcing activities in those regions.

One solution to this situation would be for companies to supply prospective international outsourcing providers with free or inexpensive software products that would allow them to participate in outsourcing activities. Such an approach, however, would contribute to a second major software problem — copyright violation. In many developing nations, copyright laws are often weak (if not non-existent) or governments show little interest in enforcing them. As a result, many developing nations have black market businesses that sell pirated versions of software and other electronic goods for very low prices (Balfour, 2005). Such software piracy reduces consumer desire to purchase legitimate and more costly versions of the same product, and thus affects a company's profit margins within that nation. In fact, global software piracy in 2004 accounted for some $33 billion U.S. in lost profits ("One third," 2005). Further compounding this problem is the fact that it is often difficult for companies to track down who is or was producing pirated versions of software products in order to stop that offender. Thus, while the distribution of cheap or free digital materials can help contribute to outsourcing activities, that same strategy can undermine an organization's ability to sell its products abroad. Open source software (OSS) can offer a solution to this situation.

Open Source Software and International Outsourcing

Software, in essence, is programming code — or source code — that tells a computer's operating system how to perform a certain action (Still, 2004). The source code of Microsoft's software program Word, for example, tells the related operating system how to transfer keystrokes into letters on a page and how to edit or to print that page. In theory, if an individual knows what programming/source code is needed to make a computer perform such activities, then that individual can simply retype that source code into his or her operating system, and the computer will respond as desired — the exact same way the original software would.

Accordingly, if and individual can see how someone else programmed software to work (see the underlying source code), all that person needs to do is copy that programming/source code, and he or she no longer need to buy the related software. Rather, that person can now achieve the same action on his or her own. For this reason, many software companies "close" their source code in order to prevent users from seeing the underling programming that allows the software to work (Still, 2004). In these situations, users need to work through an interface that allows them to activate certain commands indirectly within a software program's underlying source code. Closed software that prevents users from seeing the underlying code is know as proprietary software, for only the creator of that software is allowed to open or to see and to copy or to manipulate the underlying source code.

Open source software is a polar opposite in terms of access to source code/programming. That is, the developer of a piece of software creates it in manner that is "open" and permits any and all users to access, copy, and modify the underlying source that allows software to work (Still, 2004). A classic example of such open source programming is the HTML coding that allows Web pages

to be displayed on browsers. The coding of these pages is open for anyone to review and copy; all the user needs to do is view a page's underlying source code by using the "View source" — or related option — in his or her browser.

This openness means individuals do not necessarily need to buy open source software in order to use it. Rather, they can directly access and copy the underlying source code in electronic format, or they can re-code/re-type the programming code and create a free copy of the related software. Such openness also means individuals can alter the underlying source code to make a software program perform different functions. So, in theory, a copy of the underlying source code from one kind of software could be modified into a variety of programs — each of which performs a different task. Updating software, moreover, becomes a matter of copying new/updated code versus purchasing the newest version of a product.

In the case of international outsourcing, OSS can provide individuals with access to affordable software that allows them to work within outsourcing relationships. Moreover, the flexibility permitted by OSS means outsourcing workers could modify the software they use to perform a wide variety of tasks and reduce the need for buying different programs in order to work on different projects. As the software it is produced by the outsourcing employee and not the client, concerns related to copyright and proprietary materials are no longer stumbling blocks to outsourcing relationships. Thus, it is perhaps no surprise that the use of OSS is growing rapidly in many of the world's developing nations ("Open source's," 2003).

The Problems of Open Source Software in International Outsourcing

While the free and flexible nature of open source software allows it to contribute greatly to international outsourcing situations, OSS also brings with it limitations that could affect the success of BPO relationships. First, because OSS is open for the user to modify as he or she sees fit, it is easy for each individual to use the same programming foundation/source code to develop a different kinds of non-compatible software or other digital products. Such divergence is know as forking code, for each programmer can take a different "fork" in the programming "road," and such forking code has long been considered a major problem in OSS use (Still, 2004). These prospects for divergence mean international outsourcing situations are open to a variety of problems related to compatibility. Such problems include

- Employees might generate software or other materials that the related client company cannot use due to compatibility issues.
- Software products might not work as desired or work in unexpected ways/ways not compatible with the client company's intentions.
- Employees working on different parts of the same project might produce component parts that cannot be integrated into the same whole or do not work together correctly or as intended within that same whole due to compatibility issues.
- Addressing compatibility issues either among international work groups or between offshoring workers and clients could take more time and cost more than if the product had been produced domestically.

Thus, the freedom that allows one individual to operate software might also prevent others from making use of digital materials. Additionally, any or all of these factors could contradict the advantages of reduced cost, quicker production time, and increased quality that encourage organizations to use international BPO.

Some OSS companies, such as LINUX, have successfully addressed the problem of forking

code through focused oversight processes that govern programming practices (Hamm, 2005). The result has been successful and relatively stable software products that work effectively with other systems. The same kind of management, oversight, and standardization, however, becomes more complicated in an international outsourcing situation where a variety of employees can be working in different nations and different time zones

A second problem involves the kind of technical support available to OSS users — both outsourcing employees and consumers using OSS products. Because no individual or organization really owns an open source software product, there is often no formal or standardized mechanism for providing technical support to OSS users. Rather, technical support often comes in the form of a loose network of OSS programmers/developers who interact informally — and sometimes haphazardly — in online contexts such as chat rooms or listservs (Still, 2004). The idea is that a user who is experiencing software difficulty posts a query to one of these online forums and waits for a member of that forum to read the posting and respond to it.

A major problem in such an informal system is that technical support/answers are not readily available. Rather, individuals who experience "glitches" related to time-sensitive OSS projects could find themselves missing or offsetting deadlines as they wait for some random programmer to respond to a request for help. Unfortunately, such a response could take anywhere from minutes to days depending on who is reading what lists or who is posting when. In the fast-paced environment of global business, such delays could have a major effect on production schedules, profits, and access to international markets. These delays also counteract one of the key benefits of and reasons for using international outsourcing — quicker production times.

Equally problematic is the fact that such support systems are open for anyone to respond to, regardless of a person's technical skills or understanding of the situation (Still, 2004). Thus, the quality of the advice related to such a support system can be haphazard, inconsistent, or even wrong. Moreover, different individuals might offer varying suggestions/solutions to the same situation. Thus, requests for assistance could introduce the similar problem of forking/diverging source code into situations where different individuals encounter problems when working on the same project. As a result of these limitations, materials created using OSS could be incompatible, inconsistent, or even non-functional depending on the kind of "help" one received.

Finally, open source software creates interesting and often unintended problems related to copyright. Most proprietary operating systems (e.g., Microsoft Windows) and proprietary software programs (e.g., Microsoft Windows Media Player) work by opening digital materials loaded into a computer system. That is, Windows Media Player opens the video files on a DVD or the audio files on a CD so that individuals may view or listen to films or music stored on those DVDs or CDs. This approach means the individual is able to access materials while maintaining the copyright related to such items.

Open source software allows users to access the same kinds of files in a very different way. Rather than simply opening files located on a DVD or CD, OSS programs often automatically make a copy of such digital products, store that copy on the hard drive of the user's computer, and then open that copied file so the user can view or can listen to it (Zoellick, 2001). This automatic copying process, known as ripping, means that users of OSS materials automatically violate copyright just by trying to access or to use certain digital products. These unauthorized copies are available for anyone with access to the related computer — an important problem in outsourcing situations where more than one employee often uses the same terminal for work. Such ripped copies, moreover, can easily be replicated and

sold in black market exchanges — a situation of particular importance as such illegal activities are well established and difficult to monitor in many of the world's premier outsourcing locations (e.g., the People's Republic of China). As a result, the copyright — or piracy — problems OSS could solve in terms of misuse of proprietary software could affect different products should

- The original client company needs to provide outsourcing employees with proprietary digital materials in order to perform certain outsourced tasks.
- The original client company sells OSS products resulting from outsourcing work to different customers who can now use such products to make illegal copies of digital materials.

Thus, the use of OSS in BPO can be a double edged sword. Yet, the use of OSS in international outsourcing is a situation that must be addressed if organizations wish to succeed in the global business environment of the 21st century. For this reason, organizations must find ways to strike a balance between OSS's benefits and detriments in offshoring situations. Such a balance can be established through the development and the use of various organizational and industry-wide practices and policies.

Strategies for Using OSS Effectively in International Outsourcing

While OSS use in outsourcing creates a complicated situation, these complications can be addressed successfully through approaches that create standards for design and for use. What follows are ten strategies that can create an effective foundation for using OSS in international BPO.

- Strategy 1: Create a programming standard all outsourcing employees will use when working with OSS. By creating and sharing standards for how one should program with OSS, organizations that outsource activities can provide overseas workers with guidelines that impose consistency on the programming process. These guidelines would instruct outsourcing OSS users on what programming choices to make during different points in an overall process. Such instructions would help reduce the prospects of forking code and increase compatibility across employees.

Standards should also be created to address the coding of comments — or the bits of verbal explanation that appear within the coding of a program. These comments allow users and clients to know why a particular programming decision was made and to know the effects that decision could have on the operation of the related digital product. By creating standards for when and for how to create such comments, companies that use outsourcing create a built-in mechanism for reviewing materials produced by overseas workers. To make sure that such standards are followed by outsourcing employees, the related client company should suspend all or part of final payment until it has had time to review the final product, and its related comments, in order to confirm standards were followed. While such monitoring could initially slow production, as outsourcing employees become aware of the fact their work is being monitored, they will spend more time adhering to standards. In so doing, these employees will create a better product. Thus, over the long term, the time spent monitoring OSS processes would decrease while the quality of products would increase. The increase in quality resulting from such practices would therefore eventually offset any initial slowdown in production caused by the introduction of monitoring procedures.

- Strategy 2: Require outsourcing workers to identify themselves as OSS users and require those users to complete organiza-

tional training programs in standard OSS practices. Such identification and training — which can be completely online — would allow organizations to provide outsourcing workers with instruction in how to follow a particular organization's standards for OSS use and coding. It would also allow companies to identify which standards seem most problematic for individuals (via automatically reviewed online tests). Organizations could then provide follow-up online training to help individuals address their particular problem areas related to OSS. Initial expenses dedicated to the development and the delivery of such training can contribute to increased later profits in the form of more efficient, more effective, and more standard uses of OSS by outsourcing employees.

- Strategy 3: Develop an organizational support mechanism for helping outsourcing OSS users. To make sure that outsourcing employees using OSS get consistent and accurate answers to technical questions, organizations should develop their own OSS online support lists or Web sites. Such Web sites would be free access, but they would require individuals to enter a username and password (supplied by the organization) to gain access to such resources. Such a measure would cut down on non-employees taking advantage of this resource and thus diverting attention away from the help given to an organization's actual workers.

This resource would be staffed by employees who have a high degree of proficiency in OSS use, and these "help" persons would be "on call" 24 hours a day to accommodate queries from outsourcing employees in different nations and time zones. These help employees would also be given access to an online database of the kinds of projects being worked on by different outsourcing employees. Help workers could then use this information to determine the prospective problems (and related best solutions) affecting different outsourcing workers. Such help workers would also record all comments and suggestions in a central online database that would cross-reference those suggestions with the names of all outsourcing workers involved in the same project. Such cross-listing would allow other help providers to offer consistent advice (and germane suggestions) to all outsourcing employees working on a project.

- Strategy 4: Establish a peer-mentoring program for outsourcing workers who use OSS. The effective use of OSS in a business environment is part understanding programming processes and part understanding the culture and the goals of the organizations using those programs. For this reason, the more outsourcing workers know about an employing organization's culture, goals, and objectives, the more effectively they can use OSS to engage in processes or develop products that address those factors (Clement, 1994; Tan, 2000; Sandholm, 2004). That is, if the outsourcing employee knows how a given OSS product will fit into a company's objectives, that employee will have a better idea of how to program to meet those objectives (Clement, 1994; Tan, 2000; Sandholm, 2004). Similarly, if the outsourcing employee is aware of the importance of copyright and proprietary products to a company, that employee is less likely to make unauthorized copies — either inadvertently or intentionally.

By pairing outsourcing employees with a peer mentor from the client company, organizations provide such outsourcing employees with someone who can provide an introduction to the organization's culture and its objectives. This mentor could make the outsourcing employee feel more a part of the overall organization. Such a peer could also familiarize outsourcing employees with company standards for OSS use and copyright

policy (as well as the consequences for violating such a policy). In so doing, the mentor creates a mechanism for accountability, for the outsourcing employee is no longer a "faceless" worker who exists as a mere payroll number.

Such a peer mentor should also have a background in OSS so he or she can provide the outsourcing worker with advice on how to best address a particular programming/software situation. The mentor could similarly introduce the outsourcing employee to corporate training materials and support services and review programming work to check its quality. Work with such a peer mentor could be made an essential part of the training process associated with outsourcing workers. The mentor's evaluation of that worker's progress could likewise serve as an important resource when making the decision to continue using a particular outsourcing employee.

- Strategy 5: Create protocols for sharing or for forwarding work among international outsourcing employees. In many cases, work that is outsourced to one overseas employee is shared or forwarded to another international outsourcing employee working on the same process (Freedman, 1999). Within such contexts, effective information sharing is essential, for the more time it takes an individual to determine what a predecessor has done and expects that individual to do, the less efficient the international outsourcing process is. For this reason, outsourcing employees need to use a standard mechanism for listing what they have done/how they've used or programmed OSS so colleagues can understand this work.

Outsourcing employees also need to provide standard directions on how the recipient of that information should use OSS to continue with work on that project. By developing standards ways of reporting such information to colleagues, organizations decrease the chances that confusion or a programming inconsistency will occur. These comments should also be shared with/downloaded to a database controlled by the client company. This database can help organizations anticipate or backtrack to locate prospective problems related to OSS use in outsourcing. Such a database should also be accessible to any individuals providing help in such outsourcing situations. The information in this database could allow help providers to better understand and address prospective problems encountered by one of more of the outsourcing employees working on forwarded materials (Strategy 3).

- Strategy 6: Require all international outsourcing relationships to be long-term contracts. Many of the disaster stories associated with international outsourcing occurred because of short-term relationships (Goolsby, 2004). In such relationships, neither party feels particularly invested in the long-term success of the other, so the desire to conform to standards or respect copyright is low. Moreover, such short-term relationships often mean that the employment of the outsourcing worker ends just as he or she finally begins to understand and feel comfortable with a client's expectations. By requiring international outsourcing employees who use OSS to contract for longer times, client organizations can retain experienced employees who are accustomed to working according to a company's OSS programming guidelines.

Such long-term relationships also improve accountability in outsourcing, for outsourcing employees are more likely to conform to programming standards and to respect copyright if they wish to maintain long-term business relations with the client (Atwood, 2004). Thus, long-term contracts enhance accountability in international outsourcing situations. Within such long-term relations, client companies should conduct regular

audits of outsourcing employee work to make sure programming and comment guidelines are being followed and that the copyright of materials is respected.

- Strategy 7: Hire in-house (domestic) employees who are proficient in OSS or train in-house (domestic) employees to be proficient in OSS. Such employees will be essential to an organization, for they can serve as the peer mentors and the trainers of international outsourcing workers. They can also serve as the individuals who provide online support to outsourcing employees and as the individuals who will audit OSS work in long-term contract situations.

By having trained OSS employees in house, organizations can rapidly respond to a variety of OSS crisis situations, easily develop new OSS coding practices, and readily share information about new OSS processes. Organizations can also adapt mentoring, training, support, and auditing activities to conform to new computing and business developments. Finally, such in-house employees can help client organizations understand different developments in, successes of, and problems related to OSS when used in an international outsourcing context.

- Strategy 8: Develop a list of "copyright-friendly" outsourcing locations and refer to this list whenever considering international outsourcing. To protect materials from piracy, organizations need to make sure they send such materials to nations that will acknowledge and enforce copyright laws and treaties. Such laws and treaties would provide organizations with a method for stopping such piracy if it happens in another nation and provide a method of recourse for curbing sales of and profits made from pirated materials. For these reasons, organizations can benefit from regularly updated listings that overview what international outsourcing destinations offer the best copyright protections. Such listings should include the following information:
  - If a convention or a treaty on copyright protection exists between the nation of the client organization and that of the outsourcing employee; a sample listing of such copyright-based treaties involving the Unites States can be found at www.copyright.gov — the Web site for the U.S. government's Copyright Office
  - What laws a particular nation has to guarantee copyright protection and how enforceable those laws are, for some nations have copyright agreements that are either not enforced or enforced haphazardly (Doyle, 2004; Orr, 2004)
  - How the client organization should file a grievance or petition according to a particular convention or treaty, or according to the laws of a certain nation
  - With which government agencies — both in the client organization's own nation and the outsourcing employee's nation — such a grievance or petition should be filed
  - What actions to expect in response to filing such a grievance and if any losses can be recovered according to international agreements or national laws

By consulting such a listing in advance of distributing work to overseas outsourcing workers, companies can reduce the risks that copyright violations will occur through restricting the distribution of materials only to employees in nations that provide effective copyright protection.

- Strategy 9: Develop an industry standard for OSS use. While an individual corporate standard can address compatibility issues related to one organization, the OSS-based products created by that organization might be used by other companies within an overall

industry. At this level, the compatibility of OSS products again becomes an issue. To avoid compatibility problems across overall industries, the members of those industries should work together to develop an industry-wide standard for how to code, record comments, offer help, and record suggested solutions to programming problems. These standards could then be shared via an online database available to all members of a given industry. As computer programming practices are continually evolving, such a site should also have a "recent developments" section in which member organizations share new information and practices, innovations, or responses to previously unconsidered problems. Industry members should also meet regularly (annually or bi-annually) to share experiences and to update the related standards for OSS use in international outsourcing practices.

- Strategy 10: Establish and maintain an industry-wide database of individuals and organizations that have violated copyright or violated coding standards in international outsourcing relationships. Several outsourcing experts have noted that the fear of lost employment — and thus lost money — is one of the few ways to force international outsourcing workers to comply with industry standards and procedural (legal) guidelines (Atwood, 2004). The problem with international outsourcing is that an individual might violate the standards or the guidelines of one organization without fear. That is, individuals could think they can simply go to work for a different organization and that new employer would be unaware of an outsourcing worker's past. In this situation, the fear of reprisal through lost income remains low as the international outsourcing employee can easily move on to other clients.

To counter this practice, organizations within an industry should establish a list of outsourcing offenders who have either failed to conform to OSS programming guidelines or who have used OSS to violate copyright. The idea would be to create an easy-access registry (e.g., a Web site) in which companies could record the names of the outsourcing providers with which they worked.

Included in such a registry would be a listing of who the violator is, what the nature of the violation was, what the client did in response to this violation, and what results came from this action. Such a registry could help companies avoid working with "disreputable" outsourcing providers as well as offer effective strategies for addressing violations that occur in outsourcing relationships. These industry-wide registries mean that a large number of companies might now avoid "suspect" outsourcing providers and greatly affect their profits by creating a boycott situation. This fear of boycott could be a powerful incentive for outsourcing providers to abide by data processing practices required by client organizations (Atwood, 2004). Such a site should also allow users to provide a synopsis of the effectiveness with which they felt an outsourcing provider performed work.

For such a registry to be effective and open to the widest range of users, it should be online and allow users to perform internal searches for different outsourcing providers. It should also be updated regularly. Perhaps the best organization to oversee such a registry would be industry oversight bodies or the chamber of commerce or the Better Business Bureau in states where a large number of companies engage in international outsourcing.

These registries would need to provide users with effective instructions on how to report concerns and locate data on outsourcing providers. They would also need to provide different kinds of information to different companies depending on the size of the company (small business or multinational conglomerate) and the related industry.

Such registries could also be made effective by having frequently asked questions (FAQ) sheets or online help functions that facilitate use and provide information on how to address problems related to international outsourcing.

Each of the afore-mentioned strategies is crucial to maintaining consistency and protecting ownership in outsourcing situations involving OSS. This list, however, is by no means comprehensive. Rather, it is a starting point that addresses some of the more fundamental aspects of OSS use in international outsourcing. For this reason, organizations might wish to address new developments in or particular uses of OSS in international outsourcing relationships.

FUTURE TRENDS

At present, international OSS use might seem to be a peripheral topic related to a limited number of activities. Certain trends in business practices and international markets, however, indicate that organizations need to explore their OSS practices now, in order to prepare for and effectively address these trends.

The Projected Demand for Computing and IT Use in International Outsourcing

Among the areas expected to experience the greatest growth in BPO are finance, accounting, and medicine, and all of these areas of growth have one thing in common — data processing that involves computers and information technology. Many managers and bankers in the United States, for example, see international outsourcing as an effective way to address different accounting practices — particularly those related to information technology (IT). This push is in part due to the Sarbanes-Oxley Act of 2002 — section 404 of which requires chief executive officers and chief financial officers of public companies to review their internal controls over financial transactions ("404 tonnes," 2004). The costs and time associated with all of the data crunching related to such activities are massive,1 and IT could greatly help with the compiling of information and the coordination of related activities. One way to curb costs, maximize time, and focus on computing in processing would be to outsource such activities. After all, the outsourcing of accounting and of IT work is not new and could address a lack of needed and costly domestic employees in both IT and in auditing2 ("Relocating the back office," 2003). Given the costs related to such activities and the fact that more complex accounting practices are being outsourced, it seems reasonable to expect that some section 404 activities would be prime candidates for international outsourcing.

New U.S. healthcare legislation also seems poised to increase the international outsourcing related to IT work and data processing. The driving force behind this trend is the Health Insurance Portability and Accountability Act (HIPAA), which requires that all of an individual's medical information be placed in electronic format so it can be easily shared via computer systems (Goolsby, 2001b; Goolsby, 2001c). Under HIPAA, print medical records must be processed into a digital format — a task that is time consuming, costly, and monotonous — and maintained via an IT system for sharing such information. The projected result of such processing is massive IT costs for all health care organizations such as hospitals (Goolsby, 2001a; Goolsby, 2001c). These factors make HIPAA-related tasks, such as medical transcription and IT development and oversight, ideal candidates for international outsourcing. They also make the development and the monitoring of related IT systems prime areas for international outsourcing, especially as most healthcare providers are relatively new to such systems and their operations (Goolsby, 2001a; Salkever, 2004; "Sink or Schwinn," 2004). As so much of this financial and medical related work involves computers, it would be safe to as-

sume that some — or many — of the outsourcing employees working on such projects would be using OSS. For this reason, the development of standard OSS practices becomes essential, not only to achieving organizational objectives, but to meeting legal requirements.

The Value of OSS in International Marketing

OSS also has important implications for how businesses access markets in developing nations. Until quite recently, the majority of individuals within these regions had little or no access to the online environment. That factor of access, however, has changed markedly in recent years, due in large part to a mixture of public and private sector programs designed to increase global Internet use. In China, for example, the number of Internet users grew from 2.1 million in 1999 to a projected 96 million by the end of 2004, and some experts expect this number to balloon to over 200 million users by 2007 ("Wired China," 2000; "China's coming," 2004). In Africa, the United Nations and private companies have undertaken initiatives to increase online access across the continent, and Africa's number of dial-up Internet connections has grown by some 20% in the past two years while the sales of laptop computers remain strong in this part of the world ("Tapping in to Africa," 2000; Kalia, 2001; Reuters, 2002; "Laptop sales," 2004).

In Latin America, Brazil has seen its number of online users grow by 430,000 in recent months, and Global Crossings Ltd. has recently completed a project that uses fiber optics to give, "multinational companies the ability to communicate with Latin America as efficiently as with any other region" ("Active Internet users," 2004; "Tying Latin America together," 2001, p. 9). Additionally, the number of individuals going online in Eastern Europe is expected to climb from 17% to 27% by 2006, and laptop sales in the region remain strong ("IDC research," 2003; "Laptop sales," 2004).

As a result of these developments, companies can now use online media to market goods, supply, and services, and even sell and distribute digital products on an unprecedented scale in the developing world. This concept of scale, moreover, is no small matter, for the poor in many developing nations have a large combined purchasing power. The collective buying power of Rio de Janeiro's poorest residents, for example, is estimated to be some $1.2 billion ("Beyond the digital divide," 2004). The value of such aggregate overseas markets is perhaps the reason that certain companies have started developing online communication technologies that could provide the "less well off" citizens of the world with affordable online access ("Beyond the digital divide," 2004; Kalia, 2001). They have also begun developing inexpensive hubs for online access in nations such as India, Ghana, Brazil, and South Africa (Warschauer, 2003; "Beyond the digital divide," 2004).

Interaction via the Internet and the Web, however, requires the use of software (e.g., a browser) that allows a wide range of individuals to navigate the online environment. For this reason, the only way companies would be able to effectively tap these poorer overseas markets would be through the use of free or inexpensive open source software. If, however, each prospective consumer in developing nations used a different kind of OSS to interact online, then the ability of companies to tap those markets in a mass, and thus a profitable, manner is lost. The development of standards in OSS programming and use provides a mechanism for avoiding such problems. Moreover, by introducing such standards early on and to the more technologically savvy individuals engaged in international outsourcing, organizations can improve the chances that such standards will either "trickle down" or spread throughout an overall region. Thus, the development of international OSS standards has important financial implications that can extend far beyond international outsourcing.

CONCLUSION

International outsourcing offers a variety of cost, time, and quality advantages to those organizations that use it effectively. In relation to the international outsourcing of knowledge work, such effectiveness often involves access to and uses of software. While open source software can provide a distinct advantage in international outsourcing situations, such uses of OSS must be guided by both client companies and overall client industries. Fortunately, uses of OSS in international outsourcing are still relatively new. For this reason, organizations have some time to develop the standards, protocols, resources, and relationships essential to effective OSS use in outsourcing contexts. The ideas presented in this chapter provide the reader with a foundation for exploring such developments. By understanding, employing, and building upon these ideas, organizations can improve the success with which they engage in international outsourcing practices both today and in the future.

REFERENCES

404 tonnes. (2004, December 16). The Economist. Retrieved December 27, 2004, from http://www.economist.com/displaystory.cfm?story_id=3503931

Active Internet users by country, August 2004. (2004, September 22). ClickZ. Retrieved October 6, 2004, from http://www.clickz.com/stats/big_picture/geographics/article.php/3410261

Atwood, M. (2004). The art of governance. Outsourcing Center. Retrieved December 27, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4616

Baily, M. N., & Farrell, D. (2004, July). Exploding the myths of offshoring. The McKinsey Quarterly. Retrieved November 11, 2004, from http://www.mckinseyquarterly.com/article_print.aspx?L2=7&L3=10&ar=1453

Balfour, F. (2005, February 7). Fakes! BusinessWeek Online. Retrieved May 31, 2005, from http://www.businessweek.com/@@ISYAAIQQjU40VAkA/magazine/content/05_06/b3919001_mz001.htm

Bendor-Samuel, P. (2004). Lou Dobbs: Here's why you're wrong! Outsourcing Center. Retrieved December 20, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4565

Beyond the digital divide. (2004, March 13). The Economist: Technology Quarterly Supplement, 8.

Byrnes, N. (2005, January 1). Green eyeshades never looked so sexy. BusinessWeek Online. Retrieved January 5, 2005, from http://www.businessweek.com/@@na*EhYQQxu80VAkA/magazine/content/05_02/b3915041_mz011.htm

China's coming of age online. (2004, November 16). eMarketer. Retrieved November 17, 2004, from http://www.emarketer.com/Article.aspx?1003139&printerFriendly=yes

Clement, R. W. (1994). Culture, leadership, and power: The keys to organizational change. LookSmart. Retrieved March 3, 2005, from http://www.findarticle.com/p/articles/mi_m1038/is_n1_v37/ai_149

Doyle, J. F. (2004). Avoiding outsourcing pitfalls. Outsourcing Center. Retrieved December 12, 2004, from http://www.outsourcingrequests.com/center/jsp/requests/print/story.jsp?id=4626

Farrell, C. (2004, November 22). Giving thanks for offshoring. BusinessWeek Online. Retrieved December 30, 2004, from http://www.businessweek.com/pring/bwdaily/dnflash/nov2004/nf20041122_7377_dbb013

Farrell, D., & Zainulbhai, A. S. (2004). A richer future for India. The McKinsey Quarterly. Retrieved August 16, 2004, from http://www.mckinseyquarterly.com/article_page.aspx?ar=1440&L2+7&L3=10&srid=6&g

Friedman, T. L. (1999). The Lexus and the olive tree. New York: Farrar, Strass and Giroux.

Garten, J. E. (2004, June 21). Offshoring: You ain't seen nothin' yet. BusinessWeek Online. Retrieved December 30, 2004, from http://businessweek.com/print/magazine/content/04_25/b3888024_mz007.htm

Goolsby, K. (2001a). Healthcare's biggest challenge. Outsourcing Center. Retrieved December 12, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=1660

Goolsby, K. (2001b). How to get ready for HIPPA. Outsourcing Center. Retrieved December 12, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=1686

Goolsby, K. (2001c). Perspectives on HIPPA. Dallas, TX: Outsourcing Center.

Goolsby, K. (2004). The disgruntled employee: A holistic model addressing behaviors in outsourcing. Outsourcing Center. Retrieved May 6, 2005, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4631

Hagel, J. III. (2004). Offshoring goes on the offensive. The McKinsey Quarterly. Retrieved November 1, 2004, from http://www.mckinseyquarterly.com/article_page.aspx?ar=1406&L2=1&L3=106&srid=11

Hamm, S. (2004, September 27). Tech's future. BusinessWeek, 82-89.

Hamm, S. (2005, January 31). Linux, Inc. Businesweek Online. Retrieved May 2, 2005, from http://www.businessweek.com/@@p3fqe4UQ4k80VAkA/magazine/content/05_05/b391 8001_mz001.htm

IDC research: Net usage up in Central and Eastern Europe. (2003, February 19). NUA Internet Surveys. Retrieved June 23, 2003, from http://www.nua.com/surveys/index.cgi?f=VS&art_id=905358723&rel=true

Kalia, K. (2001, July/August). Bridging global digital divides. Silicon Alley Reporter, 52-54.

Laptop sales continue to climb in third world. (2004, August 20). eMarketer. Retrieved October 27, 2004, from http://www.emarketer.com/Article.aspx?1003006 &printerFriendly=yes

Lewis, W. W. (2003). Educating global workers. The McKinsey Quarterly. Retrieved November 10, 2004, from http://www.mckenseyquarterly.com/article_page.aspx?ar=1357&L2=7&L3=10

Malik, R. (2004, July). The new land of opportunity. Business 2.0, 72-79.

Nussbaum, B. (2004, September 20). Is outsourcing becoming outmoded? BusinessWeek Online. Retrieved October 11, 2004, from http://www.businessweek.com/print/bwdaaily/dnflash/sep2004/nf20040920_0654.htm?cha

One third of all software in use still pirated, major study finds. (2005, May 18). IDC. Retrieved May 31, 2005, from http://www.idc.com/getdoc.jsp?containerId=prUS00150505

Orr, G. R. (2004). What executives are asking about China. The McKinsey Quarterly. Retrieved October 6, 2004, from http://www.mckinseyquarterly.com/article_pring.aspx?L2=7&L3=8&ar=1478

Relocating the back office. (2003, December 11). The Economist. Retrieved December 20, 2003, from http://www.economist.com/displaystory.cfm?story_id=2282381

Reuters: Internet use increasing in Africa. (2002, October 1). NUA Internet Surveys. Retrieved June 25, 2002, from http://www.nua.com/surveys/index.cgi?f=VS&art_id=905358408&rel=true

Reuters. (2004, July 18). France outsources, Senegal calls. Wired. Retrieved September 20, 2004, from http://www.wired.com/news/print/0,1294,64262,00.html

Reuters. (2004, September 2). Outsourcing's next big thing – Malaysia? News.Com. Retrieved September 7, 2004, from http://news.com.com/2100-1011-5344618.html.

Rosenthal, B. E. (2001). Business risk. Outsourcing Center. Retrieved December 21, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=1685

Rosenthal, B. E. (2004a). How real estate choices affect offshoring decisions. Outsourcing Center. Retrieved December 12, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4718

Rosenthal, B. E. (2004b). META predicts offshoring will continue to grow at 20 percent clips through 2008. Outsourcing Center. Retrieved December 27, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4714

Rosenthal, B. E. (2004c). Why the U.S. and UK are calling South African call centers. Outsourcing Center. Retrieved December 12, 2004, from http://www.outsourcing-requests.com/center/jsp/requests/print/story.jsp?id=4717

Salkever, A. (2004, July 7). Racing to cure sickly medical security. BusinessWeek Online. Retrieved December 30, 2004, from http://www.businessweek.com/print/technology/content/jul2004/tc2004077_9847_tc_171

Sandholm, L. (2004). Strategic responses for customer satisfaction. Sandholm Associates. Retrieved March 1, 2005, from http://www.sandholm.se/artiklar/stratrespforcustasais.html

Sink or Schwinn. (2004, November 11). The Economist. Retrieved December 6, 2004, from http://www.economist.com/printedition/PrinterFriendly.cfm?Story_ID=3351542

Still, B. (2004). An open source primer. In K. St. Amant & P. Zemliansky (Eds.), Internet-based workplace communications: Industry and academic applications (pp. 278-298). Hershey, PA: Information Science Publishing.

Tan, V. S. L. (2000, August 26). Lessons from culture change. New Straits Times. Retrieved February 26, 2005, from http://adtimes.nstp.com.my/jobstory/aug26a.htm

Tapping in to Africa. (2000, September 9). The Economist, 49.

Tying Latin American together. (2001, Summer). NYSE Magazine, 9.

Warschauer, M. (2003). Technology and social inclusion: Rethinking the digital divide. Cambridge, MA: MIT Press.

Weir, L. (2004, August 24). Boring game? Outsource it. Wired. Retrieved September 20, 2004, from http://www.wired.com/news/print/0,1294,64638,00.html

Wired China. (2000, July 22). The Economist, 24-28.

Zoellick, B. (2001). CyberRegs: A business guide to Web property, privacy, and patents. Boston: Addison-Wesley Professional.

ENDNOTES

[1] General Electric, for example, spent some $30 million in extra payments to auditors to review such documents, while J. P. Morgan Chase has 130 full-time employees working on this project, and PriceWaterhouse Coopers has spent some $40 million in training 9,000 U.S. employees to perform these functions (404 tonnes, 2004).

[2] There do not appear to be enough trained U.S. auditors available to meet current demands, and as a result, this lack of supply has driven up U.S. auditor pay by some 10-20% (Byrnes, 2005).

This work was previously published in Outsourcing and Offshoring in the 21st Century: A Socio-Economic Perspective, edited by H. Kehal, pp. 229-247, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 5.11
How to Create a Credible Software Engineering Bachelors Program:
Navigating the Waters of Program Development

Stephen Frezza
Gannon University, USA

Mei-Huei Tang
Gannon University, USA

Barry J. Brinkman
Gannon University, USA

ABSTRACT

This chapter presents a case study in the development of a Software Engineering (SE) Bachelor's Degree program. It outlines issues in SE program development, various means to address those issues, and explains how the issues were addressed in the initial and ongoing development of an undergraduate SE program. By using SEEK and SWEBOK as requirements sources to define what an undergraduate software engineer needs to know, the authors walk through the creation of a sample curriculum at a small, comprehensive university in the United States. Both the current and initial curricula are presented. The article discusses many items to consider in the process of planning and launching a new BSSE program, such as accreditation, curriculum guidelines, sources of information, and potential problems.

INTRODUCTION

Software Engineering is one of the newer engineering disciplines to emerge. Starting with the coining of the 'Software Engineering' term in 1968

(Naur, 1969), there has been continual growth in interest in software engineering education. Initially, these efforts were primarily at the graduate level, serving software engineering practitioners with undergraduate degrees in Computer Science, Computer Engineering or other related fields. In 1998, in recognition of the needs of bachelors-level computing graduates, the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) established the Joint Task Force on Computing Curricula 2001 (CC2001) to undertake a major review of curriculum guidelines for undergraduate programs in computing (Diaz-Herrera, 2004). This and other efforts (EA, 2007; CEAB, 2006; ABET, 2005) added official recognition of the need for the establishment of effective undergraduate programs preparing students to become software engineers.

The underlying assumption is that creating a new degree program for a relatively new discipline (Software Engineering), in a professional area (Computing) that already has several well-established disciplines (Computer Science, Computer Engineering, Information Systems, etc.) necessarily comes with a number of significant development risks. This chapter takes the form of an extended experience report, in the hope of presenting an overview of these risks, and practical means to mitigate them. This work is primarily based on the authors' experience in developing a software engineering undergraduate program leading to a Bachelor of Science degree in Software Engineering (BSSE) at a small comprehensive university in the United States (Frezza, 2006). Effort has been made to generalize this experience, and include questions and issues encountered in other SE program development efforts, as well as raising issues that may be more critical in other organizational settings.

ISSUES IN SE PROGRAM DEVELOPMENT

Developing a new undergraduate program, particularly one like Software Engineering that does not have long-established definitions can be (and for us was) a delicate business. Among the key stakeholders for a new SE program, the requirements for what belongs in such a major may not be well understood, or easily communicated. In all, our program development effort was similar to many of our software development experiences, in that the requirements management activities were significant, messy, and working to resolve them early proved worthwhile. Our undergraduate software engineering program, at the time of writing, has been developed, launched, gone through several on-going outcomes reviews, and we are currently preparing our first accreditation self-assessment.

Based on our reflection on the issues we encountered, and our post-design assessments, some of the key issues we've found in developing a new SE program include:

- **Organization:** Determining where the program is housed or sponsored within the institution
- **Vision:** Defining the style, or professional focus of the program
- **Accreditation:** Applying international and national standards to ensure program quality
- **Curriculum:** Designing the academic plan for students to meet or exceed the vision, and
- **Finding help:** Locating contacts to support program development

Organization

Determining where an SE program is housed is important to its success. The issue centers on

faculty ownership of the program, and administrative support for the students. Many SE programs are organized in the same academic housing as Computer Science programs, but this is not universally the case. At issue is the blend of CS, IS, and Engineering courses currently available, and ability to work with the faculty delivering these courses to be effective for the new program. While Software Engineering is normally classified as a computing sub-discipline, in many institutions computing disciplines may be in multiple departments scattered across multiple schools, or not. The character and ability of the various departments and schools to collaborate (*e.g.,* School of Engineering *vs.* School of Arts and Science, etc.) may not be easily navigated, and can delay program introduction.

The housing issue for a program is significant, as it can affect issues related to shared course content, accreditation, funding, hiring, tenure, and a plethora of other subtle and inter-related academic issues, not the least of which is the culture of the faculty leading the program. Mismatches can adversely affect program development, but more especially student learning and faculty retention issues.

While many programs are initially housed in an existing department structure, in several instances, sponsoring departments have been cross-department, or even cross-college arrangements. In some cases these more complex structures, created to launch the SE program, were later replaced. Factors that can affect complex administrative structures include growing enrollments, competition, budgets and funding, faculty issues and other sustainability factors. In some schools, the more complex structures proved workable, and have been maintained (e.g., Drexel). The common theme is the ability to gain sufficient institutional agreement for offering SE course and related program content.

In our case, this negotiation of where to house the program led to delaying the program launch by about a year. Our initial proposal was to run the new SE program with a systems orientation from the Electrical and Computer Engineering (ECE) department. What was at issue was the nature of software engineering – few faculty having significant experience beyond embedded software development, or exposure to the significant and world-wide efforts to define software engineering as a discipline (Bourque, 2000). Locating authoritative guidelines as to what a software engineering undergraduate program should include was significant to this negotiation. Even with these guidelines, identifying the nature of our new SE program, and where it should be developed/housed was by no means a simple process.

Addressing this housing issue led to several surveys and presentations, using materials from conferences (Diaz-Herrera, 2001), ABET program-specific criteria (ABET, 2005), the Certified Software Development Professional (CSDP) effort (McConnell, 1999; IEEE CS, 2001), SWEBOK drafts (Abran, 2004), and the SE2004 drafts (Diaz-Herrera, 2004) to define software engineering for students, faculty and administrators. In particular, SE2004, SWEBOK, and ABET proved to be the most useful, and served as authoritative guidelines for our proposal development. At the end of these discussions, even though the program proposal originated from our ECE department, the strong computing focus of the SE program was deemed more suitable to be housed in the Computer and Information Science (CIS) department offering our computer science (CS) and management information systems (MIS) degree programs.

As the CIS department was housed in the same school as ECE, no administrative objections were encountered. The new task was to redevelop the program vision and program details with a team of primarily CIS faculty in a way that would succeed when the new program was launched and managed from the CIS department.

The decision to house the program in a different department meant ECE relinquishing control on the proposal, the proposal champion working with a new department chair and new faculty partners. The benefit of this redevelopment work was the promise of building consensus around a shared vision from those who would ultimately deliver the program.

Vision

Following a well-documented SE best practice, identifying a coherent vision was a useful starting point, and our experience confirmed that it is a key factor for success in developing a Software Engineering undergraduate program. Within the vision for an academic program, one of the most fundamental issues is the judicious selection of the type, or character of the program that is desired. Notwithstanding other sources, at least six application models for software engineering have been identified (Jones, 2003):

- **Military:** Applications built according to military or US Department of Defense standards. This may include weapons systems, but also logistics and non-military systems that use military standards.
- **Systems:** Applications developed to control hardware devices such as computers, aircraft, telephone switches, and other physical devices and products, including embedded systems.
- **Commercial:** Applications for lease or sale to external customers, occasionally referred to as 'shrink-wrap' software. This category includes many personal computer applications, but also includes larger mainframe applications.
- **Outsourced:** Applications developed for a specific client company under a contract. Because of contractual obligations and the possibility of litigation, outsourced projects have some additional activities in comparison to in-house development.
- **Management Information Systems (MIS):** Applications built to control major business functions such as accounting, marketing, sales, and personnel. This category includes many traditional mainframe applications, but also the more recent client-server, multi-tiered and web-based applications.
- **End-user development:** Small applications that various kinds of knowledge workers—such as accountants, engineers, or project managers—build for personal use.

In developing a vision for a specific program, casting the nature of the program may be decided by other factors, such as faculty availability, skills, and influences from local and regional employers. These can, and should influence what a specific SE program graduate should be able to do. However, this 'local' approach can easily ignore the other external 'requirements' for a credible program.

There exist broad and relatively well-developed, and reasonably authoritative guidelines for what software engineers need to know (See the *Defining the BSSE Graduate* and *What Software Engineers Need to Know* subsections that follow). However, the context in which these skills are developed is also important, as these different application models have differences that can be significant in program delivery. These differences will typically show up in the determination of the content of required upper-level courses and elective courses.

Developing a vision is important in that the vision statement, once agreed to, serves as a useful guide in helping to sell the program to different academic and administrative stakeholders, as well as a useful reference during program design. Like most business exercises in vision, developing this as a shared vision, rooted in the realistic limitations of the organization will help reduce the risk of failure.

In our case, the decision to house the program in the CIS department led to a new shared vision for our SE program. The initial ECE-based proposal was based on a *systems* focus. With CIS faculty participation, the revised program proposal focused on delivering skills and knowledge for the *outsourced* and *MIS* categories. These represented trade-offs among the program developers, recognizing that the o*utsourced/MIS* view would have distinctly different courses and flavor from the *systems* view. While different from the initial vision, this was a vision that was both legitimate, and would work well within the exiting department and course structures then in place. The tradeoffs allowed the new proposal to parallel the CS program more closely, but with the recognition that software engineering concepts needed to be integrated into courses taken jointly by CS and MIS students. Ultimately, this has proven to be useful in our situation, as the integration of software engineering concepts into early courses was easily negotiated, and continues to be well received.

Accreditation

In the US and many other countries, the use of the term "Engineering" in a program or degree title is necessarily accompanied by the requirement for some form of national accreditation which serves to ensure program quality. In the United States, ABET, Inc. (ABET, 2007) is responsible for the specialized accreditation of educational programs in applied science, computing, engineering, and technology. In Canada, the Canadian Engineering Accreditation Board (CEAB), serves as the accreditation body for engineering programs (CEAB, 2006), while Engineers Australia (EA) is responsible for this service in Australia (EA, 2007). This is an important initial consideration for creating a new program, because the nature of engineering accreditation generally brings with it required documentation processes, criteria and even academic culture that may be foreign to the institution or sponsoring department. Taking the time to become familiar with the processes and documentation needed is important, and in some instances may require hiring consultants to review academic proposals.

At the time of writing, 13 such programs were accredited in the United States (ABET, 2006), 12 in Canada (CEAB, 2006), and 18 in Australia (EA, 2007). To give a sampling of the breadth of universities that have chosen this route, these universities are listed in Figure 1. In our case, the goal was to create a program that would warrant including Gannon University in the list.

While not all programs require national accreditation, international guidelines exist (such as SE2004) to help ensure the quality of Software Engineering undergraduate programs. In our case, accreditation was a process new to the sponsoring department, but not to the school. Experience in applying the ABET criteria was easy to find, and the use of the EAC accreditation criteria (ABET, 2005) and the related SE2004 volume served as significant drivers in assessing the quality of the program proposal. This validation of the curriculum development process was extremely valuable in describing software engineering to program stakeholders, and served as a very useful means of assessing changes to the program (Frezza, 2006). These processes as we applied them are described in more detail in the pages that follow.

Curriculum Development

Ultimately one of the most critical portions of the program delivery is the curriculum employed by the program. The curriculum development process includes the development of the program objectives, as well as courses and course objectives. In most institutions, these are developed within the framework of institutional standards, as well as existing computing, mathematics, engineering and other courses that would also be taken by SE students. This key issue is discussed in much more detail in sections which follow.

Sources of Help and Advice

As in most engineering endeavors, one of the most useful sources of development information comes from others who have developed similar products (Kelley, 1999). These resources are particularly useful for helping to understand issues, avoid issues, or also experience to resolve issues as they are encountered in your program development. This is also a significant source of external expertise that can be used to help validate the program, such as a 'blue-ribbon' or other external panel that can validate or provide guidance to program development.

In our case, finding help in the form of the Working Group on Software Engineering and Education, and the more recent Software Engineering Program Leaders Association (SEPLA) proved to be extremely useful for helping find and share materials to explain the SE profession to various constituents (faculty, students, administrators), as well as provide useful market data and comparison programs. Various SEPLA members also volunteered, and provided input on various program proposals. The SEPLA listserv is available at sepla@listserv.butler.edu.

In our experience, the faculty development wherein we used authoritative guidelines to define Software Engineering for ourselves was absolutely essential. The need for this education came initially in response to addressing our organization issues. Our Accreditation goal dictated that ABET criteria needed to be considered, but the more extensive international guidelines

Figure 1. Accredited software engineering related programs in the U.S., Canada, and Australia (current as of Sept. 2007)

US (*ABET*)	Canada (*CEAB*)	Australia (*EA*)
Auburn University	University of Calgary	Australian National University
Clarkson University	Carleton University	Curtin University of Technology
Embry-Riddle Aeronautical University, Daytona Beach	Concordia University	Flinders University
Fairfield University-School of Engineering	Lakehead University	Griffith University Nathan Campus
Florida Institute of Technology	McMaster University	La Trobe University (Bundoora campus)
University of Michigan-Dearborn	University of New Brunswick	Monash University
Milwaukee School of Engineering	University of Ottawa	Murdoch University
Mississippi State University	University of Waterloo	RMIT University
Monmouth University	University of Western Ontario	Swinburne University of Technology
Pennsylvania State University, Behrend College	École de technologie supérieure	University of Canberra
Rochester Institute of Technology	Laval	The University of Melbourne
University of Texas at Arlington	Polytechnique	The University of Newcastle
University of Texas at Dallas		The University of New South Wales
		The University of Queensland
		The University of Western Australia
		University of Southern Queensland
		University of Sydney
		University of Technology, Sydney

(SWEBOK and SE2004) were more informative. The process of blending these viewpoints helped establish detailed 'requirements' for our BSSE program, as well as establish agreement on these requirements. This analysis work was crucial to the success of the program proposal process, and became central to developing the program outcomes, expectations, and ultimately its detailed design. Our particular blending is summarized in the *Defining the BSSE Graduate* and the *What Software Engineers Need to Know* sections that follow.

While the authoritative sources (ABET, SE2004 and SWEBOK) are all aimed at different target audiences, the definitions of software engineering they provide for these audiences are important. These definitions all speak directly to what the expectations would be for our BSSE graduates *after* the program was established, hopefully accredited, and they were well into their careers. Because the definitions were external, they carried significantly more weight than the viewpoint of any particular faculty member. These definitions, once blended, became central to developing the shared vision for our BSSE program as the program was developed, and has since continued to support outcomes assessment and program enhancement.

DEFINING THE BSSE GRADUATE

For the purposes of creating a quality, accreditable program, it is essential to define the desired knowledge and skills possessed by the BSSE graduates. One of the more useful forms for defining the desired knowledge and skills are the "outcomes" for the program. Outcomes relate to broadly defined skills, knowledge, and behaviors that students should acquire as they progress through the program (Wankat, 1993). If a graduate achieves all the program outcomes, this indicates that the student meets the program's stated educational objectives and is equipped to function as expected of a BSSE graduate.

To create an accreditable SE program, the program design must meet the established educational objectives and program criteria for a BSSE program. There are at least two primary sources for these objectives and criteria which define the minimal knowledge and skills for a BSSE graduate. In the US, the Engineering Accreditation Commission (EAC) provides two categories of objectives and criteria that apply to the design of engineering programs (ABET, 2005).The first category, the EAC Program Educational Objectives, is a broader set which applies to all engineering programs. The second category provides each engineering discipline a unique set of program criteria specific to the discipline.

While these objectives and criteria are definitive for accreditation in the U.S., they do not provide as much detail as the SE2004 guidelines (Diaz-Herrera, 2004). The difficulty is that when comparing the SE2004 Student Outcomes (Diaz-Herrera, 2004) with the related EAC Program Educational Objectives (ABET, 2005) and EAC SE Program Criteria (ABET, 2005), there are noticeable gaps among them (Frezza, 2006). However, the superset of related SE skills indicates that a program should provide at least the following outcomes (Frezza, 2006):

- Show mastery of the software engineering knowledge and skills, and professional issues necessary to begin practice as a software engineer
- Demonstrate the ability to appropriately apply science, discrete mathematics, empirical techniques, probability and statistics and relevant topics in computer science and supporting disciplines to the development of complex software systems

- Work as an individual and as part of a multidisciplinary team to develop and deliver quality software artifacts
- Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems and organizations
- Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal and economic concerns
- Understand professional and ethical responsibility
- Demonstrate an understanding of and apply current theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, and documentation
- Demonstrate an understanding and appreciation for the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment
- Learn new models, techniques and technologies as they emerge and appreciate the necessity of such continuing professional development
- Obtain knowledge of contemporary issues
- Receive and internalize a broad education necessary to understand the impact of engineering solutions in a global, economic, environmental, and societal context

These program outcomes, which arguably must be met for a U.S.-based program, provide a useful definition of what the BSSE graduate should know and be able to do. Although important for defining and continually improving program effectiveness, the outcomes by themselves don't provide adequate detail about the specifics for "software engineering knowledge and skills". What specifically should we teach students? What should the courses contain? SE fortunately has other sources that define more specifically what students must know and be able to do.

WHAT SOFTWARE ENGINEERS NEED TO KNOW

In order to craft a credible Software Engineering curriculum that also paves the way to students' success in the workforce, we need to understand what knowledge students are expected to possess. Not only does this include knowledge that students fresh out of college are expected to know, but also knowledge that these students after a few years in the workforce are expected to hold. There are two primary sources for these requirements, SE2004 (Diaz-Herrera, 2004) and SWEBOK (Abran, 2004), addressing aforementioned types of knowledge respectively.

- **SE2004 (Software Engineering 2004):** defines the body of knowledge that every software engineering degree graduate fresh out of college needs to know as the *Software Engineering Education Knowledge* (*SEEK*).
- **SWEBOK (Guide to the Software Engineering Body of Knowledge):** characterizes the contents of software engineering discipline, i.e., the knowledge needed for the practice of software engineering after four years in the workforce.

Both of these documents carry with them extended development processes and improvements. In addition, both development efforts included significant efforts to ensure that the documents, and thus the educational patterns that might emerge from them, were not US-centric. The SE2004 effort, in particular, has been translated into Russian to support curricular development efforts in Central and Eastern Europe (Pavlov, 2006). At the time of this writing, new curriculum

pilots based on SE2004 have been started in over 30 Central and Eastern European universities (Sobel, 2007).

SE2004

The Joint Task Force on Computing Curricula sponsored by the IEEE Computer Society and the Association of Computing Machinery Joint Task Force developed *Software Engineering 2004 (SE2004)* as curriculum guidelines for undergraduate degree programs in software engineering (Diaz-Herrera, 2004). SE2004 defines a detailed set of knowledge expected of a BSSE graduate as the *Software Engineering Education Knowledge* (*SEEK*). SEEK is designed as a guide to support the development of undergraduate software engineering education curricula.

SEEK defines 10 education *knowledge areas* (*KAs*), each of which is recognized as a significant part of the body of knowledge that every bachelors-level software engineering graduate needs to know. A short description for each of the ten knowledge areas defined by SEEK (Diaz-Herrera, 2004) are listed below:

1. **Software evolution.** "Software evolution is the result of the ongoing need to support the stakeholders' mission in the face of changing assumptions, problems, requirements, architectures, and technologies." (Diaz-Herrera, 2004)
2. **Software process.** "Software process is concerned with knowledge about the description of commonly used software life-cycle process models and the contents of institutional process standards; definition, implementation, measurement, management, change and improvement of software processes; and use of a defined process to perform the technical and managerial activities needed for software development and maintenance." (Diaz-Herrera, 2004)
3. **Software verification and validation.** "Software verification and validation uses both static and dynamic techniques of system checking to ensure that the resulting program satisfies its specification and that the program as implemented meets the expectations of the stakeholders." (Diaz-Herrera, 2004)
4. **Software quality.** "Software quality is a pervasive concept that affects, and is affected by all aspects of software development, support, revision, and maintenance. It encompasses the quality of work products developed and/or modified … and the quality of the work processes used to develop and/or modify the work products." (Diaz-Herrera, 2004)
5. **Software design.** "Software design is concerned with issues, techniques, strategies, representations, and patterns used to determine how to implement a component or a system. The design will conform to functional requirements within the constraints imposed by other requirements such as resource, performance, reliability, and security." (Diaz-Herrera, 2004)
6. **Software management.** "Software management is concerned with knowledge about the planning, organization, and monitoring of all software life-cycle phases." (Diaz-Herrera, 2004)
7. **Computing essentials.** "Computing essentials includes the computer science foundations that support the design and construction of software products." (Diaz-Herrera, 2004)
8. **Software modeling and analysis.** "Modeling and analysis can be considered core concepts in any engineering discipline, because they are essential to documenting and evaluating design decisions and alternatives. Modeling and analysis is first applied to the analysis, specification, and validation of requirements." (Diaz-Herrera, 2004)

9. **Mathematical and engineering fundamentals.** "The mathematical and engineering fundamentals of software engineering provide theoretical and scientific underpinnings for the construction of software products with desired attributes." (Diaz-Herrera, 2004)
10. **Professional practice.** "Professional Practice is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner." (Diaz-Herrera, 2004)

Each knowledge area (KA) is further divided into smaller modules called *units*. The left column in Figure 2 lists the SEEK knowledge areas in light grey shades, and the knowledge units (KUs) defined for each KA in italics.

Each knowledge unit defined in SEEK is further divided into *topics*. Some topics are designated as 'essential', and constitute the core knowledge which is considered required for anyone to obtain a software engineering undergraduate degree. In its current (2004) revision, SE2004 defines 240 topics as essential that software engineers graduating from credible programs need to know (Diaz-Herrera, 2004). A summary of the number of units, topics, essential topics, and contact hours for essential topics are listed in Table 1.

The topic-level detail outlined in Table 1 can be a two-edged sword for program design. With the rigorous application of the topic-level information, those developing new software engineering programs may well find that the credit hours needed to cover the 'essential' units would be well beyond the ability to offer a program within most University constraints. Similarly, this 'essential' detail can be too detailed for effective course planning, and can obscure what units are more essential than others, particularly in the context of making a program unique to an institution (Frezza, 2003). Conversely, the detail facilitates definition of what is meant by particular knowledge units, and thus strongly facilitates measuring the completeness of a program, and clarity in communicating what constitutes a BSSE degree.

Besides the undergraduate education knowledge defined by SEEK, we also need to know about what kinds of knowledge are needed for the practice of software engineering in the workforce. The guide to the Software Engineering Body of Knowledge (SWEBOK) does just that.

SWEBOK

The IEEE Computer Society established a baseline for the body of knowledge and recommended practices for the field of software engineering in the Guide to the Software Engineering Body of Knowledge (SWEBOK) (Abran, 2004). SWEBOK characterizes the contents of software engineering discipline, *i.e.* the knowledge needed for the practice of software engineering after four years in the workforce, into ten Knowledge Areas (KAs). Each knowledge area is further divided into subareas, and each subarea is further divided into topics and subtopics. The right column in Figure 2 lists the SWEBOK knowledge areas in light grey shades, and the knowledge subareas defined for each KA in italics.

The KAs for both SEEK and SWEBOK are highlighted in light grey shades in Figure 2. The double arrowed lines given in Figure 2 outline the similarities, the dashed lines outline partial coverage, while KAs without links indicate the differences between SEEK and SWEBOK KAs. As you can see from the figure, for the first six SEEK knowledge areas (KA) each has a very related KA in SWEBOK as indicated by the double arrowed lines. Typically the KA in SWEBOK has a broader coverage in topics than the corresponding SEEK KA. However, the SEEK knowledge units listed in light grey are not covered by its knowledge area's corresponding SWEBOK KA, but are covered by the SWEBOK KA to which it is linked with solid lines. For example, SEEK unit *Software Configuration*

Figure 2. SE2004 SEEK knowledge areas and units vs. SWEBOK knowledge areas and subareas

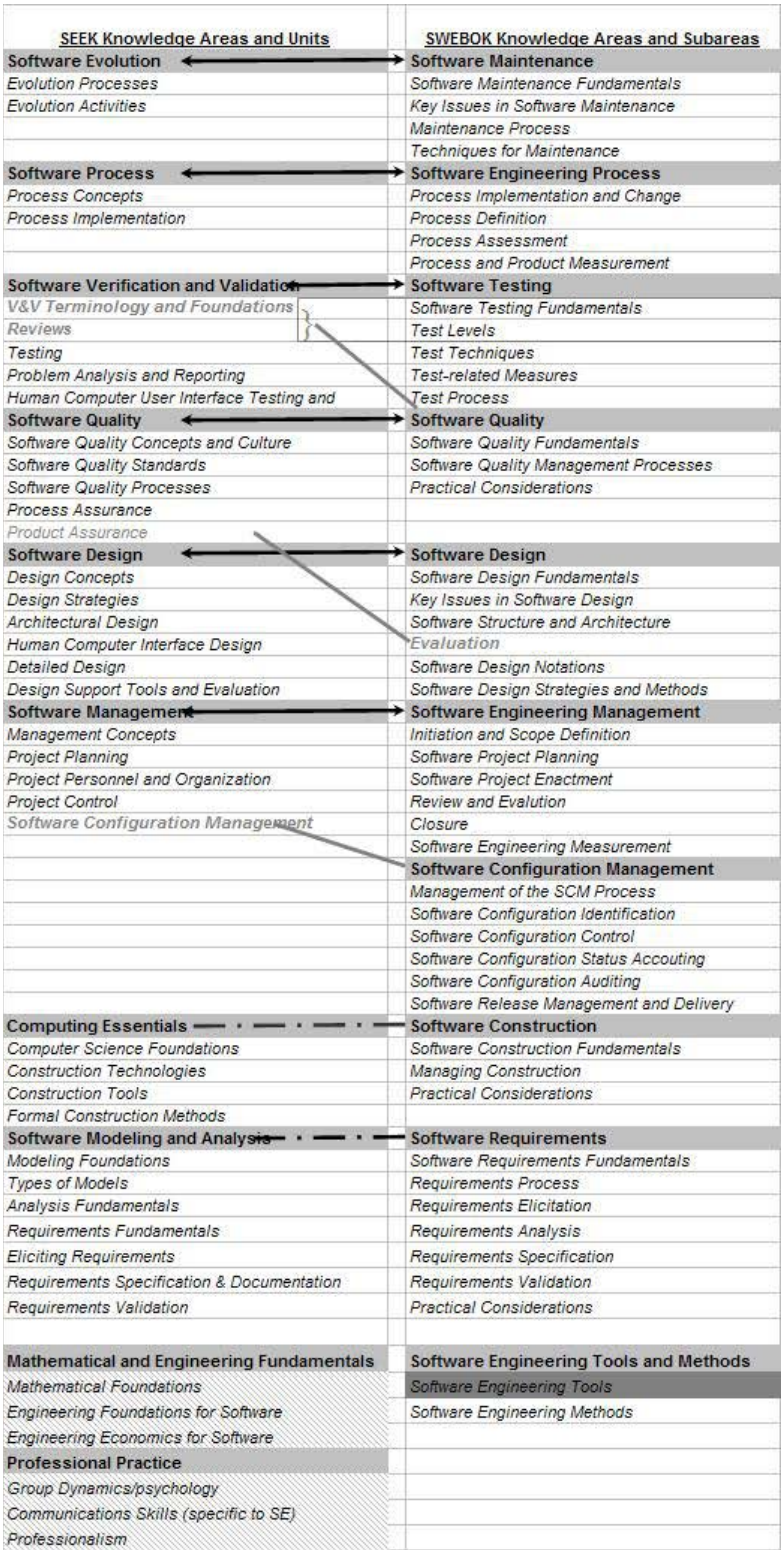


Table 1. Summary of SEEK knowledge areas, units and topics

SEEK Knowledge Area	Units	Topics	Essential Topics	Essential Contact Hours
Computing Essentials	4	42	37	172
Mathematical and Engineering Fundamentals	3	22	19	89
Professional Practice	3	17	17	35
Software Modeling and Analysis	7	42	33	53
Software Design	6	37	31	45
Software Verification and Validation	5	30	28	42
Software Evolution	2	13	9	10
Software Process	2	14	13	13
Software Quality	5	28	25	16
Software Management	5	31	28	19
Total	**42**	**276**	**240**	**494**

Management in *Software Management* KA is covered by SWEBOK *Software Configuration Management* KA. SEEK unit *Product Assurance* in *Software Quality* KA is covered by SWEBOK subarea *Software Design Quality Analysis and Evaluation* in *Software Design* KA.

In addition to the closely related SEEK and SWEBOK KAs mentioned above, two SWEBOK KAs, each having significant overlap with but only partially covering its corresponding SEEK KA, are shown in dashed lines. SWEBOK KA *Software Requirements* only covers requirements related units in SEEK KA *Software Modeling and Analysis,* while *Software Construction* covers construction related units in *Computing Essentials.*

Noticeable unit differences between SEEK and SWEBOK are highlighted in reverse diagonal shades. Both *Mathematical and Engineering Fundamentals* and *Professional Practice* SEEK KAs do not have corresponding SWEBOK KAs due to the educational nature and curricula development purpose of SEEK. As the SWEBOK focuses on the boundary of software engineering, hence non-software engineering-specific knowledge, such as the fundamental background required to acquire software engineering specific knowledge, was intentionally left out.

Another noticeable difference is in the SWEBOK *Software Engineering Tools and Methods* KA as highlighted in dark grey shades. The *Software Engineering Tools* subarea is embodied inside the *Software Evolution, Software Process, Software Verification and Validation, Software Quality, Software Design, Software Management, Computing Essentials, Software Modeling and Analysis, Mathematical and Engineering Fundamentals* and *Professional Practice* topics in SEEK, as highlighted in dark grey shades.

Both SEEK and SWEBOK define the specific knowledge and skills required of a software engineer. For undergraduate software engineering graduates fresh out of college, the knowledge they attain comes from the courses they complete in their curricula, hence curricula plays an important role in deciding what knowledge students will posses when they graduate. While both SEEK and SWEBOK are designed as a guide/foundation for software engineering curricula development, SEEK is especially designed for undergraduate software engineering curricula development with detailed topics defined, and an expectation

that accredited programs will reflect this set of knowledge and skills in their program.

SEEK provides description for each KA, but no descriptions are provided for the units and topics defined. This kind of set up could be hard for syllabus development as the contents which should be included for the name provided for units and topics could be open for interpretation. With no reference materials provided, it could be difficult for faculty to find suitable textbooks or materials to cover the desired topics. On the other hand, SWEBOK provides detailed description and interesting discussion for each KA, subarea, topic and subtopic defined as well as links to books and articles.

STARTING THE PROCESS

The practical process of developing a vision for a specific instance of a Software Engineering program includes a number of concerns to be addressed, such as focus, style, leadership, and the requirements derived from the institutional strengths, weaknesses and opportunities. The experience of the authors is that of developing a program within the context of existing computing and engineering programs within the sponsoring institution, so issues concerning the creation of new academic structures will only be by inference.

Project Leadership

The immediate starting point is necessarily one of leadership – who will lead the project to define and launch the program. While this may seem trite, clearly defining the academic stakeholders is critically important, as the risks of not involving appropriate representation from related academic departments early in the program development are real, and have proven to be stumbling blocks to SE program development. As with any successful project, establishing executive sponsorship at various levels is key, as is communication with the executive sponsors and other stakeholders. One mechanism for supporting ongoing validation, communication and development of the program is a steering committee.

An effective program development steering committee should follow effective patterns within the institution, and typically is formed from department chairs, experienced faculty, external advisors, and anyone else deemed appropriate to the institution. This committee, whether formal or informal in its makeup, should necessarily include persons who have the authority to formally propose a new program within the institution. In some cases, this process may require cross-college cooperation, and thus may also include either academic deans or their representatives.

The steering committee should at minimum approve program development decisions, but may (as in our situation) be significantly more active in the development of the program details. Among these, determining the expected style of the program was a significant set of decisions. For example, in our case, the deliberate choice was to not require co-ops, and to not focus on one particular SE style, such as embedded systems, but rather allow styles and domains to be student-selected via the use of technical electives.

In our development process, the project started with one faculty project leader, and eventually formed a development committee after the housing (which department) issue was settled. In the case of Butler University in Indiana, the housing department was clear, and they developed an external advisory board consisting of local software industry leaders which helped significantly in crafting the program and building internal credibility (Henderson, 2003).

Capitalizing on Institutional Strengths

Each educational institution has its own set of distinguishing characteristics, including things

like the faculty, teaching style(s), history, physical location, etc. Part of the success of a new program is its ability to realize these characteristics within the program in ways that strengthen the program, its appeal to students and its effectiveness for graduates.

In many cases, these institutional strengths are easily recognized, and involve institution-wide structures to support them. These structures can take on many forms: core curricula, freshmen sequences, service learning, marketing, development, alumni services, cooperative arrangements, etc. While many of these academic and non-academic features may also act as constraints, they are also what bring the institutions' unique stamp to the new program. Clearly identifying and celebrating these institutional strengths are significant for marketing the program, both internally and externally. Performing a formal Strengths, Weaknesses, Opportunities and Threats (SWOT) analysis may be useful.

One of the more common SE program development questions that hinges upon institutional strengths is that of requiring cooperative employment placements as part of the academic program. In some institutions, such as Rochester Institute of Technology (RIT) and Drexel University, this is an institutional strength, and is required of most programs. In other schools, required co-ops are common, and institutional support is readily available. Yet this is not the case in most institutions – neither the culture nor the academic structures support co-ops, so the decision to include a required co-op placement as part of the program design may involve significantly more cost to the program.

In our case, similar to that of Butler University, we developed our BSSE program in a 'liberal arts' institution, where the general education requirements included 36 semester hours of general education, and provides a significant institutional 'stamp' to the program. Similarly, there was no institutional support for required co-op placements, and despite the attraction of such an arrangement, it was deemed unfeasible by the development steering committee.

One of the more important institutional limitations to be negotiated is the availability of faculty resources; some institutions are very risk-adverse, and consequently are very reluctant to invest in new faculty positions for new programs until the enrollment proves the need. Other institutions are more accepting of risk, and are more tolerant of investment that will help distinguish a program, and ensure its early success.

In our case, after creating the initial academic plan, we projected the faculty resources needed to develop and sustain the new program. With an enrollment estimate, the request took the form of one new faculty member the year after the program launch, and another faculty member two years after launch if enrollment met or exceeded the estimates.

CURRICULUM CONSTRUCTION AND DESIGN

Students gain and build knowledge and skills from the courses they take while in college. Curriculum dictates what courses students in a specific program should take and the sequence of taking them. Hence curriculum plays an important role in determining what *knowledge and skills* students should possess when they graduate with a bachelor's degree. On the other hand, each course students take has a specific set of course objectives that students completing the course are expected to accomplish; curriculum also plays a determining role in what *program outcomes* will be achieved through the course outcomes. Furthermore, curriculum is the place where each institution showcases its strength, uniqueness and special program focus. So what courses should be included in an institution's BSSE curriculum? There is no one easy answer for that.

For the purpose of creating an accreditable program, where graduates meet or exceed expected

program outcomes, in our experience, there are several factors to consider during the curriculum design and construction process.

1. Institutional/university strength and constraints

Each university/educational institution has its unique set of characteristics as discussed in the *Capitalizing on Institutional Strengths* subsection. No matter what the characteristics are, some (such as faculty, teaching style(s) and co-op arrangement) are great for marketing as the unique strength of the institution while others (such as core curricula, co-op requirement and freshmen sequences) can be considered as constraints for curriculum development.

In our case, our university mandates 36 credit hours of general education which ensures that every graduate achieves the university outcomes (Frezza, 2003). Although the number of credits required is mandated, every program does have the autonomy of restricting course(s) that students can select from each category to better suit the needs of each program. Furthermore, our SE program does not have co-op requirement which opens up credit hours for curriculum. On the other hand, if a co-op arrangement is instituted by the university, the number of credit hours required for the co-op arrangement would be a constraint for curriculum design.

An area where this can be significant is that of total credit hours – in our case, the Computer Science and other Engineering programs all require over 130 semester credits, so this afforded a bit of room in creating courses for the SE program. Many schools have very specific credit constraints that can make this more difficult, e.g., in the United States, many schools have 128 credit limits for bachelors degree programs. As in our case, using related programs in the school (such as EE or CS) as patterns can prove to be effective. Many of the patterns these existing programs have, such as when they take general education courses, the patterns for lab courses and lab credits, the patterns for co-ops, the patterns for common and discipline-specific courses can prove to be useful. In our case we used the CS degree as the pattern, and reused as many of these courses as possible; by replacing some advanced mathematics courses with SE courses, this allowed us to create a program with essentially the same number of credit hours as the CS program.

2. SE program hosting department/college constraints

In addition to university/institutional constraints, program hosting department/college may have its own set of requirements (such as maximum reuse of faculty and existing courses offered, existing prerequisite structure, capstone project requirement, departmental outcomes, maximum number of credits required in a program) that need to be fulfilled.

In our case, the SE hosting department, CIS, expects the new SE program to be compatible with the existing CIS computing programs: computer science and management information systems. This is not only for the economic benefits of shared courses and faculty resources but also allows students to switch majors early in the program without much impact on required time to earn the degree (Frezza, 2003). Such a requirement leads to the mapping/compatibility analysis of existing CIS course offerings, prerequisite structure and the SE program vision. Since the SE program focuses on the *outsourced* and *MIS* categories, the CIS faculty felt the strong need of solid programming, systems, networking and computing background for SE students which in turn leads to the reuse of a great number of existing CIS courses. Specifically, SE and CS program students have almost identical courses for the first two years. However, such an arrangement also limits the number of credits available for Software Engineering specific courses that can be offered.

In addition, every CIS department graduate is expected to complete a capstone project and achieve a set of departmental outcomes. The capstone project, which integrates ethics and project management with a multi-disciplinary two semester long team project, mandates six credit hours of the senior year schedule. Although the set of departmental outcomes are not in the form of credit hours, it requires careful traceability analysis on the mappings of course outcomes to departmental outcomes. The analysis result could start the process of redefining current course content and outcomes, as well as creating new courses and the responsibility (assignment of departmental outcomes) of new courses.

3. Defined SE program outcomes

Besides the hosting departmental/college outcomes that need to be achieved, every SE program that hopes to be accredited should also fulfill the outcomes defined by its accreditation body. As discussed in the *Accreditation* subsection and *Defining the BSSE Graduate* section, every SE program needs to define its own set of program outcomes that can be related to the outcomes required by the accreditation body and specified in SE2004 (Diaz-Herrera, 2004). Both departmental outcomes and the defined SE outcomes will serve as targets for the courses designed into the BSSE curriculum.

In our case, we combined the generic CIS departmental outcomes with specific SE outcomes that then defined the BSSE program outcomes. A gap analysis was performed on the mapping of BSSE program outcomes to the eleven outcomes, the superset of ABET EAC, EAC SE and SE2004 outcomes, described in *Defining the BSSE Graduate* section. The result is shown in Table 2. The analysis goal was to make sure that every Gannon BSSE outcome maps to at least one generic SE outcome and every generic SE outcome maps to at least one Gannon BSSE outcome.

4. Outcomes to Courses: Applying ABET, SEEK and SWEBOK

The outcomes only convey what objectives should be achieved by the courses prescribed in a BSSE curriculum but not courses should be offered. One of the important goals in the design and construction of courses is to build up students' knowledge and skills through the curriculum. Both SE2004 and SWEBOK, described in *What Software Engineers Need to Know* section, define *Knowledge Areas* (KAs) and *topics* that are important to a software engineer. By identifying *Knowledge Areas* (KAs), *Knowledge Units* (KUs) and *topics* that need to be covered by courses, assigning them to courses, defining course outcomes, and mapping course outcomes to BSSE program outcomes, this not only ensures that every BSSE program outcome is covered by one or more courses, but also ensures that the important (essential) topics hand-picked by the SE program committee are covered by one or more courses. While most ABET program outcomes (A-K) and program criteria were well covered, issues regarding the minimal standards for science, mathematics, and the application of SE to some discipline needed specific inclusions in the program designs under consideration.

In our experience, SWEBOK proved to be invaluable in helping to define Knowledge Areas, subareas and topics to be covered by courses. Although SWEBOK is not designed specifically for undergraduate curriculum development and accreditation, we found it provided easy to understand, and more in-depth description of knowledge areas and topics than SE2004. In addition, it also provided useful references in building syllabi for new courses.

Since SE2004 SEEK and SWEBOK defined very similar *Knowledge Areas* (KAs), as discussed in *What Software Engineers Need to Know* section, it was not difficult for us to identify *Knowledge Areas* (KAs) that need to be

Table 2. Mapping of Gannon BSSE program outcomes to generic SE outcomes

Gannon Software Engineering (BSSE) program outcomes 2006-7	1. Apply problem solving strategies to software development	2. Interface with business and analytical professionals to solve software or systems development problems	3. Comprehend ethical decisions and their ramifications as professionals.	4. Demonstrate effective verbal, written, and listening communication skills as required for professional, group, and team interactions	5. Demonstrate the ability to continue in professional development and expansion of their professional interests	6. Maintain a comprehension of the changing technology and its ramifications	7. Realize and manage high quality sw development lifecycle processes in one or more application domains	8. Apply discrete mathematics and abstract structures to system development	9. Apply quantitative measures in the evaluation of software components and systems
Show mastery of the software engineering knowledge and skills, and professional issues necessary to begin practice as a software engineer	x	x	x	x	x		x	x	x
Ability to appropriately apply science, discrete mathematics, empirical techniques, probability and statistics and relevant topics in computer science and supporting disciplines to the development of complex software systems	x						x	x	x
Work as an individual and as part of a multi-disciplinary team to develop and deliver quality software artifacts		x		x			x		
Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems and organizations	x						x		x
Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal and economic concerns	X		X				X		
Understanding of professional and ethical responsibility			x						
Demonstrate an understanding of and apply current theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, and documentation.					x		x		x
Demonstrate an understanding and appreciation for the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment.			X	X	X				
Learn new models, techniques and technologies as they emerge and appreciate the necessity of such continuing professional development.					x	x			
Knowledge of contemporary issues					x				
Broad education necessary to understand the impact of engineering solutions in a global, economic, environmental, and societal context			x			x			

covered by courses in SE curriculum. SE2004 SEEK defined very detailed topics for each KA, more specifically topics that are designated as *essential* should be covered by an accreditable SE curriculum. In an ideal world, we should be able to assign every essential topic or even non-essential ones defined in SEEK to courses in our SE curriculum. In reality, we had to operate under institutional and departmental constraints; those are 36 credits of general education requirement, remaining compatible to existing CS and MIS programs (maximum reuse of existing CIS courses), six credits of capstone project, and 135 maximum credits for the SE program.

Our program development model started with using an existing program (in this case, Computer Science) as the basis for the formulation. Instead of assigning all essential and even non-essential SEEK topics, KUs and KAs to whatever courses that we imagined as important for SE curriculum, we started with mapping these topics, KUs and KA's to existing computer science courses, related discipline and general education course contents. After such a mapping exercise, we were able to find the topics, KUs, KAs and even BSSE outcomes not covered by existing courses. This enabled us to revisit existing course contents and outcomes, and even allowed us the opportunity to reshape or redirect the contents and outcomes of existing courses, as well as facilitate the discussion to remove courses to make room for SE courses.

The mappings created were managed in a spreadsheet, much like requirements traceability lists, and showed that the 'requirements' coverage was incomplete. So there remained the question of whether we could cover the rest of essential topics, KUs and KAs reasonably by redesigning existing course offerings or whether new courses should be designed to serve this purpose.

In our case, we were not able to reasonably cover most of the essential topics, KUs, KAs and even BSSE outcomes by redesigning existing courses. Seven new SE specific courses: *Software Engineering Seminar*, *Requirements and Project Management*, *Formal Methods in Software Development*, *Software Architecture*, *Software Testing and Quality Assurance*, *Human Interface Design and Maintenance*, and *Personal Software Process*, totaling 19 credits, were created and offered by the CIS department. The initial curriculum is presented in Figure 3, and the current (2007-8) curriculum is listed in Figure 4. Each SE student also needs to pick an application domain, consisting of nine credit hours of existing courses from various departments, to focus on. The math department also agreed to offer a new course – *Discrete Math 2* for our SE students. Even with the creation of these new courses, we still could not cover all the essential topics in SEEK due to the constraint of the maximum of 135 credits. The detailed examples of the mapping process can be found in (Frezza, 2003) and (Frezza, 2006).

Figure 3 presents the 135 credits initially proposed for the BSSE curriculum. These are organized by type, such as current Engineering and Computer and Information Science courses (51 credits), new Software Engineering courses (19 credits), Application Domain courses (9 credits), new and existing mathematics courses (15 credits), existing Science courses (8 credits), and existing liberal studies (general education) courses (36 credits).

5. New courses or reuse existing courses

In an ideal world where unlimited resources (faculty, budget, number of credits, etc.) are available, and no constraints are imposed for curriculum development, all new courses can be created specifically for an SE program. However, in reality, where multiple constraints exist and academic political issues are abundant, it's not always possible to create all the new courses a new SE program needs. In such a situation, a gap analysis on the mapping of existing course contents to SEEK topics, KUs, KAs and defined BSSE program outcomes could serve as the starting point for the discussion of redirecting current

course contents, negotiating whether and what new courses to create, and where and how to run the new courses.

The discussions about old, revised and new courses were at times heated – for example, a more 'systems' or 'engineering' flavor to the course would have introduced discrete mathematics in the context of digital design, and then followed up with formal discrete mathematics course to fill out the KA. However, the 'MIS' vision prevailed, (and the desire to parallel the Computer Science program closely), and a *Discrete Mathematics 1*, followed by a *Discrete Mathematics 2* combination was agreed upon, followed up by a *Formal Methods in Software Development* course, offered by our department. For each of these courses, appropriate KA's were assigned to the mappings, as well as the subproject of getting our colleagues in the Math Department to support the approach. Such tradeoffs were not insignificant, but were guided by the needs as expressed in the traceability spreadsheet. The spreadsheet framed the problem; our particular resolution to the program vision (information systems) and organization issues (CIS department) guided the debate.

Similar issues surfaced around addressing the 'Computer Organization' requirements within the Computing Foundations KU, which could be implemented with a 'microprocessors course' or a 'computer architecture course' – neither of which would come from our department. Here the traceability showed the need for, and coverage within the spirit and letter of the requirements – but a choice/decision needed to be made. In this case, the previous 'Discrete 1 vs. Digital Logic' decision drove the issue. Only the 'Computer Architecture' remained as a viable choice, both because of the desire to parallel the CS program, and the fact that the architecture course accepted the *Discrete Mathematics 1* course as a prerequisite – whereas the microprocessor course did not.

In our case, the seven new SE specific courses fell logically on the shoulders of the hosting department, CIS, due the focus and skills of the department and faculty. We were able to negotiate with Math department to offer the new *Discrete Math 2* course due to its strong mathematics content. For domain related courses, we decided to reuse existing courses offered by various departments to offer students more domain specific knowledge. Our current Software Engineering curriculum is presented in the Change Management section below.

Once the BSSE curriculum is developed, it is just a proposal – it serves as the guide and core for getting the agreement to launch the program, and ultimately for attracting students, staffing and running the courses that make up the program and continuing to improve the program to serve students' needs.

SELLING THE PROGRAM

During the early stages of conceiving and constructing the program, support must be obtained on several levels in order to reasonably proceed, and, ultimately, to launch a successful program.

Understanding the Student Market

One initial question which must be answered is whether there is and will be work for BSSE graduates. If this question is answered in the negative, there is little chance for program success since it will be extremely difficult to attract students to a program with bleak job prospects. According to the United States Bureau of Labor Statistics (U.S. Department of Labor, 2006a), computer software engineers in the United States held about 800,000 jobs in 2004. They are employed in a wide variety of industries, with employers ranging from startup companies to established industry leaders.

According to Bureau projections, computer software engineer will be one of the fastest-growing occupations through 2014 as businesses adopt

Figure 3. Initial Gannon BSSE curriculum (as designed 2004)

FRESHMAN

Fall		*Spring*	
Introduction to Engineering	3	Software Engineering Seminar	1
Principles of Computing	3	PC – Database	1
Calculus I	3	Using UNIX	1
History of the West and World	3	Principles of Systems	3
College Composition	3	Intro. Programming & Lab	3
Sacred Scriptures	3	Calculus 2	3
		Critical Analysis & Comp.	3
		Introduction to Philosophy	3

SOPHOMORE

Fall		*Spring*	
Problem Solving with OOP	3	Intro Visual Programming	3
Intro. Networks	3	Data Structures	3
Discrete Mathematics 1	3	Adv. Object-Oriented Programming	3
Approved Science 1 & Lab	3	Computer Architecture	3
Theology Series II	3	Discrete Mathematics 2	3
		Philosophy Series II	3

JUNIOR

Fall		*Spring*	
Software Design and Test	3	Software Engineering	3
Personal Software Process	3	Database Management Systems	3
Formal Methods in Software Development	3	Distributed Programming	3
Probability & Statistics I	3	Software Architecture	3
Theology or Phil Series III	3	Requirements & Project Management	3
Application Domain 1	3		

SENIOR

Fall		*Spring*	
CIS Professional Seminar	1	Senior Design II	3
Senior Design I	3	Application Domain 3	3
Software Testing & Quality Assurance	3	Literature Series	3
Human Interface Design & Maintenance	3	Fine Art Series	3
Application Domain 2	3	Social Science, Humanities or Business Elective	3
Approved Science 2	3		
Approved Science 2 Lab	1		

Shading Key

Engineering, Computer and Information Science	51	Mathematics	15
Software Engineering (new)	19	Science	8
Application Domain	9	Liberal Studies	36

and integrate new computer-based technologies. Jobs openings will be created both through employment growth and from the need to replace workers who retire or otherwise leave the occupation. Consulting opportunities for computer software engineers also should continue to grow. Growth in the field will come from rapidly evolving technologies as well as new software needs driven by information security concerns.

With growing internationalization of software development, some countries will see more software development contracted out abroad. However, jobs in software engineering are less prone to being sent abroad because the occupation requires innovation and intense research and development. Most companies prefer to keep this function in-house whenever practical.

Since the BSSE is a relatively new major, it is often difficult to obtain accurate statistics which isolate the major. This is true when discussing starting salaries. Some basic information is available from the National Association of Colleges and Employers web site (National Association of Colleges and Employers, 2007) (more detailed information is available to members). This press release lists the average starting salary offer to recent bachelor's graduates in Management Information Systems/Business Data Processing as $46,966. The average offer for Computer Science graduates is listed as $52,177. The press release does not distinguish between types of job, nor does it list Software Engineering as a category of degree.

The United States Bureau of Labor Statistics (U.S. Department of Labor, 2006b), lists median salary data for computer software engineers in the United States, but the data is for the profession. As discussed below, most practicing software engineers do not have a degree in Software Engineering, so the data applies to anyone working as a software engineer, not just to those with a Software Engineering degree.

Given the promising employment outlook, a follow-up question is whether students are enrolling in BSSE programs. If we build it, will they come?

As with employment trends, it is difficult to obtain accurate enrollment statistics which apply specifically to the Software Engineering major. The Digest of Education Statistics, 2005 (National Center for Education Statistics, 2005), lists 163 students receiving a Bachelor's degree in Computer Software Engineering in 2003-04, while the 2004 Digest (National Center for Education Statistics, 2004), lists 121 students receiving a Bachelor's degree in Computer Software Engineering in 2002-03. These are the most recent official statistics generally available as of this writing. While these statistics indicate solid growth in the number of bachelor's degrees granted, the number of SE graduates is still extremely small when compared with the number of software engineers needed.

It is also difficult to obtain statistics for software engineering as a career preference. For example, the Post-Secondary Planning Survey Analysis, conducted by the National Research Center for College and University Admissions (NRCCUA) Career-Choice Preferences lists "Computer Sciences," "Information Technology," and several Engineering choices, but "Software Engineering" is not included as a separate choice (NRCCUA, 2007). The SAT survey of intended majors of college bound students in 2007 includes only the broad categories of "Computer and Information Sciences and Support Services" and "Engineering" (The College Board, 2007).

Based on the above numbers, it can be concluded that most practicing software engineers do not have a degree in Software Engineering, but rather in Computer Science or some other related discipline. The question is whether a traditional Computer Science degree program best prepares a student for today's typical software engineering jobs and future career need not be addressed; there is room for and need for both majors.

With promising employment and enrollment outlooks, a final general question is whether

the market is already flooded with new BSSE programs. The brief, simple answer is "no" (or at least it was in 2002 and still is at the time of writing).

In 2003, there were 21 known programs in the United States offering some type of bachelor's degree in Software Engineering, with more being proposed (Bagert, 2003). As of January 1, 2007, there are at least 34 known BSSE programs in the U.S. 13 of which are accredited by ABET (ABET, 2006). Based on these numbers and the projected need for Software Engineers, it seems that many more programs with a Software Engineering focus are needed.

Building Additional Support

Once these initial questions are answered, support to proceed must be obtained from several groups. First, the affected faculty must support the program. If housed in a Computer Science program, one adjustment will be a shift in the focus of some upper division courses. The basics of programming, database, networking, operating systems, and computer architecture will be taught in both majors. However, Computer Science majors will then study topics such as analysis of algorithms, comparative programming languages, compilers, and formal languages. Software Engineers, on the other hand, with study topics such as software design and test, software architecture, requirements, project management, quality assurance, and human interface design.

For some faculty, this will be an issue. It will be seen as a move from a set of courses with technical, well-defined content to a set of courses with a more subjective content. If co-located with a Computer Science program, the split in focus and teaching assignment for upper division courses may fall naturally along the strengths and interests of the faculty and be seen as a positive. If not, some accommodation will need to be made for and by the faculty. This can often be accomplished by judicious distribution of the core set of courses. Regardless of initial reaction, recent downward trends in enrollment in Computer Science programs should provide motivation for faculty to support a program which will likely lead to an increase in enrollment.

Once the faculty is behind the concept of a Software Engineering program, the university administration must be convinced that launching the program is a good idea. While there are several factors involved, the overriding issue with the administration is likely to be economic: will the new program make money or lose money? For a university, the business case will boil down to whether revenue from increased enrollment will offset increased costs. The overall impact of the program proposal was the creation and staffing of seven new courses in software engineering; if offered annually, as was the plan, this defined the need for a new faculty member. In our case, we developed our proposal to feature launching the program first, and then adding the new faculty member in the second year in order to offset the economic impact.

Cost increase can vary greatly depending on factors discussed in preceding sections. If the program is housed in a department such as Computer Science, there can be much reuse of courses and faculty. As such, there will likely be little additional cost when the program is first launched. As the program moves into its third and fourth years, new upper division courses must be developed and taught. Part of this cost can be shifted by eliminating or reducing the frequency of current courses which are under-enrolled. Other costs are more than offset by the increase in enrollment.

A further advantage to the university is the visibility of offering a cutting-edge program. This will help attract both students and faculty. It will also bolster the overall image of the university. In our case, adding the SE program led to a change in the CS+SE enrollment trends – what had been on significant decline grew slightly and stabilized.

Finally, to be successful, the program must be attractive to both students and their parents. One major point, of course, is that Software Engineering is a promising career. The field provides an adequate number of jobs and good salaries. A degree in Software Engineering provides the right skill set for a student entering the job market, and degree in Software Engineering distinguishes the graduate from a graduate with a Computer Science degree. This can be highlighted by the admissions department as well as in any marketing provided by the university. One criticism of adding Software Engineering was the question of impact on Computer Science enrollment. In our case, there has been an impact on CS enrollment (and MIS for that matter). We have 'lost' students to SE from these programs, but we also found many new students who would have otherwise not applied to the university. For our (rather small) program, this has been averaging about 50% for the four years we have accepted students to the SE program. Over this period, about half of the new students who join the BS-SE would not have joined the university. For our department, this has resulted in a slight decline in CS undergraduates, but more recently it has also seen an increase in CS applicants, as well as a qualitative difference in those students who join the CS program – they are increasingly joining because they are genuinely interested in CS topics and approaches.

Finally, the entry in the university catalog for the new program becomes more important than for more established programs. In addition to just being a listing of required courses and their content, the catalog entries server as a marketing tool. A good entry will highlight the potential of the career path, the promise of the department to the students to deliver the appropriate courses and material, and the commitment of the university to support the new program.

LAUNCHING THE PROGRAM

Based on the plan initially proposed to and then approved by the university (Frezza, 2003), Gannon was able to launch the SE major with little impact on courses or faculty for the first two years. Based on the assessment of Gannon's strengths and those of the various departments as well as a review of requirements imposed by the policies of both the university and college, the courses taken by SE majors in the first two years were already offered within the university. These courses consisted of Liberal Arts core courses taken by all Gannon students, CIS core courses already offered and taken by the CS and MIS majors at the university, and introductory science courses already offered to several majors. The only exception to this is the *Discrete Math 2* course taken second semester of sophomore year. The Math department was willing to develop and offer this course in the necessary time frame. By design, the first two years of the program had little immediate impact on course delivery or teaching load – only one new one-credit course was needed. Our internal goal of trying to reuse as many of the existing Computer Science curriculum courses and sequences as possible had the significant benefit of our being able to launch the program prior to searching for new faculty.

Adding New Faculty

Adding a new faculty member to the department proved to be significantly easy for our situation, as one of the proponents of the program (who was CSDP qualified), by mutual agreement, essentially transferred from the Electrical and Computer Engineering department. Finding a second qualified faculty member was not as easy. Finding a potential faculty member with a good academic

background, worthwhile industry experience, an appropriate commitment to teaching and scholarship, who fits in the university/department culture and has not already taken on a higher-paying job in industry is a tall order.

While our experience has been somewhat limited, some of the problem areas have included recruitment of junior faculty; getting the right match for credentials, experience and fit have proven to be difficult. The typical Computer Science Ph.D. may not have any interest in 'core' software engineering topics; whereas Ph.D.'s from Engineering or Information Science disciplines may not have any significant software development background. In any case, many typical Ph.D.s do not have either experience or education in SE 'special topics' areas such as project management, software maintenance or even requirements.

Additional issues abound when searching for seasoned faculty members – while they often have more project and teaching experience, they also have the potential for more (and more complex) issues in areas such as benefits and tenure. Such issues are often institutional, and flexibility may not be available.

Course Pilots

This particular facet of our program design did not eliminate the need for additional staffing, however. Based on our assessments, we would need a new faculty member in the second year of the program to teach the first round of the new SE courses, and if enrollment took off, in year three or four we would need another faculty member to help with the additional sections that would be needed in the introductory sequence. These points were very important to painting the financial picture to the university about the distribution of costs and risks in launching the program. Financially, the big impact would be in years two and four; academically, the real impact would begin in the third year of the program; a change-management plan was needed.

The plan was to hire a new (additional) SE-qualified faculty member in year two, and additionally to offer all of the new junior-level SE courses in advance. This latter plan was part of a (pedagogical) risk-management strategy so that the new faculty member and other department faculty members could pilot the new SE courses prior to the first wave of SE students entering the courses. The expanded department faculty would then offer each of the new SE courses one year in advance of the first wave of SE students – with three semesters containing significant piloting of courses.

For the fall (piloting semester five) of their junior year, SE majors were scheduled to take three upper division SE-specific courses: *Software Design and Test*, *Personal Software Process*, and *Formal Methods in Software Development*. Of these, only one was new, as *Software Design and Test* was already a course offered and required of our CS majors and *Personal Software Process* was offered to our 1st-year graduate students. Hence neither of these courses needed significant modifications for the undergraduate BSSE population. The *Formal Methods* course was a different story – it was new to our faculty and needed to be developed and offered. Finding educational resources and faculty development seminars to support this course proved to be difficult. Developing this course offering required identifying and making decisions on course topics, approaches, tools, methodologies and textbooks that would work for our students.

During the next spring (piloting semester six), the impact intensified. The SE majors were scheduled to take two more new courses in the following year, so *Requirements and Project Management*, and *Software Architecture*, which were developed and offered in their pilot forms, knowing the regular group of majors would register in the following year. This additional workload was covered by the new faculty member.

The third year of the program (pilot for Semester seven) required the development and offering

of two new undergraduate courses, both offered in the first semester: *Software Testing and Quality Assurance*, and *Human Interface Design and Maintenance*. These courses, along with the first regular offerings of the three fifth-semester SE-specific courses accounted for the additional time provided by the faculty hired at the beginning of the second year of the program.

In addition to the courses and faculty needed, two other factors needed to be considered. First, the university Admissions department needed to be involved in the process. The program needed to be advertised as much as possible, and admissions needed to understand the requirements for admission to the program. They also needed to place the incoming SE students in the proper courses for the freshman year.

Finally, the issue of transfer students, both internal and external, needed to be addressed. Since the program was phased-in over four years, for the first two years, we would be unable to accept upper division transfers (unless they were willing to stay an extra year) since the upper division courses were not available during those first two years. We did, in fact, accept one sophomore transfer the first year of the program and he was able to graduate with just one additional semester.

CHANGE MANAGEMENT

One important issue to address early in the program development is to plan for change. This begins with outcomes and measurement. We were fortunate that many of the processes that we required in this area were already largely in place. The CIS department already had outcomes defined for the CS and MIS majors and some measurement tools in place.

Further, the Electrical Engineering and Mechanical Engineering departments were ABET accredited. Concurrently with the launching of the SE major, the CIS department was preparing for ABET accreditation for the two existing majors, and the EE and ME departments were preparing to renew their accreditation. To aid in this process, an online course evaluation tool was prepared to gather information specifically required by the ABET process. (A university-wide course evaluation instrument had been in use for all courses for several years, but it did not gather all information required by ABET.) This tool was used from the beginning of the SE program.

Even though we are only about to begin our fourth year of the program, and the first time the senior level courses will be offered, we have already made changes to the program. First, we realized that we had a hole in coverage in the operating systems area. To address this issue within faculty time and budget constraints, we added the full Operating Systems course to the list of required courses. To make room, we dropped an Introduction to Engineering course which had only partial content relevant to our needs. We moved the relevant content into pieces of existing courses where they fit the best.

One of the core Liberal Studies requirements is a basic business course. Many of our majors chose either Microeconomics or Macroeconomics. After two years of the program, we saw an opportunity to provide a business course more directly applicable to the SE majors. As such, we co-developed a course (*Project Economics*) with the business school which provides basic economics theory as well as the application of the theories in a project setting. This course is now the designated business course for SE majors.

Our current Software Engineering Curriculum (2007-8) is listed in Figure 4. Diagonal shading indicates reused Computer and Information Science courses. The darker grey shading indicates new courses developed for the Software Engineering major by the Computer and Information Science department. One of the seven originally proposed new Software Engineering courses (See Figure 3), the *Software Engineering Seminar*, was offered two times and it was determined that the course was not serving the needs of our software

Figure 4. Current software engineering curriculum (2007-8)

FRESHMAN

Fall		*Spring*	
Principles of Computing	3	Using UNIX	1
PC – Database	1	Intro. Programming & Lab	3
Calculus I	3	Calculus 2	3
History of the West and World	3	Critical Analysis & Comp.	3
College Composition	3	Introduction to Philosophy	3
Sacred Scriptures	3	Approved Science 1 & Lab	4

SOPHOMORE

Fall		*Spring*	
Problem Solving with OOP	3	Data Structures	3
Intro. Networks	3	Adv. Object-Oriented Programming	3
Discrete Mathematics 1	3	Computer Architecture	3
Approved Science 2 & Lab	4	Database Management Systems	3
Project Economics	3	Discrete Mathematics 2	3
		Philosophy Series II	3

JUNIOR

Fall		*Spring*	
Software Design and Test	3	Software Engineering	3
Operating Systems	3	Visual Database Programming	3
Personal Software Process	3	Software Architecture	3
Formal Methods in Software Development	3	Requirements & Project Management	3
Probability & Statistics I	3	Application Domain 1	3
Theology Series II	3	Theology or Phil Series III	3

SENIOR

Fall		*Spring*	
CIS Professional Seminar	1	Senior Design II	3
Senior Design I	3	Distributed Programming	3
Software Testing & Quality Assurance	3	Application Domain 3	3
Human Interface Design & Maintenance	3	Literature Series	3
Application Domain 2	3	Fine Art Series	3
Social Science, Humanities or Business Elective	3		

Shading Key

Computer and Information Science	48	Mathematics	15
Software Engineering (new)	18	Science	8
Application Domain	9	Liberal Studies	36

engineering students. Therefore, it was removed from the curriculum. As noted in a previous section, *Discrete Math 2* was developed by the Math department for our software engineering program. The remaining courses required for the software engineering major, including the application domain courses, are coded in Figure 4 as indicated by the shading key. These courses were already

offered by various departments within the university, with the exception of *Project Economics* discussed above.

CONCLUSION

This article does not address the relative merits of the education provided by a traditional Computer Science program compared with a Software Engineering program. It does seem that there is a need to provide the type of Software Engineering curriculum discussed in the article and that the curriculum can be provided along with, rather than instead of, a more traditional Computer Science curriculum. Programs have been and can be introduced at institutions with diverse size, diverse overall focus, diverse program style, and diverse strengths.

This article discusses many items to consider in the process of planning and launching a new BSSE program. Further, obtaining program accreditation is highly desirable, in some cases necessary. Understanding the steps required by the appropriate accrediting body to obtain accreditation is mandatory at some point in the program's lifecycle. Understanding the steps very early and accounting for them during program planning will help smooth the journey to accreditation.

REFERENCES

ABET Engineering Accreditation Commission (EAC) (2005). Criteria for Accrediting Engineering Programs, Effective for Evaluations during the 2006-7 Accreditation Cycle. Baltimore, MD. Retrieved May 28, 2007, from http://www.abet.org/forms.shtml.

ABET (2006). List of Accredited Programs in Software Engineering, October 1, 2006. Retrieved May 28, 2007, from http://www.abet.org/ABETWebsite.asp#area

ABET (2007). Home Page, Retrieved May 28, 2007, from http://www.abet.org/index.shtml

Abran, A., & Moore, J. (Eds.). (2004). Guide *to the Software Engineering Body of Knowledge, 2004 Version*, IEEE Computer Society Press. Available at http://www.swebok.org.

Bagert, D., & Ardis, M. (2003, November). *Software Engineering Baccalaureate Programs In The United States: An Overview.* Proceedings of the Frontiers in Education Conference (FIE'03). Boulder, CO.

Bourque, P., Dupuis, R., Abran, A., Moore, J., & Tripp, L. (2000, August). *Developing Consensus on the Software Engineering Body of Knowledge.* Proceedings of the 2000 World Computer Congress, Beijing, China. Available at *http*://www.gelog.etsmtl.ca/publications/pdf/535.pdf

Canadian Council of Professional Engineers, Canadian Engineering Accreditation Board (2006). CEAB Accreditation Criteria and Procedures. Ottawa, Ontario, Canada. Retrieved October 26, 2007, from http://www.engineerscanada.ca/e/files/report_ceab.pdf

Diaz-Herrera, J. L., Hilburn, T., Hislop, G., Lutz, M., MacNeil, P.E., & McCracken, M. (2001, October). *Software Engineering Education Should Be Presented as A: Science, B: Engineering, C. Technology, D. None of the above, E. All of the above, Other.* Proceedings of the Frontiers in Education Conference (FIE'01), Reno, NV.

Diaz-Herrera, J. L., & Hilburn, T. (Eds.). (2004). *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering A Volume of the Computing Curricula Series.* Available at http://sites.computer.org/ccse

Engineers Australia (2007). Australian Professional Engineering Programs Accredited by Engineers Australia. Last updated 6 September 2007. Retrieved November 11, 2007 from http://www.

engineersaustralia.org.au/education/program-accreditation/accredited-programs/accredited-programs_home.cfm

Frezza, S., Sasi, S., & Seol, J. (2003, November). *Report from the Trenches: Applying the SEEK to BSSE Program Development.* Proceedings of the Frontiers in Education Conference (FIE'03). Boulder, CO.

Frezza, S. T., Tang, M-H., & Brinkman, B. J. (2006). Creating an Accreditable Software Engineering Bachelor's Program. *IEEE Software*, 23(6), 27-35.

Henderson, P., Linos, P., & Tinsley, E. (2003). *Crafting an Undergraduate Software Engineering Program in a Liberal Arts Environment.* Unpublished extended abstract, Butler University, Indianapolis, IN.

IEEE Computer Society (2001). *The Certified Software Development Professional Program*, Available at http://www.computer.org/portal/pages/ieeecs/education/certification.

Jones, C. (2003). Variations in Software Development Practices. *IEEE Software*, 20(6), 22-27.

Kelley, R. E. (1999). How to be a Star Engineer, *IEEE Spectrum*. 36(10), 51-58.

McConnell, S., & Tripp, L. (1999). Professional Software Engineering: Fact or Fiction? *IEEE Software*, 16(6), 13-18.

National Association of Colleges and Employers (2007). *Higher Starting Salary Offers Reflect Positive Trend in Job Market for New College Graduates.* Press Release. Retrieved May 28, 2007, from http://www.naceweb.org/press/display.asp?year=2007&prid=256

National Center for Education Statistics (2004). Institute of Education Sciences, U. S. Department of Education. *Digest of Education Statistics, 2004.* Retrieved May 28, 2007, from http://nces.ed.gov/programs/digest/

National Center for Education Statistics (2005). Institute of Education Sciences, U. S. Department of Education. Digest of Education Statistics, 2005. Retrieved May 28, 2007, from http://nces.ed.gov/programs/digest/

National Research Center for College and University Admissions (2007). Post-Secondary Planning Survey Analysis, 2007-2008 Edition. Retrieved November 15, 2007, from http://www.nrccua.org/downloads/reports/survey_analysis.pdf

Naur, P. & Randell, B. (Eds.) (1969). *Software engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7–11 October 1968, Brussels, Scientific Affairs Division, NATO.

Sobel, A. E. K., Bagert, D. J., Frezza, S. T., & Pavlov, V. L. (2007, October). Panel - *Assessing The Impact of the SE2004 Curriculum Guidelines*, presented at the Frontiers in Education Conference (FIE'07), Milwaukee, WI.

The College Board (2007). 2007 College Bound Seniors, Total Group Profile Report. Retrieved November 15, 2007, from http://www.collegeboard.com/prod_downloads/about/news_info/cbsenior/yr2007/national-report.pdf

U. S. Department of Labor, Bureau of Labor Statistics (2006a). *Occupational Outlook Handbook (OOH)*, 2006-07 Edition. Retrieved May 28, 2007, from http://www.bls.gov/oco/

U. S. Department of Labor, Bureau of Labor Statistics (2006b). *Occupational Employment and Wages.* May 2006. Retrieved May 28, 2007, from http://www.bls.gov/oes/current/oes151032.htm

Wankat, P. & Oreovicz, F. (1993). *Teaching Engineering*, Upper Saddle River, NJ: McGraw Hill.

This work was previously published in Software Engineering: Effective Teaching and Learning Approaches and Practices, edited by H. Ellis, S. Demurjian, & J. Naveda, pp. 298-325, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 5.12
Facilitating eLearning with Social Software:
Attitudes and Usage from the Student's Point of View

Reinhard Bernsteiner
University for Health Sciences, Medical Informatics and Technology, Austria

Herwig Ostermann
University for Health Sciences, Medical Informatics and Technology, Austria

Roland Staudinger
University for Health Sciences, Medical Informatics and Technology, Austria

ABSTRACT

This article explores how social software tools can offer support for innovative learning methods and instructional design in general, and those related to self-organized learning in an academic context in particular. In the first section, the theoretical basis for the integration of wikis, discussion forums, and Weblogs in the context of learning are discussed. The second part presents the results of an empirical survey conducted by the authors and explores the usage of typical social software tools that support learning from a student's perspective. The article concludes that social software tools have the potential to be a fitting technology in a teaching and learning environment.

INTRODUCTION

One major task of higher education is to train students for the requirements of their future work by applying and adapting their knowledge to specific workplace-related requirements and settings. Due to the ongoing pressure on enterprises to cut costs, the periods of vocational adjustment in a company will become shorter and shorter.

On the one hand, the rising pressure of innovation and fast-paced development in the economy results in increased demand for continuous employee training. On the other, growing global competition forces enterprises to use available resources very economically so that employee training is considered to be necessary and desired even though it is conducted under considerable time and cost pressure (Köllinger, 2002).

According to these goals, the settings of the education must be changed adequately: "While most of higher education still ascribes to traditional models of instruction and learning, the workplace is characterized by rapid changes and emergent demands that require individuals to learn and adapt in situ and on the job without the guidance of educational authorities" (Sharma & Fiedler, 2004, p. 543).

In the field of higher education, it has become an important goal to develop "digital literacy" and educate learners as competent users and participants in a knowledge-based society (Kerres, 2007), but it can be assumed that there is a new generation of students, the "digital natives," who are accustomed to digital and Internet technology (Prensky, 2001a, 2001b).

Oblinger and Oblinger (2005) characterize next-generation students (called "n-gen," for Net generation) as digitally literate, highly Internet savvy, connected via networked media, used to immediate responses, preferring experiential learning, highly social, preferring to work in teams, craving interactivity in image-rich environments, and having a preference for structure rather than ambiguity.

According to a study conducted by Lenhart and Madden (2005), half of all teens in the USA may be considered "content creators" by using applications that provide easy-to-use templates to create personal Web spaces.

Classical face-to-face learning is seen as rigid and synchronous, and it promotes one-way (teacher-to-student) communication. Thus, it is not surprising that more and more students are opting for Web-based education as a more flexible and asynchronous mode (Aggarwal & Legon, 2006).

The higher education system should provide answers to this new generation of students who enter the system with different backgrounds and skills. They are highly influenced by social networking experiences and are able to create and publish on the Internet (Resnick, 2002).

Educators and teachers therefore have to consider the implications of these developments for the future design of their courses and lectures.

In 2002, a new term, "social software," entered the stage to refer to a new generation of Internet applications. One focus of this new generation is the collaboration of people in sharing information in new ways such as through social networking sites, wikis, communication tools, and folksonomies (Richter & Koch, 2007).

Wikis, Weblogs, and discussion forums will play a central role in the new context, so the areas of application and possibilities will enlarge enormously. It can be assumed that this will also have considerable influence on learning and the usage of these instruments as learning tools.

This article presents the results of an empirical survey in order to highlight the benefits of the above-mentioned Web-based social software tools from the student's point of view; 268 first-semester students, all in the first term of their studies at Austrian universities from different study programs, took part in this survey. The students were asked to use one or more of these tools as a learning tool. Participation in this survey was voluntary.

The presentation of the results of this survey is divided into three parts: first, the use of the tools by the students (before they started their studies); second, the experiences the students had made with the tools during the study; and third, the potential future usage.

The article concludes with a discussion of the results of this survey in contrast with other empirical studies already published. Also, the limitations of this survey and ideas for further research are pointed out.

THEORETICAL FRAMEWORK

This part refers to the necessary theoretical background required for the following empirical study, especially the areas of social software and learning.

Social Software

The term *social software* emerged and came into use in 2002 and is generally attributed to Clay Shirky (2003). Shirky, a writer and teacher on the social implications of Internet technology, defines social software simply as "software that supports group interaction."

Another definition of social software can be found in Coates (2005), who refers to social software as "software that supports, extends, or derives added value from human social behaviour."

Users are no longer mere readers, audiences, or consumers. They have the ability to become active producers of content. Users can act in user and producer positions and they can rapidly change the position.

Nowadays the term *social software* is closely related to "Web 2.0." The term *Web 2.0* was introduced by Tim O'Reilly (2005), who suggested the following definition:

Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform: delivering software as a continually updated service that gets better the more people use it, consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others, creating network effects through an "architecture of participation," and going beyond the page metaphor of Web 1.0 to deliver rich user experiences.

Web 2.0 technologies such as blogs, wikis, podcasts, and RSS feeds or discussion forums have been dubbed social software because they are perceived as being especially connected and allow users to develop Web content collaboratively and publicly (Alexander, 2006).

Until now, the Internet (Web 1.0) has one big disadvantage: It is easy to get information in it, but it is quite complicated and inconvenient to act as an author and take part in the development of content. Web 2.0 should enable all Internet users to actively take part in the further development of the Internet. Everyone should be able to contribute easily. The focus of Web 2.0 is on the behaviour of the user. It should empower people to communicate, collaborate, contribute, and participate.

This growing phenomenon is very interesting and ought to be examined carefully in order to understand how the Web is evolving and how this continuously regenerative cycle of performance and technological innovation empowers "learning by sharing" (Thijssen & Vernooij, 2002).

Based on the key principle of the architecture of participation, social software can be seen as part of Web 2.0. Wikis, Weblogs, and discussion forums are tools that are seen as social software applications and were selected for further research and the empirical study presented below.

Related Empirical Research

Institutions in the field of higher education have made efforts to introduce various IT-supported learning tools in the daily routine of students and lecturers (Aggarwal & Legon, 2006; Dooley & Wickersham, 2007; Duffy & Bruns, 2006; Evans & Sadler-Smith, 2006; McGill, Nicol, Littlejohn, Grierson, Juster, & Ion, 2005).

Published results of the usage of Weblogs in the Prolearn project (http://www.prolearn-project.org) have shown that a large majority of respondents considers personalization and adaptation of the learning environment as important and crucial factors. Learning should be individualized to become more effective and efficient. Personalization is a key element of the learning process, and specific problems need specific solutions as students differ greatly in their backgrounds and capabilities.

Learning materials are typically too general in order to cover a very wide range of purposes and personal learning needs. Compared to classical learning, personalization can be the most important added value that e-learning can offer. With it, education can be optimised and adjusted to various working conditions and needs because students have different goals, interests, motivation levels, learning skills, and endurance (Klamma et. al., 2006).

Chao (2007) explored the potential uses of wikis in the field of software engineering (38 participants), especially for software project team collaboration and communication. Overall, 25 students agreed and 1 student disagreed (2 were neutral) that the wiki is a good tool for project collaboration. Concerning the applications of wikis, more than 23 students found that a wiki is a good tool for maintaining a group diary, managing user stories (project requirements), and project tracking and reporting. While a majority of students found that a wiki is a good tool for updating a project plan, managing acceptance tests, tracking defects, and developing user documents, there were also a significant number of students who disagreed.

First results using wikis for collaborative writing (about 40 participants) also reported similar results. In this study, students used wikis to write articles partly together with the lecturer.

After early problems with using the software and writing contributions in the wiki, students were able to write articles by themselves or in teams. The motivation among students was on different levels, so the lecturer had to increase it during lessons. Other students, however, were highly motivated and were creating the content and adding them to the wikis (Bendel, 2007).

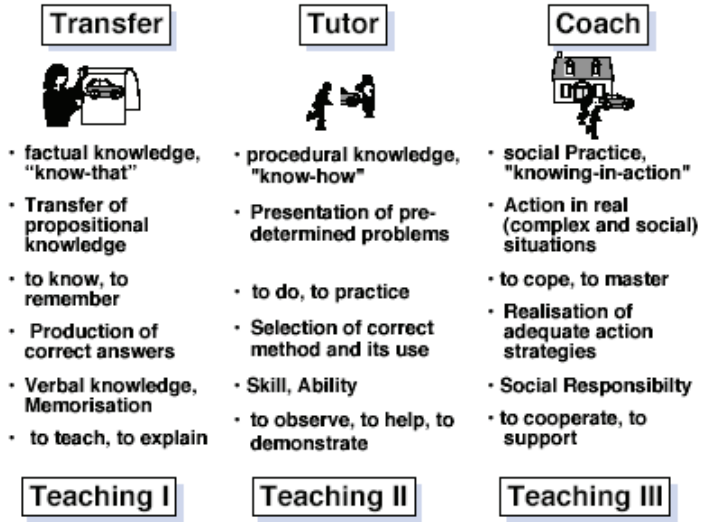
Constructivism and Learning: Presentation of the Learning Model

A constructivist point of view focuses on the learning process by looking at the construction of knowledge by an individual. As a consequence, there is a recommendation to align learning environments, especially in the academic context, and associated complex learning objectives with constructivist learning principles (Du & Wagner, 2005; Jonassen, Mayes, & McAleese, 1993; Tangney, FitzGibbon, Savage, Mehan, & Holmes, 2001). Learning is not seen as the transmission of content and knowledge to a passive learner. Constructivism views learning as an active and constructive process that is based on the current understanding of the learner. Learning is embedded in a social context and a certain situation (Schulmeister, 2005).

The constructivist approach shifts learning from instruction and design centered to learner-centered learning and teaching. The role of the educator changes from directing the learner toward supporting and coaching the learner.

Baumgartner (2004) has suggested three different prototypical modes of learning and teaching. These three different modes of learning and teaching can be neutral or specific so they can be applied across all subject domains. Therefore, each teaching model can be used to teach, for example, sociology subjects as well as to teach technical sciences. Learning can be portrayed as an iterative process that can subsequently be subdivided into different phases, which are summarized in Figure 1.

Figure 1. Prototypical modes of learning and teaching (Baumgartner, 2004)



In particular, these three different prototypical modes for learning encompass the following.

Learning and Teaching I: Transferring Knowledge

At the starting point, the learner needs to be provided with abstract knowledge to lay the theoretical foundations and to understand relevant signposts, road markings, and orientation points. This kind of factual knowledge is static and has little value by itself in real and complex situations. It merely serves as a shortcut to prevent pitfalls and to help to organize the student's learning experiences.

The knowledge of the student is based on knowledge possessed by the teacher. Students have to learn what teachers ask them to learn. The teacher has the responsibility to make the knowledge transfer as easy as possible.

Learning and Teaching II: Acquiring, Compiling, and Gathering Knowledge

In this section of the individual learning career, the student actually applies the abstract knowledge and gathers his or her own experiences. In order to limit the action and reflection possibilities, the learner interacts within a somewhat restricted, artificial environment, which is reduced in complexity and easy to control by the teacher. To provide feedback, the learning environment is designed to include relevant devices where students can deposit their interim products and teachers can inspect them.

The emphasis in this model lies on the learning process of the student. Teachers try to help the students overcome wrong assumptions and wrong learning attitudes, and assist in the reflection process of the subject domain.

Learning and Teaching III: Developing, Inventing, and onstructing Knowledge

Teacher and learner work together to master problems. This model includes problem generation and/or invention. The environment is constructed in such a way that it represents, at least in certain aspects, reality or reality in a constrained form. This model includes two-way communication on equal terms, using either linguistic representations or other adequate kinds of language.

Teaching III has strong links to constructivism. From a constructivist point of view, learning is considered as an active process in which people construct their knowledge by relating it to their previous experiences in complex and real situations in life. In their practical lives, people are confronted with unique, unpredictable situations whose inherent problems are not readily observable (Baumgartner, 2004).

Students should be enabled to invent new things, and produce or generate new knowledge. Consequently, learning and teaching at universities in most cases can be assigned to the requirements presented in Learning and Teaching II and III. In order to achieve this goal, a special learning environment must be provided.

Consequences for IT-Supported Learning and Teaching

Computer software can be used for all three models, ranging from programmed instruction (Learning/Teaching I) to problem-solving software (Learning/Teaching II), to complex simulations and/or so-called micro worlds (Learning/Teaching III). It is said that the inherent nature of the Internet brings the real world into the classrooms, and with its hyperlink structure it clearly advocates the model of Teaching III (Baumgartner, 2004).

The use of the Internet, especially through its social software, gains importance because it can contribute to exceed the limits of classical teaching models. By adapting learning and teaching models to the new technical possibilities, the roles of learner and teacher are becoming more indistinct because the learner can take a central part in the design and arrangement of the learning process (Kerres, 2006).

Systems that support learners with respect to the Learning Model III are called personal learning environments (PLEs). PLEs are mostly Web-based applications and are based on learning management systems (LMSs; Seufert, 2007).

PLEs are personal and open learning environments, and they are suitable for cross-linking content and people. Learners can use PLEs to manage individual learning progress. They are ideally available for lifelong learning and are supported by the following processes.

- setting up individual learning goals
- planning and controlling one's own learning concerning the content as well as the learning process
- combining formal and informal learning activities
- communicating with peers during the learning process
- establishing social networks or communities of practice
- using Web-based services, for example, syndication
- verifying the learning process with respect to the learning goals

Unlike an LMS, which is usually related to one special institution or to one special course, a PLE is focused on the individual learner. A PLE should combine a broad mixture of different resources and subsystems in a "personally-managed space" (Attwell, 2006).

In the previous decade, learning management systems were developed that moved toward enterprise-level applications, "but the wealth of new, user-friendly, tools in the Web 2.0 environment suggests that the all-in-one monolithic e-learning systems may be entering a phase of obsolescence by the ongoing development of the web" (Craig, 2007).

Social software applications have the potential to cope with these requirements (Brahm, 2007).

DESCRIPTION AND CLASSIFICATION OF SOCIAL SOFTWARE TOOLS

In the following section, three social software tools—Weblogs, discussion forums, and wikis—are described in more detail and the tools are compared. Students were able to select these tools during the empirical study.

Weblog

A Weblog, a compound of *Web* and *logbook*, usually just called "blog," is a Web site that contains new articles or contributions in a primarily chronological order, listing the latest entry on top.

Primarily, a Weblog is a discussion-oriented instrument especially emphasizing two functions: RSS feeds and trackback. RSS feeds, also called RSS files, can be read and processed for further use by other programs. The most common programs are RSS readers or RSS aggregators that check RSS-enabled Web sites on behalf of the user to read or display any updated contribution that can be found. The user can subscribe to several RSS feeds. Thus, the information of different Web sites can be retrieved and combined. Preferably, news or other Weblogs are subscribed to.

Trackback is a service function that notifies the author of an entry in a Weblog if a reference to this contribution has been made in another Weblog. By this mechanism, a blogger (person who writes contributions in a Weblog) is immediately informed of any reactions to his or her contribution on other Weblogs (Hammond, Hannay, & Lund, 2004).

Forum

A discussion forum or Web forum is a service function providing discussion possibilities on the Internet. Usually, Web forums are designed for the discussion of special topics. The forum is furthermore subdivided into subforums or subtopics. Contributions to the discussion can be made and other people may read and/or respond to them. Several contributions to a single topic are called a thread.

The application areas of the two instruments, Weblogs and forums, are quite similar. The most essential differences between Weblogs and discussion forums can be described as follows:

- A forum is usually located on one platform while many bloggers develop their own, individual environments. They connect their Weblogs via RSS feeds and trackback functions.
- Through the integration of RSS files and trackback functions, a discussion process can be initiated and continued, crossing the boundaries of the bloggers' own Weblogs without authors having to observe other Weblogs.
- Weblogs tend to be more people centered whereas forums are more topic focused. Through the use of Weblogs, learner-specific learning environments can be constructed without interfering with the learning environments of others (Baumgartner, 2004).

Wiki

A WikiWikiWeb, shortly called wiki, is a hypertext system for storing and processing information. Every single site of this collection of linked Web pages can be viewed through a Web browser. Furthermore, every site can also be edited by any person. The separation between authors and readers who write their own text, and change and delete it is obsolete as also third parties can carry out these functions (Augar, Raitman, & Zhou, 2004).

Learning Activities Supported by Social Software

The integration of different social software tools offers support in the following learning activities:

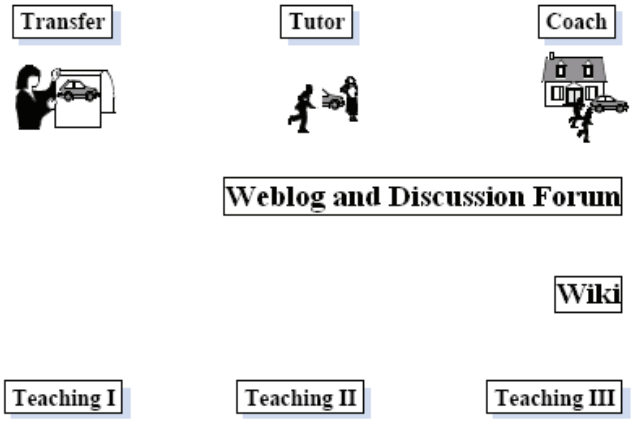
- Learning from different perspectives: The integration supports the exchange of ideas as well as finding like-minded people. Furthermore, social software tools simplify the process of establishing connections between people of the same or similar interests. Simultaneously, its open and expandable philosophy supports going beyond the thinking in groups (of a common interest) by supporting diversity and bringing together different perspectives and backgrounds (Efimova & Fiedler, 2004; Schulmeister, 2004).
- Synergies of self-organized and joint learning: Social Software tools provide a personal learning area for their authors. However, this does not force a general learning flow or learning style. Learners are not alone and can profit from the feedback of a community in order to examine and enhance the development of their own ideas (Böttger & Röll, 2004; Efimova & Fiedler, 2004; Fiedler, 2004).
- Digital apprenticeship: Through reading other wikis, forums, or Weblogs regularly, beginners are enabled to learn from experts. At the same time, they can actively participate in discussions beyond geographic or thematic borders (Efimova & Fiedler, 2004; Fiedler, 2004).
- Support for the development of the ability to learn (learning to learn): Through the publication of one's own thoughts and reflections, content is made available for assessment as well as for further development, thereby improving self-observation and self-reflection skills. The knowledge change of the learner will be improved (Baumgartner, 2005).
- Support for reflexive writing: The simple but efficient and rather robust encoding standard usually used in social software allows for the explicit modeling of content flows, feedback loops, and monitoring procedures of various kinds, thus supporting an ongoing reiterative process of explication and reflection (Fiedler, 2004).

Integration of Social Software Tools and the Learning and Teaching Modes

Baumgartner (2004) has integrated different types of content management systems in relation to the most suitable learning and teaching mode. He clearly states that the boundaries are overlapping and that every tool, in one way or another, could be used for every teaching model. Figure 2 presents the integration of the social software tools and the learning and teaching modes.

Weblogs and forums can be defined as discussion-oriented tools because the discourse and exchange of ideas related to a certain topic is the preeminent aim. Weblogs offer the possibility to support all three phases of the learning process. However, the main focus can be assigned to Modes II and III.

Figure 2. Prototypical modes and social software tool



Based on the multitude of interaction possibilities, wikis can be attached to Teaching III (Baumgartner, 2004). Additional functions were added to Weblog tools that go beyond the scope of the central use of Weblogs; for example, longer articles can also be stored. Through the creation of directories, a structured collection of links can be implemented.

Through the additional linking of Weblogs, wikis, and forums, there is the possibility to develop a personal knowledge collection (Kantel, 2003).

EMPIRICAL SURVEY

The purpose of this survey was to determine if the integration of Web-based social software tools (wikis, discussion forums, and Weblogs) are suitable to foster learning from the student's point of view.

Aim of the Survey and Methodology

Scrutinizing the possibilities and constraints of social software tools (wikis, discussion forums, and Weblogs) as personal learning environments, students at Austrian universities were asked to use one or more of the offered tools for their research, homework, and documentation purposes. In most cases, the collaboration of students was required to perform the assigned tasks.

The students were asked to use the tools for one course only during the winter term of 2006. Furthermore, there was no obligation for the students to use a tool at all; they were just encouraged to do so. Students were also offered the possibility to use two or three tools; the selection was up to the students.

The courses were organized as blended-learning courses so they included on-campus lessons and off-campus work in which the students could work face-to-face or using the social software tools.

More than 90% of the students attending the courses took part in this survey. In order to give the participants an impression of the functionality and usage of the tools, short presentations of the tools were made by an instructor before the students made their choice.

At the end of the testing phase—after 4 weeks of using the tools—selected students reported their experiences with the tools used. Students who had decided not to use the tools in the first place got an impression about the usage, advantages, and disadvantages of the tools from their fellow students. Following these short presentations, a

questionnaire was completed that provided the basic findings for further inspection and research.

A total of 268 first-semester students of different Austrian universities in five selected courses took part in this survey. The majority of the participants were between 18 and 20 years old. The portion of female students was about 17%.

According to a survey conducted by Seybert (2007) concerning gender differences in computer and Internet usage by young people (aged between 16 and 24), there is no gap between men and women in Austria. The proportion of women and men (in the relevant age class) that use a computer (almost) once a day is 72% the same. A study by Otto, Kutscher, Klein, and Iske (2005) indicates that there is a positive correlation between a formal educational background and the usage of the Internet in Germany: "Beside socio-cultural resources like family background, peer structures and social support in general, the formal educational background turns out to be the main factor for explaining differences in internet usage" (p. 219). As a consequence, for the analysis of the results of this survey, no distinction between male and female students was made.

Table 1 presents the distribution of the participants concerning the degree programs the students are attending.

For the further analysis of the results, no distinction according to degree programs will be made.

The questionnaire asked each participant questions about her or his subjective impression of the application of the tools. It included 5-point Likert scales for rating constructs such as eligibility, perceived quality, or enjoyment.

The study was conducted to find answers about the:

- usage of social software before the study started,
- selection of the offered tools,
- perceived quality of the contributions and the support for learning,
- applicability of the instruments to support communication and community building,
- correlation of the usage for private and educational purposes of the tools,
- fun factor in using the instruments, and
- potential future usage.

Table 1. Distribution of students regarding degree program

	Distribution
Management & Law	17%
Management & IT	31%
Management & Industrial Engineering	22%
Mechanical Engineering, Electronics	30%

The results of the study are presented in three parts:

- Part 1: Analysis of the usage of wikis, discussion forums, and Weblogs of the students before the study was started
- Part 2: Experiences made with the tools during the study
- Part 3: Potential future usage of the tools

Part 1: Tool Selection and Prestudy Usage

Due to the fact that the students could select the tools on their own, the Table 2 shows the results of this selection process.

According to Table 2, the combination of wikis and discussion forums is the most selected combination of tools (42.9%), followed by wikis only (23.1%) and discussion forums only (22.4%). In the end, only five students (1.9%) did not take part in the study; they did not select a tool, although they first had the intention to do so. Only one student used Weblogs only. Generally, Weblogs were not used very intensively by the participants.

Table 2. Tools selected by the students

	Percent	Number
Only one tool selected		
Wikis only	23.1%	62
Discussion forums only	22.4%	60
Weblogs only	0.4%	1
More than one tool selected		
Wikis and discussion forums	42.9%	115
Wikis and Weblogs	1.9%	5
Discussion forums and Weblogs	0.7%	2
Wikis, discussion forums, and Weblogs	6.7%	18
No tool selected		
No tool selected	1.9%	5

Table 3 shows the usage of the tools by the participants before they took part in the study. It indicates that wikis (76%) and discussion forums (78%) are currently the most widely used tools. Weblogs are only used by 11% of the asked students.

The results clearly show that the Weblog hype had not yet reached the surveyed students. Due to the fact that only about 11% of the students are currently using Weblogs, the results for this instrument are not published for the first part of the analysis. When it comes to the potential future usage of the instruments, Weblogs are taken into consideration again.

The following section presents the results for statements used in analyzing the usage in more detail.

Tables 4 and 5 present the current usage of the tools for private and educational purposes. First, the statement "I often use wikis or forums for private purposes" was presented.

Table 5 presents the results for the statement "I often use wikis or forums for educational purposes."

A huge majority (90%) stated that they use wikis for educational purposes and about two thirds (68%) used wikis for private purposes. Wikis are therefore more intensively used for

Table 3. Students already using the tools

	Wiki	Forum	Weblog
Yes	76%	78%	11%
No	24%	22%	89%

Table 4. Usage for private purposes

	Wiki	Forum
I totally agree	33%	33%
I generally agree	35%	29%
Neither...nor (neutral)	9%	9%
I slightly disagree	16%	17%
I disagree	8%	12%

Table 5. Usage for educational purposes

	Wiki	Forum
I totally agree	57%	22%
I generally agree	33%	29%
Neither...nor (neutral)	3%	12%
I slightly disagree	8%	24%
I disagree	1%	12%

educational purposes than for private purposes, whereas the usage of forums is exactly the opposite: They are more used for private purposes than for education.

The responses of the students concerning these statements were that wikis are foremost considered as a source of serious information, whereas forums are ideal for getting hints or clues to problems because of their privacy. Questions about computer problems, computer games, leisure activities, and so forth were mentioned. A repetition of this image can be identified when the disagreement with the statement is analyzed; 29% of the students do not or rarely use forums for private purposes compared to 36% for education.

Part 2: Experiences Made During the Study

This section presents the results of the study concerning experiences with the usage of the tools during the study.

Quality and Support for Learning

The following refers to statements concerning the quality of the contributions of wikis and discussion forums and their support for learning.

The results of the statement "The quality of contributions in wikis or forums is in general good" are presented in Table 6. The contributions of wikis are evaluated to be much better than those of forums.

The surveyed pupils had the opportunity to give reasons for their assessment concerning the quality of contributions via additional qualitative feedback. The following summarizes the addressed reasons.

One reason for the excelling grade for the quality of wikis is the "Wikipedian community." The term *wiki* is often seen as a synonym for the free online encyclopedia Wikipedia (http://www.wikipedia.org). Wikipedia is widely used for a great variety of tasks, including research on all topics needed for educational and private purposes.

In contrast to the good evaluation of the contributions of wikis, the open architecture of wikis was also mentioned. In most cases, this open architecture allows everyone to edit entries, which results in the uncertainty of whether the knowledge presented is correct or not. The quality of contributions in discussion forums was rated rather mediocre. Forums are primarily used for technical problems, especially computer-related problems; to get in contact with experts on certain topics, and to get information on online games.

The next statement, "The usage of wikis or forums leads to misunderstandings and confusion," is about the clarity of the contributions.

Only a minority think that the contributions are not clear and may lead to misunderstandings. In this case, wikis are also rated better than forums.

The next statements addressed the support of these instruments for learning. Table 8 summarizes the results for the statement "When reading contributions in wikis or forums, it is easier for me to acquire the learning contents."

More than half of the students express that reading contributions in wikis is helpful for learning, whereas only about 8% think that it is not helpful. Compared to forums, wikis were again much better evaluated, especially considering the big difference from the negative evaluations of forums.

Table 9 presents the learning support achieved by writing contributions. ("When writing contributions in wikis or forums, it is easier for me to acquire the learning contents.")

A different picture emerges in the statistics when comparing the evaluation of how writing an article or post supports the learning process. Here, forums take the lead when it comes to positive assessment. In both cases, there was a large number stating that writing is neither positive nor negative. The majority of the students read rather than wrote, while more students wrote in forums than in wikis.

Table 6. Perceived quality of contribution

	Wiki	Forum
I totally agree	38%	10%
I generally agree	52%	31%
Neither...nor (neutral)	10%	41%
I slightly disagree	2%	15%
I disagree	0%	4%

Table 7. Clarity of contributions

	Wiki	Forum
I totally agree	2%	4%
I generally agree	6%	18%
Neither...nor (neutral)	29%	37%
I slightly disagree	34%	27%
I disagree	29%	14%

Applicability for Communication and Community Building

The statement was formulated as follows: "Wikis or forums are appropriate to support communication."

The results clearly demonstrate that discussion forums are made for communication whereas wikis are rather seen as a kind of reference book or encyclopedia, as already mentioned above.

The results of the next statement, "Wikis or forums support the setup of communities," can be seen in Table 11.

Opinions about the applicability of wikis to establish a community is split. About 35% say that wikis are supportive of building a community compared to 25% who said that wikis do not support community building. The support of forums to build a community is rated much better: 50% indicated that forums are well suited to build a community. These results were to be expected because they confirm the nature of the instruments.

Fun Factor in Using the Instruments

In surveying whether students gain pleasure ("I enjoy using wikis or forums"), wikis again came out on top.

A majority enjoy using wikis (62%) and forums (56%). Considering the percentage of students who said that there is no ("I disagree") or little ("I slightly disagree") fun when using these instruments, wikis (6%) are much better rated than forums (18%).

Part 3: Potential Future Usage of the Tools

The third section of the empirical study deals with the potential usage by students who had not used the instruments before the study. Students gained knowledge and experiences by using the tools during the study by themselves or on the basis of the reported experiences made by their fellow students.

The first statement, "I will use wikis, forums, or Weblogs for educational purposes in the future," yielded the results in Table 13.

According to this study, wikis will have a bright future and will be used often for educational purposes, whereas forums will be used less often. About 54% of the surveyed students had the intention of using wikis more or less often in the future. About 16% did not think that they will use wikis often in the future and 30% are not yet sure if they will use this instrument.

The results for forums and Weblogs indicate no clear trend, but forums were rated slightly higher than Weblogs; 39% of the students stated that they can imagine using forums in the future for their education compared to 36% for Weblogs. At the other end of the scale, 45% did not have the intention to use forums (40% for Weblogs).

The statement "I will use wikis, forums, or Weblogs for private purposes in the future" leads to similar results.

Table 8. Reading contributions helps to acquire contents

	Wiki	Forum
I totally agree	23%	8%
I generally agree	36%	21%
Neither...nor (neutral)	32%	31%
I slightly disagree	5%	25%
I disagree	3%	15%

Table 9. Writing contributions helps to acquire contents

	Wiki	Forum
I totally agree	8%	7%
I generally agree	13%	19%
Neither...nor (neutral)	45%	34%
I slightly disagree	14%	22%
I disagree	19%	17%

Table 10. Applicability for communication

	Wiki	Forum
I totally agree	9%	39%
I generally agree	33%	37%
Neither...nor (neutral)	29%	17%
I slightly disagree	15%	4%
I disagree	15%	3%

Table 11. Support for community building

	Wiki	Forum
I totally agree	10%	28%
I generally agree	25%	32%
Neither...nor (neutral)	39%	23%
I slightly disagree	15%	11%
I disagree	10%	6%

From this point of view, wikis are again the leading instrument, followed by forums and then Weblogs.

It must be said that the responses to this set of statements represented feelings, attitudes, and opinions about instruments that had not yet been used by the asked participants. The purpose of posing these statements was to gain insight into the mind-set in regard to these instruments.

DISCUSSION

The results clearly show that wikis are currently the most often used instrument and furthermore have the greatest potential as a tool for learning and knowledge management in the field of learning; these findings are in line with other empirical studies (Bendel, 2007; Chao, 2007).

Other studies (McGill et al., 2005; Nicol & MacLeod, 2004) report that a shared workspace helps to support collaborative learning; the possibility of being able to access and contribute to the development of resources at any time and from any location was especially appreciated by the students.

The survey at hand made a distinction between reading and writing contributions to wikis and discussion forums. The results show that 59% of students said reading contributions in wikis is helpful for learning (stating “I totally agree” and “I generally agree”) while only 21% stated that writing in wikis is helpful for learning. Reading contributions in forums helped 29% of the participants, whereas writing in forums was helpful to 26%. This survey supports the general statement that a shared workspace that supports a constructivist and learner-centered approach is helpful for learning.

The pedagogical value in the context of learning is described in several publications (Babcock, 2007; Hurst, 2005). Weblogs can foster the establishment of a learning and teaching environment in which students and teachers experience a greater degree of equality and engagement. Du and Wagner (2007) published results of a study

Table 12. Fun Factor in using the instruments

	Wiki	Forum
I totally agree	26%	19%
I generally agree	36%	37%
Neither...nor (neutral)	31%	26%
I slightly disagree	5%	14%
I disagree	1%	4%

Table 13. Future usage in educational context (current nonusers)

	Wikis	Forums	Weblogs
I totally agree	18%	16%	13%
I generally agree	36%	23%	23%
Neither...nor (neutral)	30%	16%	24%
I slightly disagree	9%	12%	13%
I disagree	7%	33%	27%

Table 14. Future usage in private context (current nonusers)

	Wikis	Forums	Weblogs
I totally agree	11%	14%	9%
I generally agree	36%	23%	22%
Neither...nor (neutral)	30%	25%	24%
I slightly disagree	14%	7%	16%
I disagree	9%	32%	28%

of an information systems undergraduate course (31 participants). This study indicated that the performance of students' Weblogs was a significant predictor for learning outcomes, while traditional coursework was not. Moreover, individuals' cognitive construction efforts to build their own mental models and social construction efforts to further enrich and expand knowledge resources appeared to be two key aspects of constructivist learning with Weblogs. According to this study, there is a potential benefit of using Weblogs as a knowledge construction tool and a social learning medium.

According to the survey at hand, Weblogs are not yet widely used, and their potential seems to be limited. It can be assumed that these limited prospects will change when the penetration of Weblogs into the daily routine of the students increase—for private as well as for educational purposes.

To avoid possible pitfalls about the application of these instruments in the context of learning, some social and psychological issues must be taken into consideration (Kreijns, Kirschner, & Jochems, 2003). Social interaction is essential for members of a team to get to know each other, commit to social relationships, develop trust, and develop a sense of belonging in developing a learning community. The size and the composition of the learning communities seem to be important factors in how interaction and communication within the learning community will take place (Dooley & Wickersham, 2007). There are also many unresolved issues, like the provision of the technology and the services, intellectual property rights and digital rights management, the security of data, access restrictions to the content, and information ethics (Attwell, 2006; McGill et al., 2005; Sharma & Maleyeff, 2003).

CONCLUSION

The aim of this contribution was to investigate the experiences of students using social software tools in the context of learning. Wikis, Weblogs, and discussion forums are typical social software tools and were used for this survey.

The results clearly show that wikis and discussion forums can support learning and collaboration. The usage of Weblogs in this study was limited and hence no statements about their applicability can be made. In order to assure a successful implementation of these tools, social and psychological issues must be taken into consideration as well.

The results of this study are the basis for the introduction of social software into education to help students set up individual learning environments. These learning environments should support lifelong learning.

There are likely to be other unplanned consequences of the intensive use of the Internet in general and social software especially. Further research is needed to explore possible problems and solutions.

The results of the empirical survey indicate that a long-term study in combination with the further development of social software tools may be promising.

REFERENCES

Aggarwal, A. K., & Legon, R. (2006). Web-based education diffusion. *International Journal of Web-Based Learning and Teaching Technologies, 1*(1), 49-72.

Alexander, B. (2006). Web 2.0: A new wave of innovation for teaching and learning? *Educause Review, 41*(2), 32-44.

Attwell, G. (2006). *Personal learning environment.* Retrieved May 17, 2007, from http://www.knownet.com/writing/weblogs/Graham_Attwell/entries/6521819364

Augar, N., Raitman, R., & Zhou, W. (2004, December 5-8). Teaching and learning online with wikis. In R. Atkinson, C. McBeath, D. Jonas-Dwyer, & R. Phillips (Eds.), *Beyond the Comfort Zone*: *Proceedings of the 21st ASCILITE Conference*, Perth, Western Australia. Retrieved May 17, 2007, from http://www.ascilite.org.au/conferences/perth04/procs/contents.html

Babcock, M. (2007). Learning logs in introductory literature courses. *Teaching in Higher Education, 12*(4), 513-523.

Baumgartner, P. (2004). The Zen art of teaching. *Communication and Interactions in eEducation*. Retrieved May 17, 2007, from http://bt-mac2.fernuni-hagen.de/peter/gems/zenartofteaching.pdf

Baumgartner, P. (2005). Eine neue lernkultur entwickeln: Komptenzbasierte ausbildung mit blogs und e-portfolios. In V. Hornung-Prähauser (Ed.), *ePortfolio Forum Austria: Proceedings of the ePortfolio Austria 2005*, Salzburg, Austria (pp. 33-38).

Baumgartner, P., Häfele, H., & Maier-Häfele, K. (2004). *Content management systeme in e-education: Auswahl, potenziale und einsatzmöglichkeiten.* Innsbruck, Austria: Studienverlag.

Bendel, O. (2006). Wikipedia als methode und gegenstand der lehre. In K. Hildebrand (Ed.), *HMD: Praxis der wirtschaftsinformatik: Vol. 252. Social software* (pp. 82-88). Heidelberg, Germany: dpunkt-Verlag.

Böttger, M., & Röll, M. (2004, December 15-17). Weblog publishing as support for exploratory learning on the World Wide Web. In P. Isaias, K. Demetrios, & G. Sampson (Eds.), *Cognition and Exploratory Learning in Digital Age (CELDA 2004): Proceedings of the IADIS International Conference*, Lisbon, Portugal. IADIS Press.

Brahm, T. (2007). Social software und personal broadcasting: Stand der forschung. In S. Seufert & D. Euler (Eds.), *Ne(x)t generation learning: Wikis, blogs, mediacasts & Co. Social software und personal broadcasting auf der spur* (pp. 20-38). SCIL Arbeitsbericht.

Chao, J. (2007). Student project collaboration using wikis. In *Proceedings of the 20th Conference on Software Engineering Education & Training* (pp. 255-261). Washington, DC.

Coates, T. (2005). *An addendum to a definition of social software.* Retrieved March 4, 2007, from http://www.plasticbag.org/archives/2005/01/an_addendum_to_a_definition_of_social_software

Craig, E. (2007). Changing paradigms: Managed learning environments and Web 2.0. *Campus-Wide Information Systems, 24*(3), 152-161.

Dooley, K. E., & Wickersham, L. E. (2007). Distraction, domination and disconnection in whole-class online discussions. *Quarterly Review of Distance Education, 8*(1), 1-8.

Du, H. S., & Wagner, C. (2005). Learning with Weblogs: An empirical investigation. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05): Track 1* (p. 7b). Washington, DC: IEEE Computer Society.

Du, H. S., & Wagner, C. (2007). Learning with Weblogs: Enhancing cognitive and social knowledge construction. *IEEE Transactions on Professional Communication, 50*(1), 1-16.

Duffy, P. D., & Bruns, A. (2006). The use of blogs, wikis and RSS in education: A conversation of possibilities. In *Proceedings of the Online Learning and Teaching Conference 2006*, Brisbane, Australia (pp. 31-38).

Efimova, L., & Fiedler, S. (2004, March 24-26). Learning webs: Learning in Weblog networks. In P. Kommers, P. Isaias, & M. B. Nunes (Eds.), *Web based communities: Proceedings of the IADIS International Conference 2004*, Lisbon, Portugal (pp. 490-494). IADIS Press. Retrieved May 17, 2007, from https://doc.telin.nl/dscgi/ds.py/Get/File-35344

Evans, C., & Sadler-Smith, E. (2006). Learning styles in education and training: Problems, politicisation and potential. *Education + Training, 48*(2/3), 77-83.

Fiedler, S. (2004). Personal Webpublishing as a reflective conversational tool for self-organized learning. In T. Burg (Ed.), *BlogTalks* (pp. 190-216). Retrieved May 7, 2007, from http://seblogging.cognitivearchitects.com/stories/storyReader$963

Hammond, T., Hannay, T., & Lund, B. (2004, December). The role of RSS in science publishing: Syndication and annotation on the Web. *D-Lib Magazine*. Retrieved April 20, 2007, from http://www.dlib.org/dlib/december04/hammond/12hammond.html

Hurst, B. (2005). My journey with learning logs. *Journal of Adolescent & Adult Literacy, 49*(1), 42-46.

Jonassen, D. H., Mayes, T., & McAleese, R. (1993, May 14-18). A manifesto for a constructivist approach to uses of technology in higher education. In T. M. Duffy, J. Lowyck, D. H. Jonassen, & T. Welsh (Eds.), *Xpert.press: Vol. 105. Designing Environments for Constructive Learning: Proceedings of the NATO Advanced Research Workshop on the Design of Constructivist Learning Environments Implications for Instructional Design and the Use of Technology*, Leuven, Belgium (pp. 231-247). Berlin, Germany: Springer.

Kantel, J. (2003). *Vom Weblog lernen: Community, peer-to-peer und eigenständigkeit als ein modell für zukünftige wissenssammlungen*. Retrieved March 12, 2007, from http://static.userland.com/sh4/gems/schockwellenreiter/blogtalktext.pdf

Kerres, M. (2006). Web 2.0 and its implications to e-learning. In T. Hug, M. Lindner, & P. A. Bruck (Eds.), *Micromedia & E-Learning 2.0: Gaining the Big Picture: Proceedings of Microlearning Conference 2006*, Innsbruck, Austria. University Press.

Kerres, M. (2007). Microlearning as a challenge for instructional design. In T. Hug & M. Lindner (Eds.), *Didactics of microlearning.* Münster, Germany: Waxmann. Retrieved March 10, 2007, from http://mediendidaktik.uni-duisburg-essen.de/files/Microlearning-kerres.pdf

Klamma, R., et al. (2006). Social software for professional learning: Examples and research. In B. Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. Sampson, & W. Didderen (Eds.), *Advanced Learn-*

ing Technologies: ICALT 2006, Los Alamitos, CA (pp. 912-917). IEEE Computer Society.

Köllinger, P. (2002). *Report e-learning in deutschen unternehmen: Fallstudien, konzepte, implementierung* (1. Aufl.). Düsseldorf, Germany: Symposion.

Kreijns, K., Kirschner, P., & Jochems, W. (2003). Identifying the pitfalls for social interaction in computer-supported collaborative learning environments: A review of the research. *Computers in Human Behaviour, 19*, 335-353.

Kutscher, N., Klein, A., & Iske, S. (2005). Differences in Internet usage: Social inequality and informal education. *Social Work & Society, 3*, 215-233.

Lenhart, A., & Madden, M. (2005). Teen content creators and consumers. In *Pew Internet Project Data Memo*. Retrieved June 21, 2007, from http://www.pewinternet.org/pdfs/PIP_Teens_Content_Creation.pdf

Lewinson, J. (2005). Asynchronous discussion forums in the changing landscape of the online learning environment. *Campus-Wide Information Systems, 22*(3), 162-167.

McGill, L., Nicol, D., Littlejohn, A., Grierson, H., Juster, N., & Ion, W. J. (2005). Creating an information-rich learning environment to enhance design student learning: Challenges and approaches. *British Journal of Educational Technology, 36*(4), 629-642.

Nicol, D. J., & MacLeod, I. A. (2005). Using a shared workspace and wireless laptops to improve collaborative project learning in an engineering design class. *44*(4), 459-475.

Oblinger, D., & Oblinger, J. (2005). Is it age or IT: First steps towards understanding the Net generation. In D. Oblinger & J. Oblinger (Eds.), *Educating the Net generation*. Educause. Retrieved June 12, 2007, from http://www.educause.edu/educatingthenetgen/

O'Reilly, T. (2005). *Web 2.0: Compact definition?* Retrieved May 17, 2007, from http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html

Otto, H.-U., Kutscher, N., Klein, A., & Iske, S. (2005). *Social inequality in the virtual space: How do young people use the Internet? Results from empirical research about online use differences and acquiring patterns of young people*. Retrieved January 16, 2008, from http://www.kib-bielefeld.de/externelinks2005/Social_Inequality%20KIB.pdf

Prensky, M. (2001a). *Digital natives, digital immigrants: Part 1*. Retrieved April 13, 2007, from http://www.marcprensky.com/writing/Prensky-Digital Natives, Digital Immigrants-Part1.pdf

Prensky, M. (2001b). *Digital natives, digital immigrants: Part 2*. Retrieved April 13, 2007, from http://www.marcprensky.com/writing/Prensky-Digital Natives, Digital Immigrants-Part2.pdf

Resnick, M. (2002). Rethinking learning in the digital age. In G. S. Kirkman, P. K. Cornelius, J. D. Sachs, & K. Schwab (Eds.), *Global information technology report 2001-2002: Readiness for the networked world* (pp. 32-37). New York: Oxford University Press.

Richter, A., & Koch, M. (2007). *Social software: Status quo und zukunft* (Tech. Rep. No. 2007-01). Fakultät für Informatik, Universität der Bundeswehr München. Retrieved June 1, 2007, from http://www.unibw.de/wow5_3/forschung/social_software/

Schulmeister, R. (2004). Diversität von studierenden und die konsequenzen für elearning. In D. Carstensen & B. Barrios (Eds.), *Kommen die digitalen medien in die jahre? Medien in der wissenschaft* (Vol. 29, p. 133-144). Münster, Germany: Waxmann.

Schulmeister, R. (2005). *Lernplattformen für das virtuelle lernen: Evaluation und didaktik* (2. Aufl.). München, Germany: Oldenbourg.

Seufert, S. (2007). Ne(x)t generation learning: Was gibt es neues über das lernen? In S. Seufert & D. Euler (Eds.), *Ne(x)t generation learning: Wikis, blogs, mediacasts & Co. Social software und personal broadcasting auf der spur* (pp. 2-19). SCIL Arbeitsbericht.

Seybert, H. (2007). *Gender differences in the use of computers and the Internet.* Retrieved January 15, 2008, from http://epp.eurostat.ec.europa.eu/portal/page?_pageid=1073,46587259&_dad=portal&_schema=PORTAL&p_product_code=KS-SF-07-119

Sharma, P., & Fiedler, S. (2004, June 30-July 2). Introducing technologies and practices for supporting self-organized learning in a hybrid environment. In K. Tochterman & H. Maurer (Eds.), *Proceedings of I-Know '04*, Graz, Austria (pp. 543-550).

Sharma, P., & Maleyeff, J. (2003). Internet education: Potential problems and solutions. *17*(1), 19-25.

Shirky, C. (2003). *Social software and the politics of groups.* Retrieved May 3, 2007, from http://shirky.com/writings/group_politics.html

Tangney, B., FitzGibbon, A., Savage, T., Mehan, S., & Holmes, B. (2001). Communal constructivism: Students constructing learning for as well as with others. In C. Crawford, D. A. Willis, R. Carlsen, I. Gibson, K. McFerrin, J. Price, & R. Weber (Eds.), *Proceedings of Society for Information Technology and Teacher Education International Conference 2001* (pp. 3114-3119). Norfolk, VA: AACE.

Thijssen, T., & Vernooij, F. (2002). Breaking the boundaries between academic degrees and lifelong learning. In T. A. Johannessen (Ed.), *Educational innovation in economics and business: Vol. 6. Teaching today: The knowledge of tomorrow* (pp. 137-156). Dordrecht, The Netherlands: Kluwer Academic.

This work was previously published in International Journal of Web-Based Learning and Teaching Technologies, Vol. 3, Issue 3, edited by L. Esnault, pp. 16-33, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.13
Continuous Curriculum Restructuring in a Graduate Software Engineering Program

Daniela Rosca
Monmouth University, USA

William Tepfenhart
Monmouth University, USA

Jiacun Wang
Monmouth University, USA

Allen Milewski
Monmouth University, USA

ABSTRACT

The development, maintenance and delivery of a software engineering curriculum present special challenges not found in other engineering disciplines. The continuous advances of the field of software engineering impose a high frequency of changes reflected in the curriculum and course content. This chapter describes the challenges of delivering a program meeting the needs of industry and students. It presents the lessons learned during 21 years of offering such a program, and dealing with issues pertaining to continuous curriculum and course content restructuring, the influence of the student body on the curriculum and course content. The chapter concludes with our recommendations for those who are seeking to create a graduate program in software engineering, with a special note on the situations where an undergraduate and graduate program will need to coexist in the same department.

INTRODUCTION

The objective of this chapter is to prepare those who are seeking to introduce a graduate program in software engineering (SE) for the challenges they will face. Towards that end, the lessons learned during 21 years of offering such a program at Monmouth University will be presented. As it will be demonstrated, the development, main-

tenance and delivery of a software engineering curriculum present special challenges not found in other engineering disciplines.

This chapter describes the challenges of delivering a program that meets the needs of industry and students in a highly dynamic field. The evolution of the curriculum induced by the domain's continuous advances and evolution in industry practice will be presented. The special meaning of continuous course content development in software engineering will be argued through issues pertaining to dated textbooks, ever-changing programming languages, operating systems, and software tools. The chapter will also present our experience in dealing with the diversity of the student body, and its influence on the curriculum and course content. The chapter will conclude with our recommendations for constructing a similar program, with a special emphasis on situations where an undergraduate and graduate program in software engineering will need to coexist in the same department.

BACKGROUND

Although software engineering was recognized as a field in 1968 at the NATO sponsored conference on the subject (Naur, 1968), it took universities and colleges a significant amount of time to respond to that fact. It was not until 1986 that Monmouth University (MU) started a graduate program dedicated to software engineering, which was offered by its Computer Science Department. In 1995 Monmouth created the first Software Engineering Department in United States. Now it is one of the pioneer universities offering a bachelor's degree in software engineering.

One motivation for creating a separate software engineering program and department was the awareness of the skills that industry would like students to have upon graduation, which are not stressed by most computer science curricula. These skills include teamwork, communications, time management, engineering problem solving, quantitative and qualitative process management, reuse, requirements management, system architecture, testing and project management.

As one of the few universities with extensive and comprehensive experience in offering software engineering programs, we have learned much about providing such a program. With more and more undergraduate software engineering programs appearing, we feel it is beneficial to other institutions for us to share the problems encountered and lessons learned over the past 21 years. A summary of the problems encountered and the lessons learned are presented here:

- **Continuous curriculum restructuring.** One can expect to revisit the overall curriculum of the program every four to five years, in order to accommodate changes in industry practice and educational expectations. This is reflected also in the historical investigation of the graduate software engineering curriculum reported in (Duggins, 2002).
- **Continuous course content restructuring.** It is critically needed due to the dynamics of the field. The continuous development of course content implies also a continuous development of course projects, and dealing with dated textbooks, ever changing operating systems, programming languages and software tools.
- **Hiring and retaining faculty.** The need for new faculty to have a record of sustained scholarly accomplishments and industrial experience enforces great restrictions on the number of available candidates, as it was also notified by Glass (2003). Retaining faculty is complicated by the fact that in addition to performing their normal teaching duties SE faculty must continually keep up with changes in the field as a whole.

- **Influence of the diversity of the student body on the curriculum and course content.** Issues raised by a diversity of educational backgrounds, employment status, educational goals, and communication skills introduce challenges that need to be dealt with by any software engineering program.

The remainder of the chapter discusses in detail the topics presented above. It begins with the presentation of the curriculum evolution over the history of our program. Then it discusses various issues involved in the continuous changes of the software engineering course content. The subsequent section outlines our experience in hiring and retaining the faculty, followed by a presentation of the student body influence of the diversity of the curriculum and course content. The chapter concludes with the presentation of our recommendations for those interested in starting a graduate program in software engineering, and future trends of the MU program. This discussion will emphasize the accommodation of an undergraduate and graduate program in software engineering in the same department.

CONTINUOUS CURRICULUM RESTRUCTURING

Over its short history, software engineering (SE) as a field has been a moving target. We have observed the introduction of the capability maturity model, the unified modeling language, personal and team software process, and corporate adherence to ISO Standards emerge as major forces within software engineering organizations. We have observed important changes in analysis and design with transitions from structured analysis and design ,to object-oriented analysis and design. Even the architectures being released today have shifted from client-server architectures to distributed architectures with the current trend being focused on service-oriented architectures.

A curriculum that addresses the skills and practices required by professionals in this field must continuously reinvent itself over time. The curriculum of Software Engineering changes with a frequency on the order of twice a decade as opposed to decades for engineering (Clough, 2005) and sciences (Stryer, 2003), in general. Just about every aspect of the software engineering curriculum is susceptible to change over a decade. In order to accommodate industry's needs and to keep pace with the advances of software engineering as a field, we have added or dropped courses, and added new tracks and programs. The decisions were made in the context of creating and maintaining a balance between the theory, technology and practical aspects of software engineering.

Changing the curriculum can not be performed in an ad hoc manner. We follow a well-defined process. First, the faculty discusses the need for change based on feedback from industry, students, and current publications. The acquisition of feedback is a continuous process that is assisted through an industry advisory board, alumni surveys, student exit interviews, student learning outcomes assessment, periodical evaluation of the program by an external reviewer who is a prominent figure in software engineering education, and attendance at professional meetings. We have established a set of learning outcomes that we monitor on a regular basis and we take into consideration when the need for a significant curriculum change is required.

Next, the program director writes a proposal identifying the new curriculum, and any additional courses that might be required. This effort includes writing a complete syllabus for each course that is introduced, modifying existing syllabi for courses

with significant content changes, and a justification for each course that is dropped.

The proposal is put forth to the faculty in the department for comments. Comments include challenges to the changes in course content and discussion of the overall package. The syllabi for the various courses may undergo several iterations.

Once the proposal is approved within the department, the proposal is sent to the chairs meeting within the school. Often times, changes to the software curriculum may require changes in the course content of other departments (e.g., computer science and the business school). If significant push-back arises from the other departments within the school, the proposal is reconsidered by department.

At the school chairs meeting it can be decided that a stronger business case is required. An external body typically develops this business case. It is either a survey developed by an independent firm, or by an external industrial advisory committee. The business case reflects the needs and state of industry, which will attract new students.

After the eventual suggestions for change are incorporated into the proposal, it is submitted to the university graduate studies committee. Here the curriculum is evaluated in terms of its consistency with the University Policies applied to graduate programs. This includes establishing maximum class sizes, the number of contact hours, assignment of lab fees, and other factors.

Next we present the evolution of the Monmouth University's Graduate Software Engineering Curriculum. This evolution shows a gradual transition from a software engineering program created inside a Computer Science department, towards a program with engineering courses that span the entire software lifecycle. It incorporates the results of a strong collaboration between academia and industry (Powell, 1997).

The Initial Curriculum (1986)

The initial software engineering curriculum at Monmouth University consisted of 30 credits, with 6 core courses and 4 electives (see Figure 1). The core courses covered in detail only the implementation (in Ada) and project management aspects of the software lifecycle, due to the limited availability of faculty with an appropriate background. The curriculum looked more like "a computer science curriculum with an engineering flavor" (Dart, 1997), covering classic computer science courses such as algorithms, operating systems, computer architecture and database management systems.

Students' practical training was accomplished in a 3-credit practicum course. This course consisted of a team project to develop a software system from initial requirements to the final, tested and documented product. The early curriculum was biased more on theoretical aspects (notice the heavy concentration on mathematical foundations of SE), with less exposure to specific SE technology and practice, as was very early recommended in (Ford, 1987).

1991 Curriculum Changes

This curriculum added a number of SE courses, including formal methods, formal specifications, software process and SE environments (see Figure 2). However, it still had a bias towards computer science, offering an artificial intelligence course, 4 courses of mathematical foundations and formal methods, and 4 courses in network technology, due to our geographic location in an area dominated by the telecommunications industry. It was similar to the First MSE model curriculum (Ardis, 1989a; Ardis 1989b) that recommended a set of 10-12 courses, which comprised 6 core courses, 3 or more electives and a two-semester practicum

Figure 1. 1986 curriculum

Core Courses
(6 Courses = 18 Credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 505 Programming-in-the-large (3 credits)
SE 510 Computer Network Design (3 Credits)
SE 516 Software Engineering I (3 credits)
SE 518 Project Management (3 credits)
SE 525 System Project Implementation (3 credits)
Elective Courses
(4 Courses = 12 Credits)
SE 502 Mathematical Foundation of Software Engineering II (3 Credits)
SE 506 Programming-in-the-small (3 credits)
SE 509 Programming Languages (3 Credits)
SE 511 Protocol Engineering (3 Credits)
SE 512 Algorithms Design and Analysis (3 Credits)
SE 514 Computer Architecture (3 Credits)
SE 515 Operating Systems Implementation (3 Credits)
SE 517 Software Engineering II (3 Credits)
SE 519 Database Management (3 Credits)

Figure 2. 1991 curriculum

Core Courses
(6 Courses = 18 Credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 505 Software System Design (3 credits)
SE 506 Formal Methods in Programming (3 credits)
SE 516 Software Engineering (3 credits)
SE 518 Project Management (3 credits)
SE 525 System Project Implementation (3 credits)

Elective Courses
(4 Courses = 12 Credits)
SE 502 Mathematical Foundation of Software Engineering II (3 Credits)
SE 503 Intro. to Computer-Communication Networking (3 Credits)
SE 509 Programming Languages (3 Credits)
SE 510 Computer Network Design (3 Credits)
SE 511 Protocol Engineering (3 Credits)
SE 519 Database Management (3 Credits)
SE 522 Software Engineering Environments (3 Credits)
SE 532 Software Process Quality (3 Credits)
SE 534 Formal Specifications of Software Systems (3 Credits)
SE 536 Fundamentals of Computer Security (3 Credits)
SE 538 Advanced Topics in Networking Topology (3 Credits)
SE 540 Introduction to Artificial Intelligence (3 Credits)

project. However, due to the lack of qualified faculty, the core courses offered were not able to cover the entire software lifecycle.

1995 Curriculum Changes

In 1995 the curriculum was substantially changed to include 36 credits, with 10 core and 2 elective courses (see Figure 3), in order to comply with the Software Engineering Institute model curriculum (Ardis, 1989) and the 1991 Computing Curriculum guidelines (Tucker, 1991; Ford 1991). That curriculum covered the entire software lifecycle in detail, by offering 3 new courses, specifically in requirements, implementation and reuse, and testing and quality. A former elective, software systems security, became a core course.

Having such a heavy core, this curriculum offered little flexibility for learning aspects of SE that students would be most interested in. Another major change was reflected in the introduction of several new courses that would form 6 credit elective specialization tracks: in distributed software systems, software management, information systems, and real-time systems. These tracks were introduced as a response to the needs and feedback from the local industry, and government collaborators (Powell,1997). The curriculum change was made possible by hiring faculty with both theoretical background and working experience in industry, supplemented

Figure 3. 1995 curriculum

Core Courses
(10 Courses = 30 Credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 504 Principles of Software Engineering (3 Credits)
SE 505 Software System Design (3 credits)
SE 506 Formal Methods in Software (3 credits)
SE 507 Software Systems Requirements (3 Credits)
SE 508 Software Implementation and Reuse (3 Credits)
SE 512 Software Testing and Quality (3 Credits)
SE 513 Software Systems Security (3 Credits)
SE 518 Software Project Management (3 credits)
SE 525 System Project Implementation (3 credits)
Advanced/Elective Specialization Tracks
(2-course track = 6 Credits)
Distributed Software Systems
SE 526 Networked Software Systems I
SE 527 Networked Software Systems II
Software Management
SE 531 Software Organization Management
SE 532 Software Quality Management
Information Systems
SE 541 Information Systems Architecture
SE 542 Information Systems Engineering
Real-Time Systems
SE 551 Real-Time Software Analysis & specification
SE 552 Real-Time Software Design & Implementation

with a substantial help from adjunct faculty with expertise in specialized areas of SE.

1996 Curriculum Changes

In 1996 minor changes were made in the curriculum. It remained a 36-credit program, with 9 core and 3 elective courses, which offered a bit more flexibility than the previous program. The curriculum covered all the aspects of the software lifecycle. The capstone course was either a 3-credit practicum, or 6-credits of thesis research. The introduction of a thesis option was made possible by attracting faculty with the desire to engage in research activities.

1998 Curriculum Changes

The 1998 curriculum represented another major change by providing for much more flexibility in a 36 credit program, with 5 core, 5 elective courses, and a 6-credit practicum or a 6-credit thesis (see Figure 4).

The recognition of the importance of exposure to practical experience in a software engineering program has lead to the increase of the practicum project from 3 to 6 credits, and to the introduction of term projects in most of the courses in the curriculum. This is similar to the recommendations of the First MSE curriculum (Ardis,1989) of offering a two semester practicum and as much as 30% of the program be dedicated to project work.

The MU curriculum continued to follow the software life-cycle model, as opposed to the CMU Model (Garlan, 1995), which emphasized teaching "cross-cutting principles of software development" throughout the curriculum. As such, the CMU Model offered five core courses organized around modeling, methods of development, management, analysis and architecture. Also, they included a software development studio for the development of practice skills, during the entire duration of the program.

The 1998 MU curriculum has added a new course, The Process of Engineering Software, which largely follows Watts Humphrey's Personal Software Process (PSP) principles (Humphrey, 2005). The introduction of this course was justified by the need for graduates who are aware and have the necessary skills for predictably producing high quality systems, in a timely and cost effective manner, using reusable components as much as possible in their work. In spite of the hard work necessary for the manual input of the data for the various forms and templates involved in the PSP, students have given us very positive feedback about the usefulness of the principles learned in this course. For alleviating the clerical work related to the manual input of data, we created a semi-automated tool to support the PSP process (Rosca, 2001). This tool was the result of a two-semester practicum project of one group of students.

Two of the former core courses, mathematical foundations of SE, and principles of SE, have been transformed into preparatory (bridge) courses (see Figure 4). Together with three other programming courses the "bridge" program is offered for students with an undergraduate major other than computer science, computer engineering, electrical engineering, or information systems. After taking the 15 credit preparatory courses and a one-semester project course, students can receive a certificate in software development if they don't wish to pursue a Master's program.

The elective courses included in this curriculum were necessary for completing a chosen specialization track, such as organizational management, telecommunications, embedded systems, and information systems. These 15-credit tracks were much more comprehensive than their counterparts in the 1995 curriculum. They comprise courses from other disciplines such as business, electrical engineering and

Figure 4. 1998 curriculum

Preparatory Courses (15 Credits)
CS 500 Program Development
CS 503 Fundamental Algorithms I
CS 505 Operating Systems
SE 501 Mathematical Foundation of Software Engineering
SE 504 Principles of Software Engineering

Core Courses (15 Credits)
SE 500 The Process of Engineering Software
SE 505 Software System Design
SE 506 Formal Methods in Software
SE 507 Software Systems Requirements
SE 512 Software Testing and Quality

Capstone Course (6 credits) – Practicum/Thesis

Specialisation Tracks (15 Credits)

Organizational Management Track **Required** (9 Credits) SE 531 Software Organizational Management SE 532 Software Quality Management SE 518 Software Project Management **Guided Electives** (6 Credits) BM 525 Management of Human Resources BM 565 Management of Technology SE 560 Software Risk Management SE 565 Software Metrics	**Telecommunications Track** **Required Courses** (9 Credits) EE 537 Wireless Communications SE 526 Network Software System I SE 527 Network Software System II **Guided Electives** (6 Credits) CS 526 Performance Evaluation CS 535 Telecommunications EE 505 Communications Technology EE 581 Data Networks SE 513 Software System Security SE 598T Special Topics (Telecommunications)
Embedded Systems Track **Required Courses** (9 Credits) SE 526 Network Software System I SE 551 Real-Time Software Analysis and Spec. SE 552 Real-Time Software Design and Impl. **Guided Electives** (6 Credits) CS 525 Simulation CS 526 Performance Evaluation EE 509 Digital Signal Processing SE 508 Software Implementation and Reuse SE 513 Software System Security SE 527 Network Software System II	**Information Management Track** **Required Courses** (9 Credits) SE 541 Information Systems Architecture SE 542 Information System Engineering SE 518 Software Project Management **Guided Electives** (6 Credits) BM 520 Information System in Organisation BM 565 Management of Technology BM 571 Introduction to US Health Care CS 517 Database Systems CS 530 Knowledge-Based Systems SE 508 Software Implementation and Reuse SE 526 Network Software System I

computer science. However, students have been able to select elective courses across tracks if they didn't want to pursue a specialization. A brief description of the specialization tracks is given next.

The Organisational Management track prepares students to become software development managers or specialists in software process improvement. Topics of study include process improvement, quality management, organisational development and management, risk management and project planning and management.

The Telecommunications track prepares students to become specialists in telecommunications. Topics of study include networks, software systems security, and evaluation of telecommunications systems.

The Embedded Systems track prepares students to become specialists in embedded systems development. Topics of study include specification and analysis of embedded real-time systems requirements, design and implementation of embedded real-time software systems, performance evaluation of embedded real-time software systems, and development of real-time components.

The Information Management track prepares students to become chief information officers or specialists in information systems integration and development. Topics of study include information technology management, specification and analysis of information systems, evaluation of information systems, and development of information systems software components.

2002 Curriculum Change

In 2002, the only curriculum change was the addition of a new specialization track: the Management of Software Technology, offered in collaboration with the Monmouth University School of Business. The idea of this track grew out of the recognition that industry is outsourcing increasing amounts of software development. This track prepares students to be chief technology officers or specialists in the acquisition of software systems for businesses. Topics of study include assessing the impact that software can have on organizations, the development of requirements for system acquisition via purchase or outsourcing, the assessment of software technologies with regard to organizational needs, and implementing a controlled introduction of technology into an organization.

All the knowledge areas of the Software Engineering Body of Knowledge (SWEBOK) project (Bourque, 2004) can be identified in this curriculum.

CONTINUOUS DEVELOPMENT OF COURSE CONTENT

Technologically, the computing field has undergone significant changes that have forced alterations in the material taught within Software Engineering courses. Since the inception of our SE Master's program, we have witnessed the widespread adoption of Object-Orientation (along with massive changes in techniques and methodologies), the phenomenal explosion of the World Wide Web, the emergence of Java, and the move of security requirements from corporate to consumer platforms, just to name a few of these changes. Therefore, the material covered within a curriculum that addresses the technological understanding required by professionals in this field, needs to be continuously updated over time.

This problem emerges in several different forms. In particular,

- Continuous course content changes
- Dated textbooks

- Operating system/programming language biases
- Continuous development of course projects

Each of these areas is discussed in greater detail in the paragraphs that follow.

Course Content Changes

One can expect to have to revise course material every year. This is necessary to accommodate technological changes and to incorporate new industrial practices. For example, since the inception of our program we have changed the programming languages taught in class from Ada, to C++ and Java; in the requirements engineering course object-oriented analysis methods were added to the structured analysis methods (Rosca, 2000); in the design course a transition was made from structured design to object-oriented design, component-based design, and architectural design. In the testing course we have added segments on testing applications that are constructed using commercial off-the-shelf (COTS) components, using automated testing and test management tools. For project management more content was gradually introduced on the use of scheduling tools, such as MS project, risk simulators, like Risk+, and discussion of the use of buffer tasks in the planning of software development projects (McDonald, 2000). The burden of this continuous course creation or updating could be alleviated in the future by the curricular materials offered by the SWENET project (Lutz, 2003), created by the professionals in the SE community for the use of the community, at large.

The continuous revisions of course material constitutes a significant amount of work on the part of the faculty. In as little as three years, the changes within the field are significant enough that many courses have to be totally redesigned. The adoption of UML and its subsequent evolution has forced revisions in design diagrams, the vocabulary used to describe designs, and introduce new best practices.

Dated Textbooks

As technology changes and software engineering evolves, the ability of texts to keep up with these changes is severely stressed. An instructor will find himself or herself utilizing three or four texts in order to properly cover a topic area. Books will seemingly contradict each other, only because they were published two years apart. Often, a book that is only three years old will contain many concepts that have been already superseded or renamed. Many excellent textbooks have not been updated to use current representations, such as UML2, for example.

This forces faculty continuously research new and updated prints. The faculty has to take into consideration student feedback on the usefulness of the recommended textbooks. Some new textbooks might be already dated at the time of their publication.

Operating System/Programming Language Biases

Few topics seem to generate as much debate as the selection of which operating system (OS) or programming language should be the lingua franca for course work. It seems that everyone has an opinion or a realistic need to learn one environment over another. The selection of one environment over the other has significant impacts on the tools available for use by the instructor, the knowledge that the instructor has to bring into the classroom, and the equipment that must be maintained. In our case, over the years we have migrated from UNIX platforms, to Windows, and to dual-boot machines that run both Windows and Linux. Most of the students are familiar with both operating systems, since different instruc-

tors favor one OS over the other. They appreciate the flexibility offered by the dual-boot machines available in our labs.

The programming language debate is a little more problematic than OS preferences. Many of the students at the graduate level have jobs in which they work in C++, Java, or C#. The students often insist that the programming language that they use in the workplace be utilized in their courses. The problem is that choice of programming language can significantly impact what is appropriate content for a course. Designing C++ programs utilizes different patterns than those used in designing Java programs, since C++ programs must necessarily and explicitly manage memory. Historically, the choice of programming language has been made largely based on inputs from the market and external program reviewers. For example, at the time this paper was written, most courses use Java, with the exception of the real-time systems course which still uses C and C++.

Continuous Development of Course Projects

Faculty, students, and industry have universally recognized the need for hands-on experience (Ellis, 2000). Without practical training, students and industry complain that the material will be too theoretical and that graduates would have trouble applying the theory to real world projects. This has led us to incorporate projects into the majority of courses taught in the program, while maintaining a balance between the theoretical and practical aspects of the courses. The type of projects has changed over the years: we have started with stand-alone systems, to continue with distributed, web-based, service-oriented systems.

The program culminates with a two semester practicum, where students work in groups on all phases of a real-world project, starting with requirements elicitation, design, implementation, and testing. Unlike the course offered at University of Southern California by B. Boehm (Boehm, 2006), in our practicum there are no lectures, because it is assumed that students have already covered all software engineering core courses in the curriculum. Students need to follow a well defined software process, producing all the necessary documentation that covers the product life-cycle. Although the process is not prescribed by the instructor, as in (Germain, 2003), most of the students follow a heavy–weight type of process, such as UPEDU (Robillard, 2001). The students practice teamwork and communication skills, while working on a large-scale project proposed by a real client. The clients are either from our campus, or companies from the area surrounding the university. They are asked to provide comments and evaluate the deliverables, in addition to the instructor. The type of project proposals we get from the industry partners points us to areas that need to be covered by the curriculum.

The course/term projects are administered at the beginning of the semester, and have a couple of milestones spread along the semester. The instructors check the documents and/or software applications delivered at each milestone and provide feedback to the students. The instructors provide the project statements. The members of the project teams are either established by the students, when they are not new to the program, or when no preferences are expressed, the instructor makes the choices. The teams have the authority to choose their leaders, and the role of each member.

The introduction of projects into a Software Engineering course encompasses its own set of difficulties. While a simple program for shuffling cards may suffice to teach students about algorithms and data structures in a programming course, software engineering has to deal with much larger problems in order to demonstrate the value and need for an engineering process. The

result is that projects have to be big, but not so big that they cannot be performed within the confines of the course. Because the project has to be big, it has to be structured such that the students can incrementally develop it as the course unfolds.

As the course content, technology and available tools change, the course projects need to change too. We have found that the size issue can sometimes be addressed by partially completing the project before presenting it to the students. This might require the development of a set of requirements before introducing a larger project into a software design course, providing some economic or financial analyses before introducing a project into a software project management course, or developing requirements and code before introducing a project into a testing course. In any case such a strategy requires that the instructor spend significant time doing the background work and documenting the results of that work so that the students can make good use of it as they proceed with the next steps. This way the students are encouraged to concentrate on tasks for a specific project that are unique to the course in which the project is being used.

Hilburn (2006), who wants to develop a comprehensive case study along with education material that can be used throughout the curriculum, proposes another alternative. This way, students will use the output of one course project work in subsequent courses, and will be able to better understand the connections between the topics taught in different courses. Again, this approach requires more work for faculty while making it more difficult to adapt courses to technological advances.

DIFFICULTIES ATTRACTING AND RETAINING FACULTY

Software Engineers, even in difficult economic times, are a highly sought after commodity. It is extremely challenging for any software engineering program to both attract and retain their faculty, in USA or around the globe (Grant, 2000). We noticed that the stability of the faculty makes a program more attractive to prospective students.

It is very difficult to attract appropriate faculty, as it has been observed by Glass (2003). In particular, faculty members usually are acquired from computer science backgrounds and/or from industrial practice. The problem with faculty from computer science backgrounds is that their backgrounds are in computer science rather than software engineering. The problem with acquiring faculty from industry is that often they do not have documented credentials (a PhD degree) and a documented trace of their scholarly work.

With the need to continuously update course content and curricula, to keep up or advance the state of the field, the load on a faculty member in software engineering tends to be significantly greater than in some other academic areas. Given that it is very difficult to hire faculty with the appropriate academic and industrial backgrounds, many of the hires are often non-tenure track. We are very fortunate to be positioned in a strong high tech industry area, with a steady supply of teachers with a very good industry experience, who are seeking to augment their income, are between jobs, or are retired.

A real solution for the administration is to provide competitive salaries and support consulting or research activities. This enables faculty to make up any shortfalls in salary and keep abreast of the industry needs and practices. With respect to this issue, MU offers faculty one day a week to spend on research or consulting activities. Also, MU has been successful in hiring excellent faculty with a PhD degree in areas other than computer science, with a strong industrial experience in software development.

We are aware that this solution might not be easy to implement at many universities, therefore

we are suggesting another venue for attracting and retaining faculty: the creation of a research center or institute on campus. This way faculty with complementary expertise can collaboratively work on interesting, complex projects and create rich opportunities for research and publications. This allows faculty to keep current with the state of research and practice, feed this information into a curriculum that is up-to-date (Boehm, 2000), reduce the teaching load, and build a cohesive faculty community. MU has created the Rapid Response Institute, where faculty from the SE department works together with faculty and students from around the campus on research and applications for Homeland Security.

DIVERSITY OF THE STUDENT BODY

In the 21-year history of the software engineering program at MU we have observed increasing diversity within the student population. The diversity spans several dimensions: educational background, employment status, educational goals and native language. The successful program must address all these dimensions of diversity.

Educational Backgrounds

Consistent with the origins of the program, many students in the graduate program achieved undergraduate degrees in computer science. These students have strong programming skills, but very seldom have the engineering discipline that emphasizes understanding the problem to be solved, or the process to be followed. These students tend to immediately start coding once they receive a problem to be solved. Students asked on more than one occasion why it was necessary to design a program when they could write one faster.

We also have a large population of students that are coming into the graduate program from other engineering and non-engineering disciplines. These students usually are much more accepting of engineering processes, but have relatively weak programming skills and minimal knowledge about how computers function. To accommodate them we have had to incorporate a set of preparatory courses to provide the programming skills and computer knowledge necessary to succeed in the program.

Our program has already started receiving a new group of students. These students have undergraduate degrees in software engineering and already have a good understanding of engineering practices balanced with programming skills. At this point, our program had to address increasingly more advanced software engineering topics that may be beyond the knowledge of the other two groups of students. A detailed discussion of this topic is deferred to in the Future Trends section.

Employment Status

The employment status of students has significant impact on the program. It affects how long students are in the program, the effort that they put into assignments, their willingness to accept course material, and when classes are offered. It should be noted that (with a few exceptions) students entering into the program full-time usually find work at the end of their first year and become part-time students. The majority of our student population attends school part-time with full-time employment in the software industry. Most of our classes are offered in the early evening to accommodate them.

The fact that the average student is employed full-time and attends classes part-time means that they may be in the program for as long as 8 years. In fact, the population of students is much more stable than the curriculum. Some students have

graduated on curriculums that have been replaced twice since they enrolled in the program.

Employment in the software industry has significant impact on the willingness of some students to accept the concepts taught in the classroom. These students have already acquired work habits that are not consistent with best practices. Students often state that they don't perform a particular engineering practice at work and that they don't see a need for it. Of course, many of these same students talk about how their projects at work tend to be chaotic. Other students report the difficulties they've encountered in trying to practice in their conservative organizations what they've learned in class. Either case tends to undermine the instructor in presenting new material in the classroom. Here is one of the situations where the instructor's industrial experience plays an important role in both selecting the material to be taught and in responding to student concerns regarding the usefulness of the topics learned in the real world.

Employed students also tend to focus on what they immediately need to succeed in today's workplace. There is often an insistence on learning a product (such as Oracle or Sybase) rather than the concepts (i.e., database principles). This emphasis on skill rather than knowledge runs counter to the goals of the program that are the development of software engineers who can lead their organizations into the future. We have included some of these products into our classrooms, but the main goals of the courses remain to teach the engineering principles of the field, which can be applied to a large number of products.

Students who are not employed in the industry have problems prioritizing the material being taught or placing it in the context of delivering a product. If they are required to know C++, they assume that all employers develop code in C++. They are often surprised when they get a job and discover that they will have to learn a new programming language. Students are occasionally concerned that courses cover many different methods and approaches to achieve a given goal rather than emphasize one method. They have to be taught to understand that the knowledge and skill they acquire in school will have to blend into whatever organization they join, and that they need to engage in a lifelong learning process that is inevitable in this dynamic field.

Educational Goals

It would be nice if all students entered the program with the desire and goal of becoming a software engineer and delivering a specific kind of product. However, the educational goals of the students range from wanting to know all about software and engineering, to the other extreme where they only want to get the credentials that will allow them to earn a higher salary. Our student body appears to be driven by a small number of educational goals, as we were able to derive from their application packages, advising sessions and an alumni survey. These are:

- Get the business and process knowledge that will allow them to manage software projects and people.
- Get the skills and knowledge that will allow them to be more productive in their chosen career.
- Start a career in which they can have a significant income
- Get a job in the software field that does not involve a lot of coding.

The major impact of these goals concerns the subject areas that interest the student. We have had to tailor our curriculum to respond to these different goals. We find a significant fraction of the students are very interested in the process, project management, and organizational man-

agement courses. Others find that the courses on requirements and software testing give them an entry point into a part of the software business that does not appear to require major coding efforts. Finally, the courses that emphasize specific types of software systems (real-time, information, and embedded systems) attract those students that are interested in gaining the particular knowledge and skills that will allow them to master their chosen field of work.

Communications Skills

There is significant diversity among our students in terms of their communication skills. However, communication skills are critical in software engineering, being considered as important as the technical skills (Teles, 2003; Lethbridge 2000). The average software engineering student will probably produce more documents and make more public presentations than the average English major. Communications have to be precise, unambiguous, complete, logically sound and well structured. Oral presentations have to convey complex information under time constraints. Students have to learn to gauge how much information is to be conveyed. This requires that they judge what their audience can be expected to know and what must be presented. Although typical undergraduate general education programs attempt to teach these skills most students who enter our graduate program require additional coaching and training in this area.

International students are often at a disadvantage due to the fact that English is their second language. This affects their writing ability where a weakness in vocabulary often prevents them from expressing themselves clearly and succinctly. It also undermines their confidence in public speaking due to concerns about their command of the language and fears that others will not understand them because of their accents. It can also severely limit their participation in class discussions.

International students are not the only ones with problems in communications. Many of the students, particularly those with computer science backgrounds, are not used to writing technical documents. While they may be good at writing code, they often have difficulties in expressing themselves succinctly in a written document.

The most direct approach to dealing with significant changes in the student population has been to adapt the curriculum and individual courses to meet the changing needs of our students. Employed students are encouraged to express their perspectives on the material so that their experiences can be shared with students that have yet not entered the field. In some classes, programming assignments can be written in Java or C++ depending upon the student's choice.

Another change has been the incorporation of more term papers into course work to allow students to get greater experience in writing. Papers are graded on technical content, structure, adherence to topic, and on the use of language. Corrections are suggested and students have a chance to resubmit corrected work. With respect to verbal communications, students are required to make oral presentations of their term projects. This way, until they reach the capstone project, students would have had the opportunity to exercise their communications skills several times. We have also observed significant progress in the communication skills and self-confidence of students when we created multicultural teams, and encouraged informal peer-mentoring. As one of our external program reviewers observed, the oral communications skills of students significantly improved when they were repeatedly videotaped, and discussed the strengths and weaknesses of their recorded presentations with the instructor.

GUIDANCE ON STARTING AND MANAGING PROGRAMS

Based on the experience described above in starting and managing Monmouth University's software engineering program we would offer the following advice to academic departments that are considering a similar program:

1. Conduct research to determine the most current curriculum recommendations from the IEEE, ACM and other sources.
2. Find out, by participating in national groups and committees that develop those recommendations, what likely future changes might take place.
3. Enlist the academic institution's industrial advisory boards to determine how the general recommendations need to be tailored to suit the needs of local industry. The partnership with the local industry will bring multiple benefits, such as a good source of real world projects for courses, student placement for summer internships, industry guest lecturers for courses or a research seminar.
4. Form a Task Force with professors from both SE and CS departments to make sure the two departments will not conflict each other. Also invite an external reviewer who can offer concrete guidance, based on personal experiences in building such a program at another university.
5. Recruit full-time faculty who are competent to teach the required variety of courses and who have industrial experience in applying software engineering techniques in real work environments. Don't expect this to be an easy task. You might need to manage the program initially with significant help from part-time faculty.
6. Expect that the curriculum will need to change with time to accommodate both changes in the discipline as well as changes in the needs of local employers.
7. Define a set of students learning outcomes that you will continuously monitor, and use the results to evaluate the need for improvements.
8. Periodically seek accreditation from a national board, or at least solicit a thorough review from an external evaluator, who is a prominent figure in the field. These efforts will ensure the quality of your curriculum.
9. If you intend to advertise your program to international students, make sure that you educate student's expectations regarding the research oriented or practical training oriented nature of the program.
10. If you intend to offer the program over multiple campuses, or on-line, you need to secure the equipment, technology and instructors qualified and willing to teach distance learning courses. Don't expect this to be an easy task, the instructor's effort to teach these courses might be considerably higher than teaching face-to-face courses.

FUTURE TRENDS

Having looked at the past, it is now appropriate to look to the future for our program. In particular, we recognized a need for another set of changes. The introduction of an undergraduate software engineering program had profound consequences on the graduate program, forcing severe changes in its curriculum. The redesigned curricula should allow the new graduates of the bachelor's degree in software engineering to have the opportunity of extending their knowledge and skills to new frontiers. In particular, we believe that while in

the undergraduate program students should focus on the application level software development, at the graduate level they should focus on the enterprise and global levels. Also, we expect these students to show originality in the application of their knowledge and pursue research to push the boundaries of knowledge in the SE area of their choice (similar to the UK program reported in (Edwards, 2003)).

With this respect, in 2007 we modified our graduate program, such that the students with a bachelor's degree in SE will be required to take 5 core courses, 6 elective courses and a two-semester thesis. Up to 9 credits of core courses can be waived if equivalent courses have been completed as part of the students' Bachelor of Science in software engineering program. This would make our SE graduate program similar in structure to masters programs in electrical engineering, mechanical engineering, etc. throughout the United States (see Figure 5).

In particular, we moved a bridge course into a core course: former SE501 (Mathematical Foundations) is combined with former SE561(Formal Methods) into a new and augmented core course, SE561(Mathematical Foundations of Software Engineering). This course will include mathematical methods that a software engineer needs to master, such as graph theory, formal languages, logic, sets theory, etc.

Former SE565 and SE570, the requirements and design courses, have been changed to cover techniques at the global and enterprise levels of software development. Former SE575, the software verification and validation course, has been changed to cover verification, validation and maintenance techniques and tools. Former SE580 course will cover the team software process (Humphrey,1999) due to the recognition that the graduates will need to work in teams for most of their careers, and the feedback received from graduates.

We have also made some changes in the bridge courses of our curriculum by adding two bridge-courses, SE510 (Object-oriented Analysis and Design) and SE515 (Disciplined Software Development). We strongly believe that all our students should know the basic analysis and design methods by the time they enter the graduate program. This would allow us to teach advanced methods for software analysis and design in the corresponding core courses (SE565 and SE570), instead of spending a considerable amount of time teaching basic knowledge. Also, we are strong believers of the engineering principles emphasized by a disciplined approach to developing programs, such as the Personal Software Process (PSP). We would like all our students to be familiar with these engineering techniques at the individual level, to be able to leverage them at the team level in the software process core course (SE580). This course will also introduce principles of agile software development.

We removed Operating Systems Concepts (CS505) from the curriculum since the material was covered in several other software engineering courses. We revised SE504 (Principles of Software Engineering) to focus on structured analysis and design methods while presenting the breadth of software engineering principles. This emphasis would allow us to focus on the modern object-oriented methods in the core analysis and design classes.

In the electives courses, we added a course on Secure Web Services Design (SE611) to develop a sequence of courses on security, together with SE610 (Software Systems Security). This sequence will cover both the theoretical and practical aspects of software systems security, given the ubiquity of security issues in today's systems.

Another future trend that we believe will induce major changes in the way we deliver our program will be determined by the increas-

Figure 5. 2007 curriculum

Core Courses(5 courses = 15 credits)
SE561 Mathematical Foundations of Software Engineering
SE565 Software Systems Requirements
SE570 Software System Design
SE575 Software Verification, Validation and Maintenance

Electives(4 courses = 12 credits)
Choose two pairs of guided electives from among
SE601 Outsourcing: Specifications and Strategies
and SE602 Technology assessment

SE620 Network Software Systems I
and SE621 Network Software Systems II

SE625 Information Systems Architecture
and SE626 Information Systems Engineering

SE630 Real-Time Software Analysis and Specifications
and SE631 Real-Time Software Design and Implementation

SE637 Wireless Communications
and SE638 Communications Systems

SE650 Software Project Management
and SE651 Software Organization Management

Electives(2 courses = 6 credits)
Choose two guided electives from among
SE601,SE602,SE603,SE605,SE610,SE611,SE615,SE620,SE625
SE626,SE630,SE631,SE637,SE638,SE650,SE651,SE652,SE660,
SE698,SE699,CS514,CS517,CS533,BM565

Thesis (6 credits)
Two semester Thesis:
SE690A: Thesis Research
and SE690B: Thesis Research

ingly mobile characteristic of the majority of our students, whom are working full-time and take the courses part-time. To allow them maximum flexibility, we might need to change our delivery mode to include more distance learning, maybe in the way The Open University in UK does (Quinn, 2006). At the moment we are experimenting with offering "hybrid courses" that are a combination of a traditional, face-to-face delivery, and distance learning that uses online curricular materials. This delivery mode saves students the travel time to campus, and also allows them to keep up to speed when they travel for business. Students are required to come to campus every other week, to meet with the instructor for a face-to-face class. If their grade falls below a certain threshold, they are required to come to class every week. This is a new approach for us, and we don't have enough data yet for a thorough evaluation.

Another issue, that is beyond the scope of this paper though, is the awareness of the influence that the licensing of software engineers shall have on the design of the curriculum. However, the directions and discussions that are taking place

with regard to licensing have to be followed so that appropriate changes can be implemented in the curriculum.

CONCLUSION

This chapter has presented the main problems and lessons learned from one of the oldest programs in software engineering in the USA. The evolution of the graduate curriculum over its 21 years of existence has been shown as an example for other colleges and universities considering the addition of a software engineering degree. We expect this evolution to continue in the future, as the SE field is a constant moving target.

We have argued that the continuous update of the course content has a special meaning in software engineering, due to the dynamics of the field. With this respect, we have shown the impact of the advances in the field on the textbooks used, the need for continuous reevaluation of the chosen programming language, operating system, or software tools used in class.

The chapter has shown the difficulties we have experienced in attracting and retaining the faculty over the years, due to the need of the new faculty to have both a record of scholarly accomplishments and industrial experience. The emphasis here is on the conjunction of these two requirements, which sets great restrictions on the pool of available candidates.

We explained how various issues related to the diversity of the student body influence the curriculum and course content. As such, the educational backgrounds, employment status, educational goals, and communications skills of the student body are challenges any software engineering program has to solve.

Based on our experience in dealing with these problems, we have offered some recommendations for those interested in starting a similar program, with an emphasis on the curriculum and course content issues that arise where an undergraduate and graduate program in software engineering coexist in the same department.

As a measure of success of our continuous efforts to improve, we have seen the program enrollment increasing steadily over the years. This is not a reason to rest, since the SE field will continue to evolve, and we will have to respond to new challenges.

REFERENCES

Ardis, M., & Ford, G. (1989). 1!989 SEI Report on Graduate Software Engineering Education (Tech. Rep. CMU/SEI-89-TR-21), Software Engineering Institute.

Ardis, M., & Ford,G. (1989). SEI Report on Graduate Software Engineering Education, *Proceedings of the Software Engineering Education Conference*, Springer-Verlag.

Boehm, B. (2006). Learning by Doing: Real-client Software Project Courses, *ASEE Tutorial 2006*, Retrieved from http://db-itm.shidler.hawaii.edu/cseet2006/Boehm%20ASEET.pdf .

Boehm, B., Kaiser, G., & Port, D. (2000). A Combined Curriculum Research and Curriculum Development Approach to Software Engineering Education, *Workshop on Developing Undergraduate Software engineering Programs*, *Proceedings of CSEE&T 2000*, 310-311

Bourque, P., & Dupuis, R. (2004). Guide to the Software Engineering Body of Knowledge – Final Version, SWEBOK, Feb. 2000, Retrieved from http://www.swebok.org/

Clough, G.W. (2005). Educating the Engineer of 2020: Adapting Engineering Education to the New Century. Washington, D.C.: National Academies Press, Retrieved from http://www.nap.edu.

Dart, P., Johnston, L., Schmidt, C., & Sonenberg, L. (1997). Developing an Accredited SE Program, *IEEE Software*, Nov/Dec, 66-70.

Duggins, S.L., & Thomas, B.B. (2002). An Historical Investigation of Graduate Software Engineering Curriculum, *Proceedings CSEE&T*, 78-87.

Ellis, H., McKim, J.C., & Younessi H. (2000). Issues Affecting Graduate and Postgraduate Software Engineering Curricula, Workshop on Developing Graduate and Postgraduate Software Engineering Courses*, Proceedings of CSEE&T 2000*, 190

Ford, G. (1991). 1991 SEI Report on Graduate Software Engineering Education, *Technical Report CMU/SEI-91-TR-2,* Software Engineering Institute, Carnegie Mellon University

Ford, G., Gibbs, N., & Tomayko, J. (1987). Software Engineering Education: An Interim Report from the Software Engineering Institute, Technical Report CMU/SEI-87-TR-8, Software Engineering Institute,

Garlan, D., Brown, A., Jackson, D., Tomayko, J., & Wing, J. (1995). The CMU Master of Software Engineering Core Curriculum, *Proceedings of CSEE&T 1995*, 65-86, Springer Verlag.

Germain, E., & Robillard, P. (2003). What Cognitive Activities are Performed in Student Projects?, *Proceedings of CSEE&T 2003*, 224-231

Glass, R. (2003). A Big Problem in Academic Software Engineering and a Potential Outside-the-Box Solution, *IEEE Software*, July/August,94-96.

Grant, D. (2000). Undergraduate Software Engineering Degrees in Australia, *Proceedings of CSEE&T 2000*, 308-309

Hiburn, T., Towhidnejad, M., Nangia, S., & Shen, L. (2006). A Case Study Project for Software Eductaion, *Proceedings FIE 2006*, M1F1-M1F5.

Humphrey, W. (1999). *Introduction to the Team Software Process*, Addison Wesley.

Humphrey, W. (2005). *A Discipline of Software Engineering*, Second Edition, Addison Wesley.

Lethbridge, T. (2000). What Knowledge is Important to a Software Professional?, *IEEE Computer*, 33(5), 44-50.

Lutz, M.J., Hilburn, T.B., Hislop, G., McCraken, M., & Sebern, M. (2003). The SWENET Project: bridging the gap from bodies of knowledge to curriculum development, *Proceedings FIE 2003*, vol.3, S3C-7.

McDonald, J. (2000). Teaching Software Project Management in Industrial and Academic Environments, *Proceedings of CSEE&T*, 151-160.

Naur, P., & Randall, B. (eds) (1968). Software Engineering: A report on a Conference Sponsored by the NATO Science Committee, NATO.

Powell, G., Diaz-Perrera, J., & Turner, D. (1997). Achieving Synergy in Collaborative Education. *IEEE Software*, Nov/Dec, 58-65.

Quinn, B., Barroca, L., Nuseibeh, B., Fernandez-Ramil, J., Rapanotti, L., Thomas, P., & Wermelinger, M. (2006). Learning Software Engineering at a Distance, IEEE Software, November/December, 36-43.

Robillard, P, Krutchen, P., & d'Astous, P. (2001) YOOPEEDOO (UPEDU): A Process for Teaching Software Process, *Proceedings of CSEE&T 2001,*18-26

Rosca D. (2000). An Active/Collaborative Approach in Teaching Requirements Engineering, *Proceedings of FIE'00*, T2C9-12

Rosca, D., Li, C., Moore, K., Stephan, M., & Weiner, S. (2001). PSP-EAT – Enhancing a Personal Software Process Course, *Proceedings of FIE'01*, T2D18.

Stryer, L. (2005). Bio2010: Transforming Undergraduate Education For Future Research Biologists Washington, D.C.: National Academies Press, Retrieved from http://www.nap.edu.

Teles, V.M., & Oliveira C. (2003). Reviewing the Curriculum of Software Engineering Undergraduate Courses to Incorporate Communication and Personal Skills Teaching, *Proceedings CSEET 2003*, 158-165.

Tucker, A (Editor) et al.(1991). Report of the ACM/IEEE-CS Joint Curriculum Task Force. Retrieved from http://www.acm/education/curr91/homepage.html

This work was previously published in Software Engineering: Effective Teaching and Learning Approaches and Practices, edited by H. Ellis, S. Demurjian, & J. Naveda, pp. 278-297, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 5.14
The Influence of Computer-Based In-Class Examination Security Software on Students' Attitudes and Examination Performance

Lori Baker-Eveleth
University of Idaho, USA

Daniel M. Eveleth
University of Idaho, USA

Michele O'Neill
University of Idaho, USA

Robert W. Stone
University of Idaho, USA

ABSTRACT

Expectancy theory is applied to the use of software that secures the testing environment of in-class examinations. This security software prohibits students from viewing unauthorized material during an examination. The empirical study collected 60 student questionnaire responses completed after using the security software. These responses were used to develop measures for a model derived from expectancy theory. Using structural equation modeling, the model was estimated twice for two different variables. These dependent variables were student attitude towards the security software and the student's examination grade. The empirical results indicated that student attitudes were positively impacted by self-efficacy, outcome expectancy, and the software's ease of use. However, student grade was not influenced by any measures in the model. It is concluded that the security software is neutral with regard to student performance, while there are manageable actions faculty can take to positively impact student attitude towards security software.

INTRODUCTION

Declining technology prices and a drive to innovate in the learning environment create an opportunity to incorporate computer technology into the classroom. An example of these types of opportunities is the use of laptop computers to facilitate learning and classroom activities. The mobility of laptops provides a number of teaching and learning advantages. One such advantage that has not been fully explored is using laptops to evaluate and assess student examination performance by having students complete their examinations using laptops.

When using laptops on examinations, one challenge for instructors is how to provide a secure examination environment. A secure environment restricts students from accessing notes on their laptop hard drives or the Internet as well as prohibiting communication with other students via e-mail and instant messaging. Another challenge for both faculty and students is how to make sure that the technology has no impact on a student's grade. If a student takes an examination in a traditional way, using paper and a pencil, there may be test anxiety, poor handwriting or writing hand cramps, but the testing method (i.e., paper and pencil), does not affect performance on the examination. Using technology to automate the process of the examination provides advantages (i.e., typed text and no poor handwriting and hand cramping) but adds an additional dilemma, that is, what happens if the technology fails during the examination or if an inadvertent key stroke by the student leads to the loss of typed text? Furthermore, the laptop could crash, the power could go out forcing battery backup, or the software application could fail. All these events could affect student performance and the student's grade on the examination. It is important to know if using computer technology for assessment has an effect on this important outcome.

Another advantage of using laptops for examinations is to better prepare students entering the workforce. An important skill needed by today's graduates is confidence in technology ability. Confidence in one's ability can make the difference in a student being hired quickly for a management position or struggling to find an entry-level position. Any positive experiences using technology in novel environments can help build students' confidence levels regarding technology and its use. In other terminology, these students' confidence and its role in many behavioral and affective outcomes is described by self-efficacy and outcome expectancy (Henry & Stone, 2001; Jenkins & Garvey, 2001). In general, self-efficacy is the individual's perception of possessing the requisite abilities to successfully perform a specific task (Bandura, 1977; 1982; 1986). Outcome expectancy is a companion to self-efficacy reflecting the individual's perception regarding the result or gain from successful completion of these tasks. In this context, the expectations of self-efficacy and outcome expectancy provide a well-established theory by which to understand the impacts of laptop-completed examinations on student performance and attitudes.

The research presented below focuses on how the use of computers by students to complete examinations in a secure environment impacts their performance and attitudes as predicted by a model based on expectancy theory. The remainder of the article is organized into sections to present the details of this examination of self-efficacy theory and its impact on student performance. First, a discussion of the theoretical model based on self-efficacy theory is presented. Second, the methodology used in the research is discussed. Finally, the empirical results are examined followed by conclusions and directions for future research.

LITERATURE REVIEW

The model used to guide the use of laptops on examinations and the corresponding use of security

software is rooted in expectancy or self-efficacy theory. The theory (Bandura, 1982; 1986) links an individual's cognitive state to a variety of affective and behavioral outcomes, along with perceptions of future outcomes (i.e., loss of control, low self-confidence, low achievement motivation) (Staples, Hulland, & Higgins, 1998). Self-efficacy theory has in the past been used to explain user reactions to information technologies (Meier, 1985; Bandura, 1986; Baronas & Louis, 1988; Martinko, Henry, & Zmud, 1996; Potosky, 2002; Hasan, 2003; Havelka, 2003). Recent research on computer self-efficacy investigated the influence of demographic predictors (e.g., academic major, gender, computer-related experience) on business students' self-efficacy (Havelka, 2003). Significant differences in self-efficacy ratings resulted for information systems and economics majors compared to management majors and those who had greater than five years' experience working with computers. Gender differences did not result in reported self-efficacy differences.

According to self-efficacy theory, expectations in large part determine affect and behavioral reactions in numerous situations. Bandura (1986) further separated expectations into two distinct types, self-efficacy and outcome expectancy. An individual's belief that he or she possesses the skills and abilities to successfully accomplish a specific task represents self-efficacy. Furthermore, an individual's persistence to learn a task impacts his or her perceptions of future outcomes, which in turn influences self-efficacy. Outcome expectancy, on the other hand, is an individual's belief that by accomplishing a task, a desired outcome is attained. Self-efficacy and outcome expectancy have separate impacts on behavior and affect. However, self-efficacy typically has a larger effect than outcome expectancy (Bandura, 1986) and generally self-efficacy has a direct impact on outcome expectancy (Stone & Henry, 2003).

The work of Bandura (1982; 1986) was adapted for application to the adoption and use of information technology. Based on self-efficacy theory, four constructs are proposed as directly impacting self-efficacy and outcome expectancy in this environment of students using security software and laptops to complete examinations. These variables are watching others use the security software, direct experiences using this software, faculty support in its use, and ease of the security software's use. The resulting modified model used in this research is displayed in Figure 1.

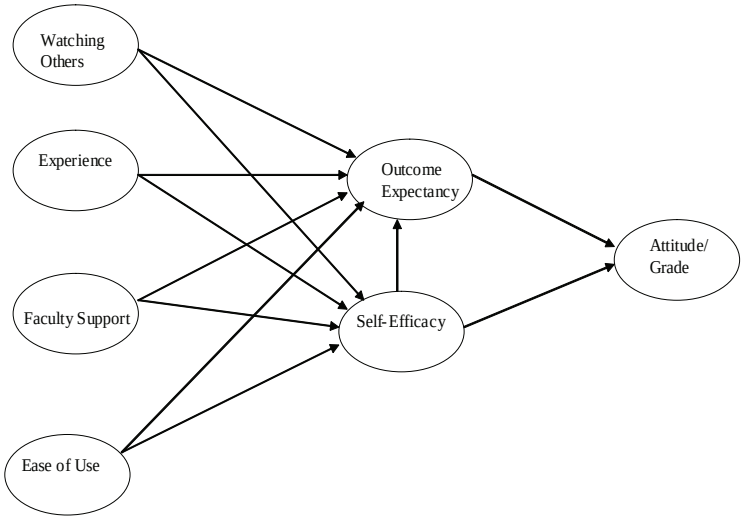
THE EMPIRICAL STUDY

A paper-and-pencil questionnaire was administered to students enrolled in what is commonly referred to in a business school as the 'Junior-level Business Core' or the 'Common Body of Knowledge' in order to collect the sample. This course is team taught by five faculty members. In two sections of the course, students completed an examination using their laptops and software that provided a secure testing environment. The security software allows students to use word processing documents and spreadsheet templates on the examination, but prevents students from performing other actions such as accessing hard drives, the Internet, and instant messaging, or e-mailing others in search of answers to examination questions. The examination was developed by the faculty in a manner that required students to review a spreadsheet template and respond to all examination questions by typing answers in a word processing document. This is because all the examination questions were essay questions.

Sample Data

A total of 107 students were enrolled in these two sections of the business core. There were 63 students enrolled in the morning section and 44 students in the afternoon section. The particular examination was administered during a common time to 98 students. Of the nine students who did not take the examination, five were excused due

Figure 1. The theoretical model



to illness and four had conflicts with the common time and were given the examination in a traditional hand-written format at another time and location.

The 98 students who used the security software to complete the examination were given the opportunity to complete the questionnaire five days following the experience, but prior to receiving their examination scores or grade. A verbal reminder to return the questionnaire was provided during class prior to the grade being released. No additional reminders were provided due to concerns over latency effect and attribution of examination performance; that is to say, no other reminders were provided due to the concern that too much time had elapsed between using the security software and the students responding to the questionnaire. In such a case, the students' responses might not reflect their immediate perceptions after using the software. Furthermore, if students received their examination grades before completing the questionnaire and if their grade was different than what they perceived as appropriate, they could attribute their unexpected performance, either better or worse, to the security software. Thus, limited in-class reminders to complete the questionnaire were made to hopefully obtain responses that best reflected students' true perceptions of the software. Furthermore, it was made clear that completing the questionnaire or not completing the questionnaire would have no impact on the student or their grade. Of the 98 students that took the examination with laptops, 62 students responded to the questionnaire, with 60 students fully completing the questionnaire, producing a 61% response rate.

Characteristics of the Respondents

The characteristics of the sample respondents are shown in Table 1. The average GPA of 3.06 appears high, but it is the case that all students enrolled in the course must have completed several qualifying courses and obtained a minimum GPA among these courses. These requirements prohibit some students, who tend to have lower GPAs, from enrolling in the course. The average age of the respondents was 22.2 years. Both genders were represented with 40% females and 60% males.

Table 1. Data characteristics

Characteristic	Sample	College Population	Test Statistic
			t-Statistic
Average GPA	3.06	2.98	1.25 [a]
Average Age (years)	22.2	21.6	1.68 [a]
			z-Statistic
Gender:			0.26 [b]
Female	40.00%	39.00%	
Male	60.00%	61.00%	
			Chi-Square Statistic
Major:			13.89 [c]
Marketing	28.33%	22.20%	
Information Systems	11.67%	12.60%	
Finance	11.67%	11.60%	
Management & Human Resources	16.67%	14.00%	
Production Operations Management	3.33%	5.60%	
Professional Golf Management	3.33%	0.70%	
Accounting	20.00%	16.20%	
Economics-Finance	5.00%	6.10%	
Other/Undeclared	N/A	9.00%	
Number of Observations	60	837	

[a] *t-Statistic = two-tailed*

[b] *z-Statistic = two-tailed*

[c] *df = 8*

The percentage of students in each major ranged from a high of 28.33% in Marketing to 3.33% in both Production Operations Management and the Professional Golf Management program.

Non-response Bias

For any research that depends on data collected using a questionnaire, non-response is a concern. To examine the possible presence of non-response bias, the sample characteristics were compared to the corresponding values at the college level. The logic of using the college demographics is that the students in the college form the population from which the sample was drawn. Any potential non-response bias would be reflected in differences in the demographics between the sample and the population. Table 1 also displays the demographic values for the college and the corresponding statistic testing the significance of the difference between the sample and the population values. All these tests were two-tailed and no meaningful differences between the sample and the population demographics were found at a 5% significance level. Based on the comparison of these demographic variables, it is concluded

Table 2. Items and measures reliabilities

Measures and Indicants	Cronbach's Alpha
Experience Using the Software	0.77
For each of the following factors, indicate the extent to which you agree or disagree that it helped your use of the software:	
Going through a short practice exercise with the software during class.	
Going through a sample exam during class.	
Watching Others Using the Software	0.90
Watching my teammates working on the software helps me understand how to use it.	
Watching my peers practice working on the software helps me successfully use it.	
Outcome Expectancy	0.85
I find it easy to recover from errors encountered while using the software.	
I find it easy to get the software to do what I want it to do.	
Helps me complete the exam faster.	
Ease of System Use	0.74
I often become confused when using the software.	
I make errors frequently when using the software.	
Attitudes Towards the Software	0.93
Completing exams using the software is the way I prefer to work on exams.	
I would like to use the software for all my exams.	
I wish all my classes would use the software on exams.	
Faculty Support	0.80
The faculty team:	
Encourages us to use the software.	
Seems to value the software.	
Self-Efficacy	0.89
I know enough to successfully use the software.	
I fully understand how to complete an exam using the software.	

that non-response bias does not present a problem for the sample.

MEASURES AND THEIR PSYCHOMETRIC PROPERTIES

The work of Stone and Henry (2003) was modified and adapted to develop the questionnaire items used to measure the constructs in the model. For all measures, students were asked the extent to which they agreed or disagreed with each of the items and were presented with the ordered answer choices of strongly disagree, disagree, neutral, agree, or strongly agree. The questionnaire was pre-tested by 11 students who had completed the course a year earlier and were familiar with the examination security software. Using the feedback from these students regarding the intent and readability of the questions, four items were deleted from the questionnaire.

Each measure used in the model and the questionnaire items forming them are shown in Table 2. Also shown in Table 2 are the Cronbach's

Alphas for each measure. These values ranged from a low of 0.74 for the measure ease of system use to a high of 0.93 for the measure attitudes towards the software. These values indicate that all the measures have acceptable levels of reliability and therefore satisfy composite reliability (Nunnally, 1978).

In order to examine the measures for the psychometric property of discriminant validity, a series of correlations were performed. Analyzing one questionnaire item at a time, the item was removed from its measure and correlated with its own modified measure and all other measures in the study. If the item satisfies discriminant validity, it correlates more highly with its own modified measure than with all other measures. The process was repeated for each questionnaire item in the model. This is the approach to evaluate discriminant validity proposed by Campbell and Fiske (1959). For this particular study, a total of 112 of these correlations were computed. Only one of these correlations violated discriminant validity. This violation was for an item measuring *Outcome Expectancy* that correlated more highly with a measure for Attitudes towards the Software. As a result, it is concluded that discriminant validity is satisfied (Campbell & Fiske, 1959). Based on the desirable composite reliability and these discriminant validity results, it is concluded that the measures used in the empirical study display satisfactory psychometric properties (Rainer & Harrison, 1993). As a result, we can be confident that the items and measures perform reasonably well in measuring the intended underlying constructs.

MODEL ANALYSIS

Because the model was relatively complex compared to the sample size, the questionnaire items were summed to form the measures of the model's constructs. The hypothesized paths among the measures were estimated using structural equations modeling as performed by covariance analysis of linear structural (CALIS) equations in PC SAS version 9.1. The estimation was done using maximum likelihood. Two models were estimated that differed only by the dependent variable. The model was estimated once using the dependent variable attitudes towards the software and once again using as the dependent variable student grade on the examination.

Summary statistics for the fit between the model and the data from the estimation of both models are reported in Table 3. The model using attitudes towards the software as the dependent variable had a goodness of fit index of 0.99 (adjusted for degrees of freedom it was 0.93). The root mean square residual was 0.02 while the chi-square statistic (four degrees of freedom) was 2.15 and insignificant at a 5% level. The normed chi-square statistic was 0.54. Bentler's comparative fit index was 1.00 while the incremental fit indexes ranged from 0.88 to 1.13. A couple of these values indicate an "over-fitting" of the model to the data, which is most likely due to the relatively complex model and small sample size.

For the model using student examination grade as the dependent variable, the goodness of fit index was 0.98 (adjusted for degrees of freedom it was 0.86). The root mean square residual was 0.04. The chi-square statistic (four degrees of freedom) was 4.03 and not statistically significant at a 5% level. The normed chi-square statistic was 1.01. Bentler's comparative fit index was 0.99 and the incremental fit indexes ranged from 0.48 to 0.99.

In summary, even though these statistics provided mixed findings regarding the goodness of fit between the model and the data, the values were sufficient to conclude the fit was acceptable (Hair, Anderson, Tatham,& Black, 1992). This acceptability judgment was based on the fact that more of these summary statistics meet the generally accepted cutoff values than not. Thus, the variations found in the data were consistent with those implied by the theoretical model, and

Table 3. Fit of the models

Statistic	Value
Goodness of fit index	0.99
Adjusted goodness of fit index	0.93
Root mean square residual	0.02
Chi-square (4 degrees of freedom)	2.15
Normed chi-square	0.54
Bentler's comparative fit index	1.00
Bentler & Bonett's non-normed fit index	1.13
Bentler & Bonett's normed fit index	0.98
Bollen non-normed fit index	0.88
Bollen normed fit index	1.02

Dependent variable: Attitudes toward software

Statistic	Value
Goodness of fit index	0.98
Adjusted goodness of fit index	0.86
Root mean square residual	0.04
Chi-square (4 degrees of freedom)	4.03
Normed chi-square	1.01
Bentler's comparative fit index	0.99
Bentler & Bonett's non-normed fit index	0.99
Bentler & Bonett's normed fit index	0.90
Bollen non-normed fit index	0.48
Bollen normed fit index	0.99

Dependent variable: Examination grade

it was concluded that the data "fit" the model reasonably well.

The estimated structural model using Attitudes towards the Software as the dependent variable is presented in Figure 2. Ease of system use had a meaningful impact on student attitudes through outcome expectancy. Self-efficacy had a direct impact on Attitudes towards the software as well as an impact through outcome expectancy. Results for the model using student grade as the dependent variable are shown in Figure 3. Neither self-efficacy nor outcome expectancy had meaningful impacts on the grade. However, ease of use and self-efficacy did have meaningful impacts on outcome expectancy.

DISCUSSION

The expectancy model was applied to students completing examinations in class using laptops loaded with software that provided a secure testing environment. The features of this security software were described earlier in the article. The intent of the research was to identify those actions impacting not only student attitudes towards the software, but also student performance on the examination as measured by grade.

The most interesting empirical result comes from comparing evidence from the two estimated models. Student attitudes towards the security software are positively impacted by self-efficacy and outcome expectancy, as well as ease of use via outcome expectancy. When the model was estimated using examination grade as the dependent variable, none of the other variables in the model had either a direct or indirect impact on this grade. The interpretation of these results is that the examination security software did not influence student examination performance as measured by grade and ease of use, self-efficacy, and outcome expectancy positively impacted student attitudes towards the software. The result is intuitively appealing because it indicates that students have positive attitudes toward the examination security software yet the software is neutral with respect to student examination performance.

CONCLUSION

The presented research examines an increasingly common situation in education, the use of computer-based examinations and the resulting need for software that secures the testing environment.

Figure 2. The model with student attitude as the dependent variable

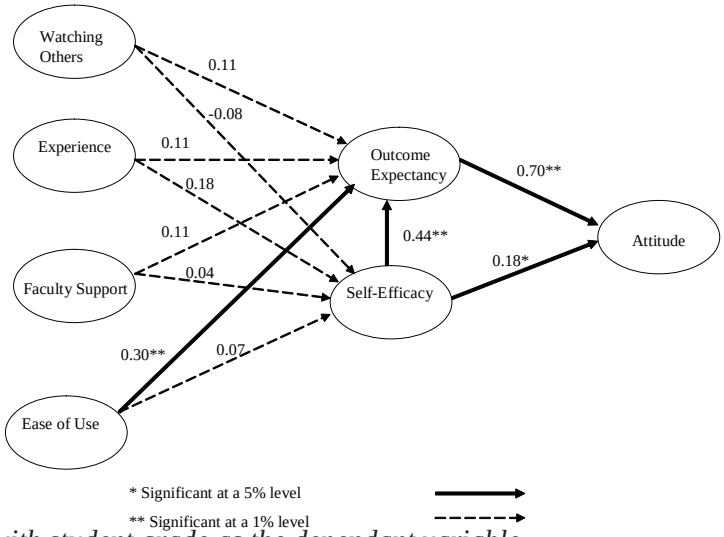
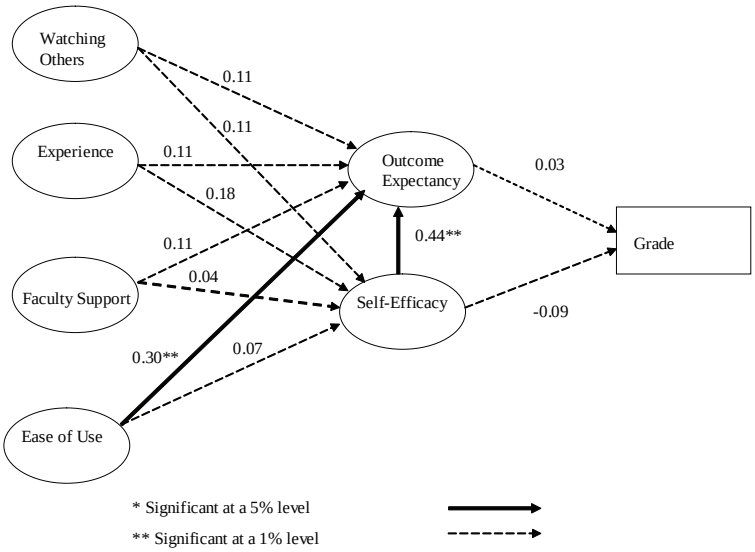


Figure 3. the model with student grade as the dependant variable



The empirical study shows student attitudes towards the software are positively influenced by several factors, yet student performance on the examination as measured by grade is not. The software providing a secure testing environment is neutral with regard to examination performance and yet student attitudes can be positively influenced. In a larger sense, these attitudes and the confidence from successfully using such software may translate into greater confidence regarding technology in general. As mentioned early, such skills may help these students secure an entry-level position upon graduation.

As more colleges and universities move toward mobile computing environments there will be additional opportunities to assess student performance using computer-based examinations. It is encouraging that student performance is not hindered by use of such software. Is this not what is desired from such software? The variables

positively impact student attitudes towards the software yet the software does not artificially help or hinder a student's performance. Future research in this area should look at other variables that may influence self-efficacy and outcome expectancy, as well as examine additional variables influencing student performance.

REFERENCES

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review, 84*(2), 191-215.

Bandura, A. (1982). Self-efficacy mechanism in human agency. *American Psychologist, 37,* 122-147.

Bandura, A. (1986). *Social foundation of thought and action: A social cognitive theory.* New Jersey: Prentice-Hall, Inc.

Baronas, A. & Louis, M. (1988). Restoring a sense of control during implementation: How user involvement leads to system acceptance. MIS Quarterly, *12*(1), 111-123.

Campbell, D. & Fiske, D. (1959). Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological Bulletin, 56,* 81-105.

Hair, J. Jr., Anderson, R., Tatham, R., & Black, W. (1992). *Multivariate data analysis with readings.* New York: MacMillan Publishing Company.

Hasan, B. (2003). The influence of specific computer experiences on computer self-efficacy beliefs. *Computers in Human Behavior, 19*(4), 443-450.

Havelka, D. (2003). Predicting software self-efficacy among business students: A preliminary assessment. *Journal of Information Systems Education, 14*(2), 145.

Henry, J. & Stone, R. (2001). The role of computer self-efficacy, outcome expectancy, and attribution theory in impacting computer system use. *Journal of International Information Management, 10*(1), 1-16.

Jenkins, A. & Garvey, R. (2001). Developing human potential at work. *Futures, 33,* 461-467.

Martinko, M., Henry, J., & Zmud, R. (1996). An attributional explanation of individual resistance to the introduction of information technologies in the workplace. *Behaviour & Information Technology, 15*(5), 313-330.

Meier, S. (1985). Computer aversion. *Computers in Human Behavior, 1(*2), 71-179.

Nunnally, J. (1978). Psychometric methods, 2nd ed. New York: McGraw-Hill.

Potosky, D. (2002). A field study of computer efficacy beliefs as an outcome of training: The role of computer playfulness, computer knowledge, and performance during training. *Computers in Human Behavior, 18*(3), 241-255.

Rainer, R., Jr. & Harrison, A. (1993). Toward development of the end user computing construct in a university setting. *Decision Sciences Journal, 24*(6), 1187-1202.

Staples, D., Hulland, J., & Higgins, C. (1998). A self-efficacy theory explanation for the management of remote workers in virtual organizations. *Journal of Computer-Mediated Communication, 3*(4), 1998.

Stone, R. & Henry, J. (2003). The roles of computer self-efficacy and outcome expectancy in influencing the computer end-users' organizational commitment. *Journal of End User Computing, 15*(1), 38-53.

This work was previously published in International Journal of Information and Communication Technology Education, Vol. 4, Issue 2, edited by L. Tomei, pp. 1-13, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.15
Integrated Software Testing Learning Environment for Training Senior-Level Computer Science Students

Daniel Bolanos
Universidad Autonoma de Madrid, Spain

Almudena Sierra
Universidad Rey Juan Carlos, Spain

ABSTRACT

Due to the increasingly important role of software testing in software quality assurance, during the last several years, the utilization of automated testing tools, and particularly those belonging to the xUnit family, has proven to be invaluable. However, as the number of resources available continues increasing, the complexity derived from the selection and integration of the most relevant software testing principles, techniques and tools into an adequate learning environment for training computer science students in software testing, increases too. In this chapter we introduce a experience of teaching Software Testing for a senior-level course. In the elaboration of the course a wide variety of testing techniques, methodologies and tools have been selected and seamlessly integrated. An evaluation of students performance during the three academic years that the course has been held show that students' attitudes changed with a high or at least a positive statistical significance.

INTRODUCTION

In this chapter we present a complete methodology for software testing training in the context of a laboratory course for senior-level computer science students. The intent of this work is to provide educators with a set of guidelines to effectively

instruct computer science students on software testing. The goal is not only to incorporate specific software testing skills into students' curricula, but also to prepare the student with skills for independent lifelong learning on the topic. The designed course spans the whole software testing lifecycle, and includes teaching recommendations to address students' common difficulties and misconceptions, as well as techniques to evaluate Students' performance for every stage.

During three academic years (2003-2006, note that results for the ongoing academic year are not currently available) we have developed and improved a software testing learning environment that has been used to train senior-level students in the Department of Computer Science of Universidad Autonoma de Madrid (Spain). In this environment, students are instructed about the elaboration of the test plan, test cases design, testing automation by means of specific tools, reporting and interpreting test results and maintenance related issues. All of these tasks are carried out over a complete pre-existent software system that has been specifically developed for this purpose.

To evaluate the effectiveness of the approach we have carried out attitudinal surveys to students during the three years that the course has been offered. These surveys provided us with inestimable information about students' progress and perception on several aspects of the course. This information was used to find out which elements of the course were perceived by students as most useful, most difficult or most personally rewarding; and, of course, to improve the learning environment along the academic years. We have found that, thanks to their immersion in this testing environment, students understood the crucial importance of software testing across the software lifecycle. Also, they incorporated a complete testing methodology and a broad set of software testing tools into their previous knowledge.

The chapter is divided into the following sections: a background section in which previous work on the topic is discussed and compared to the proposed approach, a description of the software testing learning environment including teaching recommendations and a description of the students' performance evaluation method, an evaluation of the effectiveness of the approach and a final section with the conclusions and future work.

BACKGROUND

Due to the increasingly important role of software testing in software quality assurance, during the last years, the use of testing frameworks that assist the developer during the testing process, and particularly the use of those belonging to the xUnit family, has proven to be invaluable. The production of high-quality and bug-free software products and solutions has gained a crucial importance in the software development industry, always focused to meet the needs of its increasingly more demanding end-users. In the last few years, many software testing techniques and methodologies have emerged to address these challenges, some of them influenced by agile (Beck, K. et al., 2001) and particularly by Extreme Programming (XP) (Beck, K., 2000). These techniques provide a wide set of principles, practices and recommendations for all the tasks involved in the software testing process, from test case design to automation of functional tests. In this context, an overwhelming number of testing frameworks and tools have been developed and are available (many of them under open-source licenses) with the purpose of aiding the developer in testing every particular system aspect written in any programming language imaginable.

However, as the number of resources and techniques available continues increasing and demonstrating new benefits, the complexity derived from the selection and integration of the most relevant software testing principles, techniques and tools into an adequate learning environ-

ment for training computer science students in software testing, increases too. Though several interesting experiences have been reported, to collecting and integrating all of these continuously evolving sources of knowledge and experience into a methodology to effectively teach software testing, remains an unresolved issue. As we will see later on in this section, many experiences of taking software testing to the classroom have been reported. They are focused in a number of testing related topic like for example extreme programming, unit testing or pair programming. However, it seems like there have not been any experience of collecting and integrating the most relevant and successful techniques into the same course.

There have been numerous experiences bringing Extreme Programming principles to the classroom (Astrachan, O. Duvall, R.C. & Wallingford, E. 2001; Edwards, S. 2003; Kaufmann, R. & Janzen, D. 2003; Melnik, G. & Maurer, F. 2002; Mugridge, R. 2003; Müller, M. & Hagner, O. 2002; Müller, M. & Tichy, W. 2001; Reichlmayr, T. 2003; Shukla, A. & Williams, L. 2002; Tinkham, A. & Kaner, C. 2005) as well as other less specific like (Collofello, J. & Vehathiri, K., 2005) and (Astrachan, O., Duvall, R.C., & Wallingford, E., 2001). For example, in (Shukla, A., & Williams, L., 2002) a complete report of an undergraduate course on software testing focused on Test-Driven Development (also known as TDD and considered one of the most important aspects of Extreme Programming) is presented. The course was held during three academic years and, despite positive results in terms of students performance, a main problem was identified. The problem lies in the counterintuitiveness of TDD due to the fact that, according to this technique test cases need to be written before the code to test. This problem is especially significant in graduate and nearly-graduate students (for whom the course presented in this chapter is intended) who have already become established in the traditional "write the code and then test it" software testing strategy. In general, in the vast majority of these experiences a special need for coaching and support for students has been detected due to the novelty of the topic and the large number of new concepts it involves. For this reason we decided to design an integrated learning environment in which students' progress is monitored through individualized tutoring during laboratory classes and the use of a centralized software repository where they store the work as they progressively complete it. In this respect the adoption of pair programming as the collaborative paradigm for the course has brought us the possibility of taking advantage of the benefits it provides to students when facing radically new software development related concepts and scenarios.

Pair programming is a software development model at the core of XP and is a kind of "collaborative programming". It consists of two programmers (two students), working side-by-side at one computer collaborating on the same design, algorithm, code or test. One person is the "driver", i.e. has control of the pencil/mouse/keyboard and is writing the design or code. The other person, the "observer," continuously and actively examines the work of the driver identifying tactical and strategic deficiencies in it (Williams, L., Kessler, R. A., Cunningham, W., & Jeffries, R. 2000). Despite cases of study (Müller, M., & Tichy, W. 2001) where pair programming has been shown to suffer from some waste of time and from an unclear division of work, we have chosen pair programming as the collaborative model during the laboratory course due to the following reasons:

- Pair pressure: pair programmers put pressure on each other. This is a form of positive pressure that leads students to keep each other focused and on-task (Williams, L. A., & Kessler, R. R. 2000).

- Pair programming has been shown to be beneficial independent of the developers' experience (Cockburn, A., & Williams, L. 2001). Note that our students do not have experience in formal software testing.
- Pair programming improves the success and morale of the students and increases satisfaction in the process (McDowell, C., Werner, L., Bullock, H., & Fernald, J. 2003).
- Pair programming increases confidence in the programming solutions
- Students are much less reliant on the teaching staff. When one partner doesn't know/ understand something the other almost always does, therefore the teaching workload is reduced and lab consultation hours are very calm (McDowell, C., Werner, L., Bullock, H., & Fernald, J. 2002; Williams, L., & Kessler, R. 2000).Pair programming is much more productive when developers face unfamiliar problems than when facing familiar ones (Lui, K.M., & Chan, K. C.C. 2003). This is the case we are considering since students have no previous knowledge about the software system to test, nor experience using the testing tools introduced in the course.

THE SOFTWARE TESTING LEARNING ENVIRONMENT

The Course

The software testing course has been held during the second semester of the last three academic years (2003-2006) as a laboratory course in the senior-level Software Engineering subject at the Department of Computer Science of Universidad Autonoma de Madrid (Spain). At the beginning of the course students have intermediate Java programming skills and more than 100 hours of theoretical-practical software engineering training plus specific theoretical instruction in software testing fundamentals. This instruction comprised basically the following topics:

- Test design techniques: black box and white box.
- Integration strategies in structured programming languages: top-down, bottom-up, and sandwich.
- Integration strategies in object oriented programming languages.
- Test cases design.
- Testing across the software development lifecycle: unit, integration, system, acceptance, and regression testing
- Risk management during the test process.
- Test plan document elaboration guidelines.

Table 1 summarizes the most relevant features of the course.

The first day of the course students are grouped in pairs and informed about the work to do:

- Test plan documentation: scope, description of the integration strategy and techniques selected, assignment of responsibilities and resources, schedule, milestones, risk management, completion criteria, etc.
- Test development: test procedures, test scenarios, test cases and test source code.
- Test execution: execution of the software following the plans and reporting of failures and errors detected.
- Test reporting: final conclusions about the results obtained from the executed tests.

During the explanation teachers emphasize aspects related with the testing automation level, code coverage, test cases design and maintenance. Finally students are informed about the course evaluation procedures.

System to be Tested

A complete system has been developed by teachers with the sole purpose of being tested. The main advantage of this is that students have the same starting point what makes students' performance evaluation more straightforward. Due to the strict time constraints of the course as well as the broad software developing experience students acquired in previous years, we have not seen the necessity of spending time instructing students on development issues.

The developed system presents a very interesting set of features that makes the testing process very interesting from an educational point of view: multithreading, HTTP interface, file input/output, private methods, exception handling, XML documents generation and parsing, external configuration, etc. The system consists of 7 Java classes and about 1700 lines of code. However, no more than 200 lines shared out between a few selected methods are used for testing purposes. In order to delimit the range of results that can be potentially obtained from the testing process, as well as facilitating students' performance evaluation and making the testing process more rewarding, several failures affecting different parts of the system have been deliberately introduced. These failures have been carefully selected with the intention of being detected using different testing techniques and strategies: black box, white box (grey box), unit testing, integration testing, functional testing, etc. Note that for obvious reasons the different failures introduced vary each academic year.

The system to be tested is named Road Information Server (RIS) and its aim is to serve XML documents via HTTP containing information about roads: traffic flow, presence of accidents in the road, weather forecast, etc. This information is taken by this module from the output of a hypothetical system named Road Observation and Information Providing System (ROIPS) from a data file (note that the format of this file is the only thing students need to know about the ROIP system) that acts as an interface between both systems. The RIS system is continuously reading the data from that file and generating XML documents containing the information requested via HTTP (GET method) by the clients. Since the information is published using HTTP the simplest way to interact with the system is from a conventional browser (this feature enables students to easily interact with the system). Figure 1 shows the system to be tested and its environment.

Table 1. Summary of course details

Number of students	150 students divided into 5 groups with an average of 15 pairs of students and a dedicated teacher per group.
Qualification required	Last year undergraduate computer science students.
Programming language	Java JDK 1.5
Testing tools	JUnit 4.0, JFunc, HttpUnit, XMLUnit, JTestCase, JUnitReport, JUnitAddOns, and others.
Software configuration management tools	SVN 1.3 + TortoiseSVN
Software execution and deployment	Ant 1.6.5
Evaluation procedure	• Oral presentation. • Formal written report including the Test Plan, test cases design and test execution results and interpretation. • Software generated quality and completeness (only the software present in the repository is evaluated) • Practical examination.
Duration	8 Weeks

Elaboration of the Test Plan

Once the students understand the goals of the course and get familiarized with the system to test, they must start elaborating the test plan. This document is required to be formatted as a technical report, this point is the special interest because students are very close to finish their degree and need to be prepared for dealing with the document formatting standards used in the software development industry. This document must be realistic and include a schedule and milestones adjusted to the course length.

Unit Testing

Unit testing is one of the core practices of XP and consists of taking each class of an object oriented software system and testing it in isolation. Students are encouraged to select a bottom-up testing approach, i.e. testing the classes of the system first and then testing the sum of its classes. On such an approach, integration testing becomes much easier. Teachers also encourage students to put special emphasis on unit testing due to the following reasons:

- Unit testing implicitly involves a sort of documentation that provides students with a better understanding of modules, requirements and API's.
- Good unit tests are fundamental when doing regression tests.

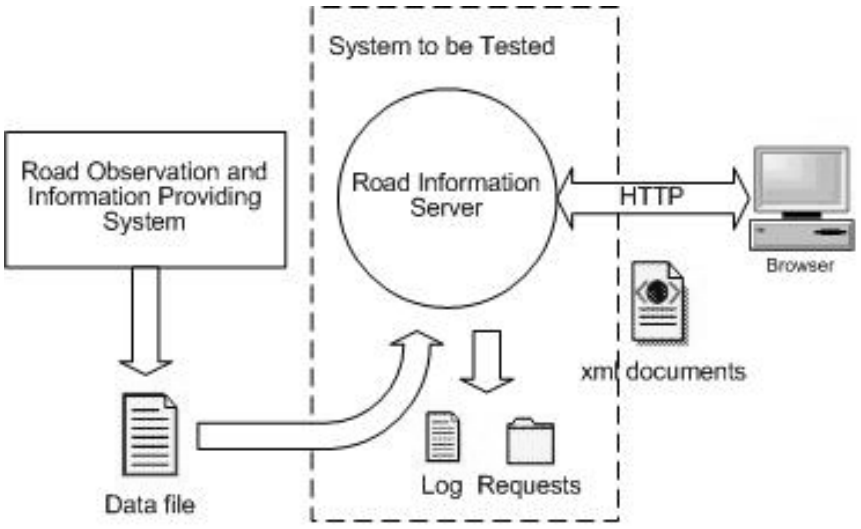
Since Java is the programming language selected for the course, the tool selected to assist the unit testing process can't be other than JUnit. Many issues concerning test cases design and the right way to test an object in isolation need to be covered for an in-depth understanding of unit testing. In the following subsections we describe these issues in detail, providing some teaching recommendations obtained from our experience holding the course. However, note that some aspects, like test cases design, are not uniquely correlated with unit testing.

Test Cases Design

The first step when doing unit testing is to design the test cases, black box and white box techniques are both suitable for this purpose. Black box consists of testing whether the output of a function or method, given certain inputs, conforms to its functional specification. White box consists of analyzing the source code in order to guide the selection of test data. In this respect, students need to have good enough Java programming skills to throughfully understand the execution flow present in the methods' source code. It is important to balance the pairs when creating them at the beginning of the course; students with less Java programming skills must be paired with the more experienced ones. Test data must be appropriately selected to achieve an adequate coverage over the code to test. Students have to decide which coverage (statement, edge, branch or path) to use when testing each method and justify the decisions made. Also, students have to create flow graphs for each method and depict on them special situations derived from exception handling when it is the case. Note that only 6 methods of the whole system are selected to be tested, so the workload is assumable.

We have encountered difficulties among students to understand and appropriately set-up the context in which a method for a given test case must be called. There is a trend to conceive a method as an execution entity which results are only determined by the input parameters regardless the context in which the method is invoked. This problem is especially notorious when designing test cases under the white-box perspective. For this reason we have selected some methods which results are strongly influenced by events like the presence of a file in the file-system or the inner-

Figure 1. Diagram of the system to be tested and its environment



state of the object in which the method is defined. Another important issue is to make students take into account all the factors involved in setting-up the method invocation context and to check all the observable results of its execution.

Testing in Isolation

Maybe the most difficult aspect when doing unit testing is to completely isolate a class from its collaborative classes. Usually an object makes use of other objects to carry out certain tasks beyond its own functionality. Obviously, the execution results (and so that the test results) of methods belonging to that object are going to be strongly determined by the inner-state of the object. Usually it is very difficult to set up domain state on such a way that it exposes the features to be tested. Even if we can do it, the written test will be probably very complex and difficult to understand and maintain. We can avoid these problems using Mock Objects (Mackinnon, T., Freeman, S., & Craig, P. 2000) that are a substitute implementation to emulate or instrument other domain code (in particular the collaborative classes). They should be simpler than the real code, not duplicate its implementation, and allow the developer to set up private state to aid in testing. Mock Objects are very useful, but to create them by hand may be tedious, therefore, students use a tool named JMock. JMock automatically generates the mock classes' source code from the original classes and presents a very intuitive interface with a very plain learning curve. In addition to the generation of the Mock Objects, a preliminary refactoring process is typically required, consisting in creating the factory methods in which the original objects will be replaced by mock objects. In the Source code listing 1, it is shown an example of a factory method that instantiates a collaborative class.

```
public class TargetClass() {
    protected CollabClass factory-
Method(){
    return new CollabClass();
   }
   ...
}
```

Source Code Listing 1

Following this procedure, it is possible to test objects that inherit from the target class and over-

ride the factory method to replace the instantiation of the collaborative object with the instantiation of the mock one. This can be seen in the Source code listing 2.

```
   public void testTargetClassMethod {
   // instantiation using mock objects
   TargetClass targetInstance = new Tar-
getClass() {
     protected CollabClass
      factoryMethod {
        return new CollabClass();
      }
     }
     // test something
   }
```

Source Code Listing 2

We have found that is very important to carefully select the code examples to which students must apply the Mock Objects technique so they can see a tangible benefit derived from its use. This way, applying Mock Objects becomes a very rewarding task rather than a nuisance. Otherwise they tend to consider the solution too complex in comparison to the problem to solve and it discourages them.

Testing Private Methods

Some TDD purists suggest that principles of encapsulation should never be violated for testing an object. Testing private methods (note that in the Java language as well as in other object oriented languages there are several access modifiers. In this respect, the qualifier private must not be interpreted literally but as "not belonging to the public interface of the class") means that you have to change your tests every time you change your private methods, and this becomes a barrier to refactoring and agile development. The reason is that, typically, private methods contain implementation details of the objects and therefore are more prone to suffer changes during the software maintenance process. We can consider, in the context of white-box testing technique, that a private method is implicitly tested by means of testing the public methods that use it. However, sometimes it is not easy to obtain an acceptable coverage following this strategy. In these cases we may need to test private methods directly, so we include such an exercise in the laboratory course. The problem here is that private methods can't be called outside the class where they are defined and obviously the test code can't belong to the class to test. The best solution is to by-pass the Java Virtual Machine (JVM) encapsulation mechanism by using the Java Reflection API. This can be done using the classes included in the `java.lang.reflect` package or by means of the JUnit-addons library (available under an open source license). As can be seen in the Source code listing 3, calling a private method with the later is straightforward:

```
   SomeClass returnValue = (SomeClass)Pr
ivateAccessor.invoke(
     instanceToTest,
     "methodToTest",
     new Class[]{ Class1, Class2},
     new Object[]{ param1, param2});
```

Source Code Listing 3

As a laboratory exercise, some private methods are selected; students must decide which of them should be tested and include the observed advantages and disadvantages of the decision taken in the documentation produced.

Testing Exceptions

Exceptions are a mechanism to handle unexpected or atypical situations during the execution of a program. Exception management code is responsible for the detection and handling of system conditions that could potentially lead to failure. As any other part of a software system,

they must be tested. However this is probably one of the aspects of an object oriented programming language, which testing procedure has never been covered in detail in the available bibliography. There are a few recommendations on the topic; even frameworks like JUnit provide helper tools. However, we have observed a lack of an in-deep analysis in which students can rely to successfully proceed in most of the possible scenarios.

When testing exceptions, students use to consider them as an `if-else` block of code, where the `if` corresponds to the `try` sentence and the `else` corresponds to the `catch` sentence. This way, the testing procedure would consist in defining two test cases, one for each possible execution path. Nevertheless, there are a fair number of non trivial questions that arise among students when taking this procedure to practice. Should all the exceptions be tested following the same procedure? Should all the potentially thrown exceptions be tested? If not, what is the criteria to decide which of them should not? In the rest of this section we will try to shed some light on these questions.

The goal is to verify that exceptions are generated only when it's due, following this consideration it makes sense to classify them as expected or unexpected. Note that this classification does not attend to the exception itself but to the nature of the test cases designed for testing it.

Expected Exceptions

Expected exceptions refer to test cases in which the method-under-test execution context is set-up so an exception must be thrown. Testing them consists of invoking the throwing method with "exceptional" data and checking that the exception is actually thrown via an assertion. Testing this kind of exceptions can be done in JUnit 4.x using the annotation `@Test(expected=ExpectedException.class)` when defining the test method. However, it presents a clear shortcoming, checking that the right exception has been produced is up to the framework and no extra verifications over the exception object itself can be done since it is not available in the test method. For simplicity and generality the procedure shown in the Source code listing 4 has been proposed to students:

```
public void testSomeMethod () {
  try{
            instanceToTest.
methodToTest(params);
   fail("An exception was expected");
  } catch (ExpectedException e){
    // Execution control must reach
here
  }
}
```

Source Code Listing 4

With these code sentences we ensure that a failure will occur if the exception `ExpectedException` is not raised when invoking the method to test with the adequate parameters. We have observed among students a common misconception of expected exceptions. Sometimes, they include some test cases in which the concept of expected exceptions is extended to "testing the Java platform". For example, test cases are written which result in a method invoked over a non initialized object that produces a `NullPointerException` raised by the JVM. This kind of test cases doesn't make any sense because testing the JVM is obviously out of the scope of the test plan.

Unexpected Exceptions

Unexpected exceptions correspond to those unpredicted situations for which the system can not suggest any solution. This kind of exceptions is easy to test since JUnit automatically catches exceptions thrown by test methods and report

them as errors (note the non trivial distinction between errors and failures in JUnit). While for unchecked exceptions (those who inherit from `RuntimeException`) nothing needs to be done, checked exceptions have to be declared in the throws clause of the test method definition.

```
   public void testSomeMethod () throws
SomeCheckedException {
     // test something
   }
```

Source Code Listing 5

Improving Maintainability

Nowadays, most of the activity and economic benefits of software enterprises come from maintenance related tasks. In fact, commonly in the vast majority of software projects, the maintenance life-cycle is much longer than the development one and so is the volume of resources dedicated to it. The interesting point here is that the larger the number of resources needed, the larger the potential for cost-effectiveness improvement and so must be the effort in teaching good practices on this topic.

During a maintenance stage in which the production code is being altered, regression tests need to be done with a very high periodicity and have to be as much automated as possible so they can be ran at a reasonable cost in resources. For this reason, it is necessary to train students in good testing practices that guarantee the production of not only maintainable test software but test software with a highly automated that can be effectively used in regression. In addition to some general recommendations, like minimizing the coupling between test code and production code and using auto-deployment scripts, students are trained in the use of an open source tool named JTestCase. This tool is very helpful assisting in the test cases design and execution tasks; it is basically a JUnit extension library that allows the test cases data to be separated from the test cases source code. This separation is provided by using XML data files to store test cases data in a very structured and readable fashion. To enhance maintainability even further, different XML files must be used to store test data belonging to different classes. JTestCase also provides the API methods required to load this data into memory from the test code during the testing process. The main advantages that led us to recommend students the use of this library are two:

- It is possible to enlarge the test cases data set with only adding a new test case description to the XML files, and without modify and having to recompile the test source code.
- Developers who design test cases data sets don't need to know about the source code of the methods to test. Therefore a clear separation between the test cases design and execution roles is established.

Nevertheless, this library also presents some drawbacks we needed to deal with when designing the laboratory exercises in order not to increase excessively their complexity. For example, storing the parameters data of the methods to test in the XML files when they are instances of complex data types or user defined classes, may result in a very complex and tedious task (because they are not directly supported by the syntax JTestCase provides). Although this problem may be overcome using the JICE library, we considered it does not worth the time students spend to learn a new tool.

Another recommendation we do is to use the XML documents generated as part of the test cases design documentation (XML files are readable by both humans and machines) and therefore avoid duplicated information that is always hard to maintain.

Reporting and Interpreting Test Results

Once the test cases execution has been carried out using the corresponding Ant script, a fair amount of information summarizing the errors and failures detected is generated. The correct interpretation and understanding of this information is a key issue to locate and solve adequately the software defects found during the testing process. JUnit includes support for the presentation of test cases execution results in textual (standard output in the command window) or graphical form. However, in real applications for which thousands of test cases are typically developed, these methods of presenting the information are unreadable and impractical. To cope with this problem we have instructed students in the use of the JUnitReport tool, which allows the generation of hypertext browsable documents in HTML format containing the execution results for every particular executed test case. This tool is able to merge the individual XML files generated by the <junit> Ant task, and apply a stylesheet on the final document. JUnitReport is provided with the Ant release as an additional task but installation of external third party libraries is required. One important thing to point out is that both <junit> and <junitreport> tasks must be written in different targets inside the Ant script so the test case execution and results reporting tasks are not interdependent.

Another fundamental topic that must be covered is the correct interpretation of the obtained results. Usually, JUnit makes a distinction between errors and failures, however, this distinction is artificial, unuseful and usually a source of misunderstandings among students. This distinction does not provide clear information about the source of the software defects found. While failures relate to assertion methods that have not been satisfied, i.e. defects in the production code, errors reflect unanticipated situations that occurred during the test cases execution and could be caused by both defects in the production code or in the test code. This issue must be covered at the beginning of the course when the JUnit tool is introduced.

Integration Testing

Integration tests are centered on the collaboration of classes in a system. Once the different classes have shown to work well in isolation, is necessary to verify that they also work well when combined. When doing unit testing over a target class, students do a little refactoring process to replace domain objects with mock objects through the use of factory methods. After that, mock objects must be replaced progressively by the original ones. This can be done straightforward using the approach presented in (Wick, M., Stevenson, D., & Wagner, P. 2008). Students have to replace the factory method of the original target class with a new factory method that returns the actual object with which students wish to integrate. Note that this approach allows a step-by-step integration, i.e. if we replace factory methods one by one, we are adding the original classes to the integration test one-by-one. In comparison with unit tests, integration tests are more difficult to implement due to the complexity of setting up the domain in the right state to test a specific behavior. In integration tests lots of objects are involved while in unit tests only a few mock objects, plus the target object, are involved. Moreover, mock objects state is very easy to set up comparatively. Due to these difficulties, we have found that students need extra support and instruction to make integration testing successfully.

Functional Testing

The final step is to make functional tests over the system as a whole. For this purpose students are provided with a brief Software Requirements Document in which, for example, the syntax of the HTTP requests served by the system and the format of the XML documents returned are

described. The goal is to make automated tests to verify that the system behavior meets the software requirements. Making functional tests from scratch over a distributed application with the only help of JUnit (note that despite its name JUnit is not exclusively attached to unit testing.) is a hard task. To cope with this difficulty we have introduced in the learning environment two interesting JUnit extension libraries (these two libraries as well as all the tools included in the learning environment described in this chapter are free-available open source tools) that facilitate this work: HttpUnit and XMLUnit. Note that despite their names, XMLUnit and HttpUnit are not unit testing tools but functional testing tools; nevertheless the "unit" prefix is an easy way to make these tools easily recognizable as belonging to the JUnit family. In one hand HttpUnit simplifies the interaction with a Web application by hiding all the HTTP protocol details from the developer. This tool basically emulates the functionality of a Web browser allowing the test code to navigate a Web application and retrieve its contents as a user would do by clicking links and reviewing documents using a conventional Web browser and the mouse. Once the test code is able to retrieve documents from the system, the next step is to validate the contents and structure of the documents retrieved to ensure they follow the specification contained in the Software Requirements Document. For this purpose, HttpUnit can be used in combination of XmlUnit. While the former is able to parse and validate the contents of HTML documents (like the title of the page, tables and forms present in it and even the correctness of the script code) the later is able to do XML documents validation.

Another interesting point when making functional tests is the possibility of allowing multiple failures, i.e. to allow more than one assert method to fail inside the same test method. JUnit typically stops the execution of a test method and continues with the execution of the next one when the first failure occurs. While this is convenient in the particular case of unit testing, in which after a failure happens the state of the object under test is potentially unpredictable, in functional tests is common to design a test case so it carries out a set of higher level operations that are often uncorrelated. In these cases, one failure may not affect the normal execution of the following operations and, since functional testing is usually a very time-consuming task, to be able to continue the testing process can save a lot of execution time. There is a specifically designed tool to overcome this drawback of JUnit when applied to functional tests, its name is JFunc and was also incorporated to the testing environment.

In the following points we summarize some interesting issues we have observed during the three academic years the course have been held.

- The process of incorporating functional testing tools to the course involves a relatively long learning curve if is not accompanied by the adequate examples and instruction.
- Once the tools are effectively applied to the functional testing process, students realize the simplicity of the test code produced and the extensibility and generality of the solution. After an initial guided research effort followed by a posterior independent research effort, they incorporate to their curriculum a set of state-of-the-art functional testing tools that clearly improve the quality and the level of automation of the tests, as well as are very helpful in regression.
- Sometimes students need extra support to distinguish between the aspects of a Web document that must be tested and those that must not. While the contents and organization of the information contained are important, aspects like presentation and Web design elements are obviously out of the scope.
- The system to test produces dynamic documents, this is an interesting point because some dynamic contents we deliberately in-

cluded, like time-stamps or auto-increment values, are by nature nearly impossible to test. In these cases, students are instructed to eliminate the validation of those values from the overall contents validation process.

Taking Advantage of Software Management Configuration Tools (SMC Tools)

Nowadays SMC tools are essential to track the evolution of the software under development and also represent the basic support for the collaborative work model of every software development team. For this reason, and in order to make the working environment as real as possible, we have considered a key issue to incorporate the use of a repository along the course. The version control system selected is SVN Subversion v1.3 (Subversion, 2000). This tool was originally created to replace CVS (Concurrent Version System) and presents some advantages over it, among them, its usage simplicity. Students interact with SVN by means of a Windows client named TortoiseSVN, which is integrated in the Windows Explorer contextual menu. The repository can also be accessed for reading purposes through a standard WEB browser using Apache authentication. Working with the repository using TortoiseSVN is a very easy task and only a few commands (import, checkout, commit …) and a basic knowledge about the work-cycle is necessary for students to get started. Each pair of students has a folder in the repository and a login/password to access it. The first day of the course students import the baseline software system to the corresponding work folder in the repository. At the end of every day in the course or after a major change has been made over the software contained in the local working folder, students are required to commit the changes to their personal folder in the repository. One common problem we have found is that some students can't clearly differentiate between software elements that must be stored in the repository (only those that evolve across the software life-cycle and can't be generated from others, as is the case of a .class file generated from a .java file) and those that must not. This concept is important because making a clear distinction between both kinds of elements prevents filling the repository with unuseful and redundant content and saves time in the interaction with it. For this purpose students are encouraged to define a "clean" task in the Ant script that allows deleting the compilation process results (binary files like .class and .jar) before committing to the repository.

The repository is also, indirectly, an excellent mechanism for teachers to track students' progress and detect misconceptions in the early stages of the course, when these problems are more likely to happen and easier to deal with. We will cover this topic in more detail in the next point.

Students' Performance Evaluation

Students' performance evaluation along the course is based on the following:

- Oral presentations in which each student explains and defends the decisions made and justify the obtained results. A key aspect is the adequate defense of the testing process completion criteria and the testing techniques and strategies selected. Also students are required to make suggestion about how to improve the learning environment and how other parts of the system that remain out of the scope of the test plan could be tested.
- A formal written report including the Test Plan, test cases design as well as test execution results and interpretation: The goal is to get students used to write formal documents as close as possible to those used in real-world software companies.
- Software generated quality and completeness: At the end of the course, the test software contained in the students' folder in the repository is examined and evaluated in terms of readability, completeness, level

of automation achieved, coverage over the production code, maintainability, etc. An existing tool designed for measuring the coverage achieved over the production code, which name is Cobertura (Cobertura, 2005), has been utilized for automatically measuring the coverage of students' generated test code and to compare it to the target coverage they described in the Test Plan. Another interesting point is the use of the repository to obtain feedback for evaluation purposes. By looking at the changes-log in the repository, it is possible to observe which versions of which software elements and when were committed to the repository. This is very helpful for evaluating up to which extent the schedule students wrote in the Test Plan was met. This is an important issue because a last-year computer science student must demonstrate enough experience to accurately estimate time and resources to accomplish a task.

- A practical examination, in which the student is tested on skills that indicate a good level of understanding and handling of the tools used. In this respect students use to perform very well and, when asked, they show to be very capable of applying the tools to new scenarios.

In the vast majority of the cases students performed very well in the oral presentations and in the practical examination. Interestingly some students went beyond the scope of the course and incorporated new tools to the testing process, like it is the case of performance analysis tools. However after carefully examining the change-log of the working folder in the repository for each pair of students, it seems like some of them have troubles estimating the time needed to accomplish each task. We attribute it to the lack of experience taking to the practice the testing techniques introduced. We expect to get better performance in this respect in the following.

The final results indicate that about 85% of students (as an average of the three years in which the course has been held) completed the course satisfactory, being the average grade 7.5 out of 10. This percentage is very similar to the number of students who actually completed all the exercises comprised on the course. This, lead us to the conclusion that the learning environment success is guaranteed whenever the teachers get students enough involved on it.

EVALUATING THE EFFECTIVENESS OF THE APPROACH

To evaluate the impact of the course on students learning and attitudes we carried out a series of surveys. Surveys took place at the beginning and at the end of the course that has been held during the last three academic years. Despite the voluntary nature of the surveys, 93% of an average of 150 students per year completed them. It is important to note that students knew that their answers to the questions would have no effect in their grades. The purpose of the surveys was to evaluate whether students attitudes over relevant software testing topics covered during the course, changed accordingly with our previously stated hypotheses.

The attitude evaluation survey was designed in a similar fashion to the one presented in (Sitaraman, M., Long, T.J., Weide, B.W., Harner, E.J. & Wang, L. 2001). The survey consists on 18 statements. Each of them must be marked by students with one of six choices: strongly disagree (1), disagree (2), moderately disagree (3), moderately agree (4), agree (5) and strongly agree (6); where the number in brackets is the score associated to each choice. Table 2 contains 6 of the 18 sentences that compose the attitudinal survey. Note that each sentence is labeled with a word ("positive" or "negative") that refers to the expected trend for the sentence's results when comparing surveys at the beginning and end of the course.

Table 2. Attitudinal survey questions and expected trends

1.	To put a special effort in ensuring the software design quality plays a fundamental role in facilitating the software testing process. (positive)
1.	Software testing is an effective and powerful way to increase software reliability. (positive)
1.	Software testing process starts when all the source code is written and it is always the last stage of the software development life-cycle. (negative)
1.	Software testing is a very time consuming process. (positive)
1.	Software testing is a very repetitive and tedious task; however, no special skills are required to get satisfactory results. (negative)
1.	The best results you can find out once the software testing process is done, is that neither errors nor failures were found. (negative)

Table 3. Summary of attitudes changes

Sentence	Before	After	Difference	P-Value	Significant?
1	3.3	5.1	+1.8	< 0.01	High
2	4.5	5.3	+0.8	0.02	Yes
3	3.9	2.8	-1.1	< 0.01	High
4	4.1	4.5	+0.4	0.2	No
5	5.1	2.3	-2.8	<0.01	High
6	5.3	2.6	-2.7	<0.01	High

Table 3 summarizes results from the sentences contained in Table 2. These results are obtained from 134 students of the course that was offered in the 2005-2006 academic year. The first column indicates the number of the sentence. The second and third columns indicate the average agreement scores for each sentence in the surveys taken at the beginning and at the end of the course respectively. The fifth column shows the P-value derived from one-sided paired t-tests on the raw data. These values are used to determine the statistical significance, which is contained in the sixth column.

CONCLUSSION AND FUTURE WORK

Looking at the overall results for the 2005-2006 course's surveys, we see that students' attitudes changed with a high or at least a positive statistical significance for 15 out of 18 sentences. Moreover, results associated to the sentences designed to evaluate the effectiveness of pair programming confirm initially stated hypotheses always with high statistical significance.

At the beginning of the course students are skeptical about the benefits of a formal software testing process when they realize the amount of work and time that such a process demands. However, we have found that the software testing environment presented here, change students' perception about the value of software testing to improve software reliability. Students see how to test an almost real application and we observe it really encourages them. During the years in which the course has been held, pair programming has demonstrated to be an effective collaborative work model, especially when two students with very different skills are grouped into the same pair. In this case, both members' skills converge to be at least equal to the higher one at the end

of the course. After the training, students work was evaluated in terms of completeness, effectiveness, maintainability and level of automation, results show that more than 85 percent of students performed above the required level. This evaluation also shows that students usually have difficulties when doing integration tests. They don't know how to start the integration and how to progressively select new classes to be added to the integration test. Coaching is much needed at this point. Final evaluations also show that students are generally satisfied with their work and consider the methodology experimented to be useful in the long term.

Our current work and intention for the future is to update and enhance the learning environment by incorporating into it the most relevant software testing related trends and techniques among those that are continuously arising in the software testing world. In particular, we are currently working to incorporate into the course testing of database access and new ways to identify test anti-patterns.

REFERENCES

Astrachan, O., Duvall, R.C., & Wallingford, E. (2001). *Bringing Extreme Programming to the Classroom*. Presented at XPUniverse Conference'01, 2001.

Beck, K. et al. (2001). Agile Manifesto. Retrieved March 30th 2007, from http://agilemanifesto.org/

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison Wesley.

Cobertura. (2005). Retrieved from http://cobertura.sourceforge.net/

Cockburn, A., & Williams, L. (2001) The costs and benefits of pair programming. In G. Succi and M. Marchesi (Eds.), *Extreme Programming examined* (pp. 223-243).Boston: Addison-Wesley.

Collofello, J. & Vehathiri, K. (2005). *An Environment for Training Computer Science Students on Software Testing.* Paper presented ad Frontiers in Education, 2005. FIE '05. 19-22 Oct. 2005, T3E-6- T3E-10.

Edwards, S. (2003). *Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance*. Paper presented at International Conference on Education and Information Systems: Technology and Applications EISTA 2003, Orlando, FL, 2003.

Kaufmann, R., & Janzen, D. (2003). *Implications of test-driven development: a pilot study.* Paper presented at 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA 2003), Anaheim, CA, 2003.

Lui, K.M., & Chan, K. C.C. (2003). When Does a Pair Outperform Two Individuals?, *Lecture Notes in Computer Science*, Volume 2675, 225–233.

Mackinnon, T., Freeman, S., & Craig, P. (2000). *Endo-Testing: Unit Testing with Mock Objects.* Presented at eXtreme Programming and Flexible Processes in Software Engineering - XP2000.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). *The Effects of Pair-Programming on Performance in an Introductory Programming Course*. Presented at 33rd SIGCSE technical-symposium on Computer science education. 2002, 38-42.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2003). *The impact of pair programming on student Performance, perception and persistence.* Presented at Int.Conf. on Software Engineering (ICSE2003), 2003, 602-607.

Melnik, G., & Maurer, F. (2002) *Perceptions of Agile Practices: A Student Survey.*" Paper presented at Agile Universe/XP Universe 2002, Chicago, IL, 2002.

Mugridge, R. (2003). *Challenges in Teaching Test Driven Development*. Paper presented at XP 2003, Genova, Italy, 2003.

Müller, M., & Hagner, O. (2002). Experiment about test-first programming *Software, IEE Proceedings* vol. 149, pp. 131-136.

Müller, M., & Tichy, W. (2001). *Case study: extreme programming in a university environment*. Paper presented at Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on, Toronto, Ontario, 2001.

Reichlmayr, T. (2003). *The agile approach in an undergraduate software engineering course project*. Paper presented at Frontiers in Education, 2003. FIE 2003. 33rd Annual, Boulder, CO, 2003.

Shukla, A., & Williams, L. (2002). *Adapting extreme programming for a core software engineering course*. Paper presented at 15th Conference on Software Engineering Education and Training, 2002. (CSEE&T 2002), Covington, KY, 2002.

Sitaraman, M., Long, T.J., Weide, B.W., Harner, E.J. & Wang, L. (2001). *A formal approach to component-based software engineering education and evaluation*. Paper presented at 23rd International Conference on Software Engineering. ICSE 2001.

Subversion. (2000). Retrieved from http://subversion.tigris.org/

Tinkham, A., & Kaner, C. (2005). *Experiences Teaching a Course in Programmer Testing*. Paper presented to Agile Conference, 2005. 24-29 July 2005, 298- 305.

Wick, M., Stevenson, D., & Wagner, P. (2008). *Using Testing and JUnit Across the curriculum*. Presented at 36th SIGCSE technical symposium on Computer science education, 2005, 236–240.

Williams, L., Kessler, R. A., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair-Programming, *IEEE Software*, 17(4), 19-25.

Williams, L. A., & Kessler, R. R. (2000). *The Effects of'Pair-Pressure' and 'Pair-Learning' on Software Engineering Education*. Presented at 13th Conference on Software Engineering Education and Training, March 2000, 59-65.

Williams, L., & Kessler, R. (2000). Experimenting with industry's pair programming model in the computer science Classroom. *Journal of Computer Science Education*, 10(4).

This work was previously published in Software Engineering: Effective Teaching and Learning Approaches and Practices, edited by H. Ellis, S. Demurjian, & J. Naveda, pp. 233-249, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 5.16
European National Educational School Authorities' Actions Regarding Open Content and Open Source Software in Education

Riina Vuorikari
European Schoolnet (EUN), Belgium

Karl Sarnow
European Schoolnet (EUN), Belgium

ABSTRACT

This chapter provides an overview into policies in the area of e-learning that ten European countries, all members of European Schoolnet, have taken regarding open content and free and open source software (FOSS) to be used to support and enhance learning. Additionally, it elaborates on European Schoolnet's initiatives to support open learning resources exchange in Europe. European Schoolnet (EUN) promotes the use of information and communication technologies (ICT) in European schools, acting as a gateway to national and regional educational authorities and school networks towards Europe. A variety of actions have been initiated by a number of European educational authorities from analysis and feasibility studies to the development of educational software based on open source as well as open educational content.

INTRODUCTION

European Schoolnet is a network of 27 national educational authorities in Europe in the area of compulsory education (K-12). European Schoolnet provides insight into the educational use of information and communications technologies (ICT) in European schools for policy-makers and education professionals. This goal is achieved through communication and information exchange at all

levels of school education using innovative technologies, and by acting as a gateway to national and regional school networks.

In recent years, European Schoolnet and a number its members have, little by little, begun a trend towards awareness building, piloting, development, and the rolling-out of open source software programs for schools, as well as investigating open content as a possible addition to a more conventional content provision.

This chapter introduces some of these policy-level actions; however, it cannot be regarded as an exhaustive summary of policy initiatives in the field of ICT and education. There are two main focuses for the chapter, the policy initiatives and EUN initiatives.

First, the chapter introduces a number of emerging initiatives lead by ten EUN member countries in the area of open source and content for education. Initiatives are categorized in four main sections: awareness raising of Free Open Source Software (FOSS), development of LMS and learning platforms, promotion of the use of Linux on desktops and educational servers, and finally, the promotion of open content. The following countries are featured: Estonia, Spain, and Slovenia as an example of countries basing part of their policy initiatives and actions on open source development; Belgium's Flemish Community and the Netherlands, which run major campaigns to raise awareness of the FOSS issues; Ireland and Finland, as well as France, with smaller scale policy initiatives to familiarize schools with alternative solutions; and finally the UK and Lithuania carrying out feasibility studies with FOSS.

The second part presents two European Schoolnet's recent initiatives in this regard: Xplora, which promotes science education in Europe, and secondly, the EUN's Learning Resources Exchange, which promotes the use and reuse of educational content across Europe. The latter introduces the implementation of a digital rights management framework and briefs on the current development of a learning toolbox to support collaborative learning based on open source development.

BACKGROUND

European Schoolnet (EUN) was funded in 1996 with the mandate of the Council of the European Union. The members of European Schoolnet represent national and regional educational authorities such as the Ministry of Education (MoE) or National Board of Education. Its mission is twofold: on the one hand, EUN works closely with national and regional policy-makers and shapers by setting up special interest committees, involving them in transfer of best practices, and in e-learning research and development. On the other hand, EUN works directly with a large network of European schools through special online events organized in collaboration with a variety of stakeholders.

European Schoolnet is committed in following open standards in e-learning research and development that it conducts in the field, partnering up with different stakeholders from public, private, and industry partners. This has resulted in services that allow multiple players' access to the field. Furthermore, the use and development of open source software in education is becoming more of a concern in different EUN member countries, whereas the promotion of interoperable content-based services, such as federations of learning resources repositories, has long been in the centre of EUN's attention. It is important to note that the members of EUN all lead their own policies based on their national policymaking, and that EUN only has an advisory role for its members.

Apart from reporting on European Schoolnet's partners on their national initiatives, this review will also extend to other national policymakers whenever the information was made available. Mostly, this review relies on contributions from European Schoolnet's partners.

EUN'S MEMBERS ACTIONS IN THE QUEST FOR EDUCATIONAL OPENNESS IN SOFTWARE AND CONTENT

A number of European Schoolnet's partners have explicit roles in promoting the use and development of open source software as an alternative choice for schools. A review on a selection of partners acting upon this challenge is provided in this section presenting the Ministry of the Flemish Community, Education department in Belgium, Kennisnet in the Netherlands, Becta in the UK, Tiger Leap foundation in Estonia, the Ministry of Education and Sport in Slovenia, the National Centre for Technology in Education in Ireland, National Board of Education in Finland, and the Ministry of Education and Science of the Republic of Lithuania. Additionally, some development is reported from Spain and France.

The actions, initiatives, and policies that countries have undertaken in this regard vary throughout Europe. Table 1 presents the main action areas, which can be classified as following; awareness building and distribution of open source software for schools, feasibility studies regarding the deployment of open source software and/or open content, initiatives or pooling resources in the area of software development of learning management systems (LMS) and learning platforms, development or localization of GNU/Linux distributions for schools' use in native languages, and finally, in the area of open content. The list presented in this report does not cover equally all the European countries and is not exhaustive by any means; rather, it serves the purpose to highlight some of the good practices and initiatives.

Estonia: Tiger Leap Foundation (Tiigrihüppe Sihatusutus)

In 2004 in Estonia, the Tiger Leap Foundation (TLF) initiated a project for distribution and

Table 1. Main action areas of a number of EUN member countries in FOSS and open content

Country	Awareness building/FOSS distribution	Feasibility studies	Common development of software	Development of GNU/ Linux	Open content
Flemish Community of Belgium	x				x
Estonia	x		x	x	x
Finland	x				
France	x		x	x	
Ireland	x				
Lithuania		x			
Netherlands	x	x	x		
Slovenia	x		x	x	x
Spain	x			x	x
UK	x	x			

promotion of freeware in schools. The project aims at releasing a Linux distribution that is suitable for schools, preparing training materials, and training teachers. Furthermore, since the spring 2005, TLF only supports projects that will be released under general public license for the code; as for the content, a creative commons license will be required.

A number of Estonian educational open source software applications have been developed, with the financial support of TLF in collaboration with Tallinn University (TU). The development of virtual learning environment VIKO started in 2001. Schools do not have to set up their own server, VIKO is offered as a free service by Tallinn University. Furthermore, KooliPlone, a Plone-based content management system for school Web sites is developed in TU.

Another large-scale development of a learning management system, called IVA, was supported by Estonian Ministry of Education and Science, the Estonian Information Technology Foundation, and Hansapank, the largest bank in Estonia. IVA is also developed in TU, based on Zope and an existing educational platform called Fle3. It has Estonian, Russian, and English user interfaces, and is currently used by more than 2,000 users in TU.

Additionally, Estonia being a country representing a small market, the government has funded the translation of OpenOffice's spell-check program in Estonian.

Slovenia: Actions by the Ministry of Education and Sport

Slovenian Ministry of Education and Sport has a focus on three main areas providing basic tools, didactic tools, and promoting open source for teachers, headmasters, and pedagogical specialists.

To the category of providing basic tools, the Ministry includes Linux, OpenOffice.org, CMS, LMS, as well as some distance learning services. First of all, all new computers in schools, which are co-financed by the Ministry of Education and Sport, have a dual boot for Windows and Linux, and have OpenOffice.org installed for both operating systems. The Linux distribution is called Pingo and is provided in Slovene. Pingo has been developed by a local association, called Lugos, with the Fedora Linux 3 open source community in Slovenia. For the last two years, the Ministry of Education and Sport, the Ministry of Information Society and the (governmental) Centre for Informatics have financed the localization in Slovene language. Currently, a tender to co-finance the localization for the next two years is under preparation.

Another example of learning management and learning content management system that is available free of charge for academic institutions is E-CHO, developed by the Faculty of Electrical Engineering, University of Ljubljana. It enables simple course creation, extensive content management, and it supports entire learning process.

Secondly, in the area of basic teacher training, among other ICT skills, the programs include the use of Windows Office as well as OpenOffice.org. The Ministry financed an expert group that supports schools with open source software, such as some CMS and LMS, and support books have been distributed to schools about the use of Linux and OpenOffice.org. Moreover, the Ministry with National Education Institute and Center for Vocational Training promotes the use of open source software among teachers, headmasters, and didactic specialists.

As for didactic tools and open content, the Ministry finances teacher training in the area of open source didactic materials (i.e., open content). It has also co-financed some new open content didactic material on the Web for the use in classrooms, with some support given for teachers in training to use this material. As well, the Ministry will co-finance, in the future, creation of didactic material that is not open source, but can be used freely by schools.

To promote the use of open source and open content, the Slovenian Ministry of Education and Sports has started the portal OKO. This project is with the intention to make the introduction of open source and free educational software into education environments faster and more efficient. The OKO project was started in 2003.

Spanish Initiatives

Spain has had some regional policy initiatives to promote the use of open source, and especially Linux use, in school servers and desktop. Probably, the region of Extremedura is the best-known large-scale example in Spain, where the local information strategy is based on the use of free and open source software. The local authorities decided to boost development in the region, which is prone to high unemployment, by delivering over 150 000 cd-roms containing a localized Linux distribution, Linex, along with dedicated office software. It has been distributed in different ways. In Extremadura, schools are immersed into using Linux; it has been deployed on around 70,000 desktop PCs and 400 servers in schools. Some other communities in Spain who have decided to do so are at least Andalucia, Madrid, and Valencia. In two of them, the school's workstations are delivered with a dual boot configuration, so that the users have the option to work either on Linux or Windows, and can familiarize themselves with two different operating systems. In the community of Galicia, new computers to schools are delivered with Linux installed.

Belgium: The Ministry of the Flemish Community, Education Department

The Ministry of the Flemish Community, Education Department in Belgium has an explicit role in promoting the use of open source software as an alternative choice for schools. In 2004, the former minister of Education, Ms. Marleen Vanderpoorten, commissioned an advisory on the issue, which led to a vision and a proposed action plan.

In 2005, a large campaign was organized to introduce free and open source software in Flemish schools, aiming to highlight its educational possibilities. In this campaign, a publication, a CD, and an educational tools database were drafted and a conference was organized.

By means of the publication "free software in the education," a practical guide for the use of FOSS and open educational tools is spread amongst all schools. Besides general information on the "what and how" of FOSS, one finds descriptions of a number of interesting open source applications. In association with the educational portal Klascement, an educational tools database was developed for these applications. This is also the general campaign Web site. Moreover, a conference was organized addressing FOSS and open educational tools, targeting audiences such as teachers, headmasters, and ICT co-coordinators.

The Education Department in Flanders has created didactic sheets on the use of educational freeware and open content, based on the primary education curriculum topics. The didactic sheets have been published as a book, "ICT on the menu," and are searchable in a database through the portal. The scenarios are a helpful means to make the ICT integration in primary education more concrete. In 2005, a similar project was developed for secondary education. This time, the work was carried out by teachers from the secondary ENIS schools. The result is a publication, both on paper and online, called "Digital resources for secondary education." In 2006, a CD was published, with open learning tools and open source educational software that is currently under a validation process by the European Network of Innovative Schools (ENIS).

The Netherlands: Open Source and Open Standards in Education (OSS in het onderwijs)

Since 2003 in the Netherlands, the government has brought open standards and open source into the central focus of its attention. A variety of initiatives have been set up to work on cross-sectoral issues that touch upon open standards, as well as open source development. As for the education, there are initiatives, programs, and actions taken to foster the efforts in the field and to muster the common efforts.

The program "OSS in het onderwijs," translated as open source and open standards in education, is a joint initiative between Kennisnet, ICT op School, and a government-wide program called OSSOS, the Program for Open Standards and Open Source Software. Additionally, to involve a diversity of partners in the field of education, an association called EduStandaard has been set up. The association aims to manage the standards that are used in the Dutch educational field, comprising stakeholders such as publishers, schools and so forth.

Kennisnet promotes a program to improve the use of open standards for content. A central point for "OSS in het onderwijs" is a Webspace where the Dutch education community can discuss open source and open standards aiming at both the novices and experts. The main focus is on primary and secondary education, but also on the field of vocational training. The program is informative, aiming at offering alternative solutions for schools that have an independent budget to spend on educational technologies. The program targets mainly the IT coordinators, administrators, and teachers who are responsible for IT set-ups in schools, but also at teachers who use computers and ICTs in their lessons.

One powerful means to transfer good practices and ideas of the use of FOSS in education are the case studies that can be found at the Web site of "OSS in het onderwijs". These case studies are simple descriptive interviews with practitioners on topics such as how to use GIMP for manipulating images, and so forth.

"OSS in het onderwijs" has also prepared an info package, on a CD, that focuses on the use of open standards in all processes in school that can involve the use of information technologies from administrative tasks to using applications for learning purposes, gathering content about the student for portfolios, as well as other actions for creation, exchange, and alteration of the content. The idea was to identify all actions and propose alternatives where closed systems or standards are used. This aims at better overall interoperability within schools' information systems. The CD was released in the end of 2005. In the same spirit, a booklet on open source software was created in 2004 for schools. These information packages can be requested from the Web site, but they are also handed out at local ICT conferences.

The program "OSS in het onderwijs" can help schools to implement open source, not only in advisory terms, but they can make small amounts of money available to pay for a third-party programmer or consultant to, for example, find compatibility solutions between an existing system and the new one based on open source and standards. On the Web site, there is also a FOSS helpline for schools to help them to solve small-scale problems. In this regard, the program tries to match the need that schools have for support with existing supply in the market. On the Web site, one can find an overview of companies with experience of FOSS and education.

"OSS in het onderwijs" has been running for 3 years, 2005 being the final year with a big push, the continuation for the next year is still unsecured. A conference, with 1-day education track on the topics, was held in December 8 2005.

France: Distribution of FOSS

In France, the National Centre of Pedagogical Documentation (SCÉRÉN-CNDP) has led a

working group that evaluated some 30 educational open source software packages, with an emphasis on multiplatform usage. The software, and accompanying educational guidelines for its use in the classroom, was made available for schools, in the format of a CD, through a network of regional centers, at minimum charges, in 2004. The CD has had a good success, and its distribution continues.

On the school server level, advances have been made to introduce GNU/Linux for school servers. An initiative, SIIEE (service intranet-internet in educational establishments and schools), has deployed about 15, 000 Linux-based servers throughout the country.

Furthermore, some regional French authorities have also embraced FOSS. For example, the Pays du Soissonnais has declared as its ambition to become an axle of excellence in open source, making it part of its regional policy, grouping a variety of local actors together to find ways to benefit from open source. The Region of Picardie participates in the initiative; already since 2004, each school receives earmarked money to be spent on open source software to buy services or to hire someone to do the work locally. Partially, the idea behind this initiative is to boost local economy (Picardie ranks high for unemployment), but the objective is also to use public money for the public good. Picardie, as a region, also has a high profile on free and open source software, and participates actively in the annual trophy competition for the best educational software.

Finland: Informing about Educational FOSS

In 2006, the National Board of Education in Finland launched an initiative to gather an information package for schools about the use of open source software and content for educational purposes. The aim is to distribute information on how open source software can be used to support educational purposes for creating educational content, as well as for school administrational matters, but also to clarify the basic terminology to be used in the area. The target audience is teachers, the ICT coordinators, and school heads. The information is gathered in collaboration with local experts in the field, as well as practitioners who have been using and adapting FOSS solutions in their schools and educational establishments.

Another type of interesting initiative can be reported from Finland, although it does not directly receive any educational attention. To make open source development rooted into a variety of sectors within society, including education, a sustainable plan for services and support should be in place. The Finnish Centre for Open Source Software (COSS) aims to help its building. COSS started at the beginning of 2004 with basic funding, granted by Finland's Ministry of the Interior, to help to build an ecosystem within business, research, and education. Currently, the Häme Centre of Expertise, as a partner of COSS, is responsible for developing open source based business in the field of e-learning. It could be contemplated that this type of idea and innovation feeding across sectors has a positive value for the uptake of open source software in educational establishments, as they will have local businesses to rely upon for the support, sale, and delivery of these e-learning services.

Ireland: Star Office for all Irish Schools

In late 2004, the National Centre for Technology in Education (NCTE), the Irish Government agency established to provide advice, support, and information on the use of ICT in education, concluded a licensing and distribution agreement with SUN Microsystems to provide all Irish schools with Star Office, an office suite based on open source OpenOffice.org. The offer was made to schools in a joint move by the NCTE and SUN Microsystems.

To help schools appreciate the opportunity and to explore the implications of taking up Star Office, or substituting the commonly used MS, schools were notified and local information sessions were organized for school representatives. These sessions were well attended and, following participation, the take up has been significant to date. Schools receive a free CD that allows unlimited copying for staff and students.

Prior to the large-scale offer of Star Office in 2004, the NCTE carried out a number of evaluations of Star Office in a number of schools in order to assess the appropriateness of this software for schools. The outcome of these trials proved very positive. Star Office was identified as being a relevant and very useful software tool, particularly for schools at primary level.

The UK: Evaluation on Open Source Software in Schools

In May 2005, the British Educational Communications and Technology Agency (Becta), released an evaluation on the use of open source software within a number of schools. In the UK, some previous government studies have suggested that the use of FOSS within the UK public sector can provide a viable and credible alternative to propriety software, and lead to significant cost savings.

The study, funded by the Department for Education and Skills, had three main aims: to examine how well the open source software approach works, compared with proprietary offerings, in supporting delivery of the school curriculum and administration; to compare the total cost of ownership (TCO) of using FOSS within school environments against that of non-open-source solutions; and to highlight examples of successful school-based open source implementations.

The report, "Open Source Software in Schools: A study of the spectrum of use and related ICT infrastructure costs," demonstrates that although the implementation of FOSS in schools needs careful planning and support, it can offer a cost-effective alternative to proprietary software. For the ways forward with FOSS, the report examines cost-effective models of support in OSS schools; best practice in licensing solutions; successful implementation to run the school's servers to provide school-wide facilities, operating systems, and administrative PCs, and FOSS applications on classroom and administrative PCs.

According to Becta's chief executive, Owen Lynch, Becta believes that software used in schools should be of a high quality and adhere to open standards, enabling compatibility and interoperability between products. Becta will now be undertaking more extensive research across a wider range of institutions to allow further analysis of these issues.

Lithuania: Research Study and Recommendations for Actions in Open Source in Education

The Ministry of Education and Science of the Republic of Lithuania commissioned a study in 2004 to further investigate the possibilities of open source software in education in Lithuania. The global fight against the use of illegal software and piracy, and the openness of the source code to guarantee more transparency and ease of localization, were mentioned in the goals of the study as important to go forward with.

The conclusions of the study outlined the following; open source software has an indirect positive impact on the economy of education through the emergence of a competitor to commercial software, forcing a reduction in the price of commercial software and translations into the languages of small nations spoken by few people.

The study furthermore proposed some actions both at the state level and ministerial level. The state level proposal included, among others, the following: it is necessary to ensure the adaptation and localization of general purpose open source

software at the state level; it is necessary to take care of the cultural and linguistic quality of open source software; the promotion and support for open source software could help solve the problem of the legality of software.

At the ministerial level of the Ministry of Education and Science, the proposed actions included analyses of localization of educational Linux distributions and open source virtual learning environments to determine which one would be appropriate to be localized and used in Lithuanian schools. As to introducing FOSS in education, the following was proposed: higher schools' training educators should introduce their students to both commercial software and equivalent open source software necessary for teaching and learning; at school, students should be introduced to general purpose software of both types: commercial software necessary for teaching and learning, as well as to equivalent open source software. Furthermore, to assure the quality of localization, an interesting proposal was made: a course on localization of software for some students specializing in information technologies and philology.

EUROPEAN SCHOOLNET'S PROJECTS DEALING WITH OPEN CONTENT AND OPEN SOURCE

This section presents the Xplora project, which promotes science education in European schools, and briefly presents the work that EUN carries out in the area of learning resource exchange (LRE), where digital learning resources and/or their metadata are exchanged between educational repositories.

Xplora, Distributing Science for Education in its True Way: Openly

Xplora, the European Science Education Gateway, is operated by European Schoolnet. Xplora portal is supported by the PENCIL project, a project funded by the European Commission's Directorate General for Research as part of Science and Society.

Xplora offers science teachers tools, information, and resources to help them to conduct engaging science lessons that make students attracted to science. Commonly, with 30+ students in classes, science teaching is somewhat blocked by poorly equipped school laboratories. Among the resources that Xplora offers are the usual Web-based tools like online games, downloadable materials, and guides to software that is usable in science lessons.

Xplora portal also offers new tools that have not been used in the classroom before. Among these tools are the Web experiments or remote controlled experiments (RCL), in which real experiments are shared via the Internet. Such experiments do not only solve the problem of the true way of science teaching by experiments, but it also opens new pedagogical concepts for science classes. These Web experiments deliver results that students have to process in order to get a lab report.

Using Software: A Key Skill in Scientific Research

Participating in science education today means extensively using software. For instance, for the Web experiments where, in many cases, students get the result of an experiment as an image, the image analysis is a fundamental task. The main tasks for students in science education are (1) create a lab report with mathematical expressions, chemical formulas, Feynman diagrams, images, tables, and graphs; (2) analyze images, for example, measure length, angles, area, and intensity; (3) calculate results, for example, numerical processing, creating graphs, regression, and curve fitting; (4) create animations; (5) run simulations; (6) create and play with mathemati-

cal models, and (7) use CAS software to verify results of calculations.

While office suites text-processing software is useful and broadly applicable to be used in schools, it is, in many cases, not sufficient for specific science tools. One of the examples is simple text editing. For science lab reports, a text writer must be able to handle mathematical equations, chemical formulas, and Feynman diagrams, just to mention the most exotic pitfalls.

Many of the open source software packages have origins in scientific environments. Thus, there are many applications that can be used for science teaching in classroom, with some prior training. Xplora recommends the use of the following software packages, displayed in Table 2, for science teaching. On the portal, one can find articles and short descriptions for their usage.

Xplora-Knoppix, Making Science Accessible for Schools

To ease some of the organizational problems that schools face in terms of software availability, installation, and access in general, the Xplora team developed a live bootable DVD called Xplora-Knoppix. It is a based on the Linux Debian distribution, and completely contained on a self-booting DVD. As this Knoppix version is especially mastered for Xplora, the team has added software applications needed for science education (Table 2), as well as a number of educational materials from the Xplora repository. The Xplora-Knoppix has multilingual support. This concept ensures easy access to scientific tools for education. Being open source software, it can be given away freely and copied as many time as needed.

Xplora produced 600 DVDs to be given freely to schools. Moreover, the ISO image of the DVD is freely downloadable from the Internet and can be used to produce the copies needed for the students. This DVD is occasionally updated. Additionally, Xplora has partnered with a company that sells the Xplora-Knoppix DVD for the production plus shipping cost.

EUN Content Services in Pipeline

Since 2000, European Schoolnet has lead EU-funded projects to give better access to digital educational resources for teachers and learners across Europe. The CELEBRATE project (2002-2004) provided the first large-scale demonstration and evaluation of learning object (LO) interoperability and the use of LOs in schools at a European level.

In 2004, a survey of 13 Ministries of Education participating in European Schoolnet also indicated that they wished to take forward the vision of learning resource exchange (LRE) based on the architecture demonstrated in the project. Furthermore, many communicated that LOs are increasingly seen as an important, and in some cases, a key component in the content development strategies of Ministries of Education. Also, the majority expressed interest in open source content development strategies where "learning object economy" was created for open source and commercial content to coexist.

EUN continues its work towards an enhanced architecture for learning resources in Europe. EUN will continue to lead the development of the LRE based on a brokerage system architecture (of which the code for the brokerage system is licensed under the LGPL) involving a variety of stakeholders, from content providers, both public and commercial, to end users in European schools.

A set of more tailored services will be offered to the members of LRE, such as federated searches, learning resource exchanges, and digital rights management. It is envisaged to support multiple digital rights expression languages, and permit content providers to select the level of digital rights management that best fits their needs in terms of intellectual property protection. This requires a proper digital rights management (DRM). The

Table 2. Open source software with open source-like licenses for use in science teaching

Name of software	URL	Description	Application
OpenOffice.org	http://www.openoffice.org	The Open Source office software for scientific text processing, database applications, graphics creation.	Lab reports. Calculation of results Creation of simple charts
LyX	http://www.lyx.org	Scientific text processor software, making use of LaTeX properties. Full support of mathematical expressions and all Postscript output from scientific programs.	Lab reports with even the equations, and output of all X11.
Xfig	http://xfig.org	Vector drawing program with a large and extendable parts library.	Preparation of schematic drawings (experimental setups) for lab reports.
Grace	http://plasma-gate.weizmann.ac.il/Grace/	Data analysis program	Plots diagrams of every complexity. Good software for creating regression and line fit.
GIMP	http://www.gimp.org	Graphics program to analyze images	Image analysis (length, angle)
ImageJ	http://rsb.info.nih.gov/ij/	Image analysis program	Analysis of intensity distribution in an image.
Xdrawchem	http://xdrawchem.sourceforge.net/	Program to draw chemical structures	Report on chemistry lab exercises.
OpenRasmol	http://www.openrasmol.org/	Program to visualize 3d molecules	Chemistry classroom use and creating images for reports.
Feynman	http://rpmfind.net/linux/RPM/suse/9.0/i386/suse/i586/feynman-1.00-581.i586.html	A program to create Feynman graphs	Particle physics teaching.
Ghemical	http://www.uku.fi/~thassine/ghemical/	A molecular modeling software package	Chemistry teaching in high schools
Gcompris	http://www.ofset.org/gcompris	A software package for the kids	For elementary schools. Many different applications around elementary schools teaching
KDE Edu	http://edu.kde.org/	The KDE Education project	Many educational software packages mainly focusing on lower-level education.

objective is to design and implement a DRM framework that takes into account requirements from all stakeholders; thus, supporting available DRM standards like ODRL and Creative Commons.

European Schoolnet supports the use of Creative Commons licenses within its services, and has already implemented an integrated interface for its users to choose an option of Creative Commons license for the resources that they submit to various EUN projects.

Open Source Learning Toolbox to Support Collaborative Learning

European Schoolnet's research into the use of learning environments confirms that a number of its members flavor the development of open source VLEs. Moreover, many expect the next generation of new learning platforms to facilitate the adoption of more learner-centered and collaborative pedagogical approaches. However, the same survey and subsequent observations suggest that these high expectations are not yet being met. Most teachers are still using VLEs as little more than a "digital distribution" space, somewhere to upload, store, and distribute content, and to issue assignments to students.

EUN, among the CALIBRATE project partners, will lead the development of a VLE that brings together two quite distinct and somewhat opposing methodologies for pedagogical affordance; the first comes from a background of social constructivist pedagogies and collaborative knowledge building, whereas the second has a background in SCORM and LCMSs. By drawing on both these approaches, a new open source toolbox will be built using the existing code from Future Learning Environment 3 (FLE3) based on Plone/Zope. The VLE will offer a richer feature set that will be developed with the help of practicing teachers.

CONCLUSION

Currently in the European educational policy and practices landscape, the existing open content and free and open source software initiatives are rather dispersed on a local, national, and European level, as well as being spread throughout all educational levels and systems. It is challenging to get a comprehensive overview on the state of the art being available, as well as capitalizing on the transfer of knowledge gained in one context. However, as this report clearly summarizes, European Schoolnet and its members are more and more focusing on the issues around open source and content development. It must be stated, though, that these activities still remain somewhat marginalized in discussions, country reports, and conferences; they rarely receive the limelight that they merit.

When it comes to actions taken by European Ministries of Education and other national educational authorities, it seems like they are keen to explore the advantages that open source software and content can offer to education. According to the interviews conducted for this chapter, it can be stated that 10 out of 25 EU member states have taken policy-level actions in e-learning to better exploit the use of FOSS in education. In this report, we looked at five different categories where attention is given: awareness raising of FOSS as an alternative solution for e-learning tools; feasibility studies regarding the use of FOSS to support education, both in teaching and administration; development of learning platforms based on FOSS; localization of GNU/Linux desktops to be used in schools; and finally, open content as an alternative means for digital content provision. It appears that the featured countries have taken actions in all these areas, some in all, whereas others in one or two. Moreover, some of the countries seem to have ventured into the area of FOSS just to find

out more about it, whereas the others have embedded FOSS into their policy-making strategies and initiatives in a more integrated way.

This chapter attempts to report on these developments, rather than going further to analyze them in details. Many interesting questions have been raised regarding these policy-level initiatives, more importantly, what are the synergies between grass-root actions in schools and the policy and decision-makers responsible for these initiatives, their continuation, their accountability, and so forth? Are these policy-initiatives a response to the demand in the field, or do they support the further uptake of FOSS in education?

It seems that it would be important to bring these somewhat disparate, but very pertinent national and regional initiatives into the European level to better help the transfer of good practices and to learn from one another. Furthermore, peer-learning possibilities on the policy level should be better exploited in this area, as have been done in other areas of ICT implementations.

Xplora carries out important work promoting science in education in European schools. The multiple ways to distribute software that is suitable for scientific studies allows schools a better access to the core of science: participate by practicing it.

Finally, the work EUN has carried out in publishing the Insight Special Reports has given a more prominent voice for FOSS in education, and been an important source of information for EUN's members and audiences on national levels.

The area of open content seems to be rather well accepted concept among EUN's partners. Thus, creating infrastructure and facilitating the content exchange of learning resources in schools is one of EUN's core areas where significant work is conducted to facilitate the coexistence of open and "closed" content. For example, the implementation of digital rights management framework is a step towards the coexistence of multiple stakeholders in the field of educational content. Also, some important work will be carried out in the context of EU-founded projects such as where the development and implementation of an open source collaborative "learning toolbox" for schools is being done.

APPENDIX I: INTERNET SESSION: INSIGHT, OBSERVATORY FOR NEW TECHNOLOGIES AND EDUCATION

http://insight.eun.org

Insight is a service focusing on e-learning in schools in Europe. It is provided by European Schoolnet (EUN) in collaboration with its consortium members.

Interaction

We publish news, reports, and analysis on e-learning policies, school innovation, and information communication technology (ICT) in education. In the section of Insight Special Reports, three relevant reports are found dealing with FOSS for Education in Europe. http://insight.eun.org/ww/en/pub/insight/misc/specialreports.htm

APPENDIX II: USEFUL URLS

- **Links Related to *Introduction* and *Background***

LRE: **http://lre.eun.org**
EUN: **http://www.eun.org; http://www.europeanschoolnet.org/**
CELEBRATE: **http://celebrate.eun.org**
CALIBRATE: **http://calibrate.eun.org**

- **Links Related to Estonia**

Tiger Leap Foundation: **http://www.tiigrihype.ee/eng/arhiiv_1.php?uID=49**
VIKO: **http://www.htk.tlu.ee/viko/, IVA http://www.htk.tpu.ee/iva/**

- **Links Related to Slovenia**

OKO **http://oko.edus.si**
E-CHO, developed by the Faculty of Electrical Engineering, University of Ljubljana **http://dl.ltfe.org**

- **Links Related to Spain**

Article on the use of FOSS in education:
http://insight.zdnet.co.uk/software/linuxunix/0,39020472,39197928,00.htm
Brings hope to Spain's poorest region:
http://news.zdnet.co.uk/

- **Links Related to Belgium**

The Flemish advice on FOSS in education, is available in English at **http://www.ond.vlaanderen.be/ict/english/**

Portal Klascement: **http://vrijesoftware.klascement.net**

ICT on the menu: **http://www.klascement.net/ictophetmenu**

Digital resources for secondary education: **http://digitaalso.klascement.net, ENIS: http://enis.eun.org**

- **Links Related to The Netherlands**

Kennisnet **http://www.kennisnet.nl**
In the section "voorbeeldprojecten," one can find short descriptions of different case studies OSSOS, Open Standards and Open Source Software in Government in English **http://www.ososs.nl/index.jsp?alias=english**

ICT op School **http://www.ictopschool.net**,
Dutch association for a wide range of stakeholders in e-learning standards **http://www.edustandaard.nl/**
Kennisnet on content: **http://contentketen.kennisnet.nl/**
Conference announcement: **http://www.ossos.nl/article.jsp?article=1820**

- **Links Related to France**

Information on FOSS for education at **http://logiciels-libres-cndp.ac-versailles.fr/**
press release about the CD **http://logiciels-libres-cndp.ac-versailles.fr/IMG/CP_Logiciels_libres.pdf**
information about the school server program SIIEE: h**ttp://www.solutionslinux.fr/fr/conferences_detail.php?id_conference=64**
Region of Picardie: **http://www.cr-picardie.fr/article.php3?id_article=374**

- **Links Related to Finland**

Finnish Centre for Open Source Software **http://www.coss.fi/en/**
EduCoss http://educoss.org/

- **Links Related to Ireland**

Irish news on Star Office **http://www.ncte.ie/NewsandEvents/Newsletter/d2413**.HTML.html, press release in pdf-format: **http://www.ncte.ie/documents/pressreleaseforStaroffice.pdf**

- **Links Related to The UK**

Open Source Software in Schools: A study of the spectrum of use and related ICT infrastructure costs – Project report **http://www.becta.org.uk/corporate/publications/documents/BEC5606_Full_report18.pdf**
Open Source Software in Schools: A case study report **http://www.becta.org.uk/corporate/publications/documents/BEC5606_Case_Study_16.pdf**
Using Open Source Software in Schools: Information sheet **http://www.becta.org.uk/publications/documents/BEC5606_Information_Sheetrev.pdf**
Previous UK Government studies include: Office of Government Commerce [2004] "Open Source Software Trials in Government: Final report" **http://www.ogc.gov.uk**

- **Links Related to Lithuania**

The Ministry of Education and Science of the Republic of Lithuania, Centre of information technologies of education, Institute of mathematics and informatics: Report of the research study open source in education **http://www.ipc.lt/english/apie/skelbiami_dok/2004/Open Source in Education. Abstract of Research Study.doc**

- **Links Related to Xplora**

http://www.xplora.org

Web experiments: **http://www.xplora.org/ww/en/pub/xplora/megalab/web_experiments.htm**
Xplora Knoppix DVD **http://www.europeanschoolnet.org/ww/en/pub/eun/news/news_headlines/1107.htm, Linux-cd.info: http://linux-cd.info/**

- **Links Related to Learning Resources Exchange**

http://lre.eun.org
CALIBRATE **http://calibrate.eun.org**
FLE3 **http://fle3.uiah.fi**

APPENDIX III: FOR FURTHER READING

Flosse Posse: Free and open source for education: http://flosse.dicole.org

Insight policy brief: VLEs, open standards and open source in European schools

Insight special report: Software patents - a potential hindrance of ICT in education http://insight.eun.org/ww/en/pub/insight/misc/specialreports.htm

Insight special report: Why Europe needs free and open source software and content in schools

Naeve, A., Nilsson, M., Palmér, M., & Paulsson, F. (2005). Contributions to a public e-learning platform: Infrastructure, architecture, frameworks, and tools. *International Journal of Learning Technology, 1*(3), 352-281.

Open source for education in Europe: Research and practice: http://www.openconference.net/index.php?cf=3

Possible Titles for Papers/Essays

- Examples on How the Support from Local Authorities Can Effect the Good Practices in the Field: Case Studies of Foss Implementations on the Regional and National Level
- Examples on Efficient Grass Root FOSS Actions in Education, and How It Has Had an Effect on Local Policy Making (look for examples where national and regional policy and decision makers have created support systems for FOSS in education, and try to find whether it has had an effect on the larger-scale uptake)
- Attitudes of Local and Regional Educational Authorities Towards FOSS in Education
- Future Scenarios for Adapting and Rooting FOSS in Regional, National, and European Level Policy Making in Education
- If I Were a Local Educational Policy Maker, What Would I Do for FOSS in Education?

This work was previously published in Open Source for Knowledge and Learning Management: Strategies Beyond Tools, edited by M. Lytras and A. Naeve, pp. 245-265, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.17
Enhancing Skills of Application Software via Web-Enabled Problem-Based Learning and Self-Regulated Learning: An Exploratory Study

Pei-Di Shen
Ming Chuan University, Taiwan

Tsang-Hsiung Lee
National Chengchi University, Taiwan

Chia-Wen Tsai
Ming Chuan University, Taiwan

ABSTRACT

The computer software education in vocational schools in Taiwan can hardly be deemed as effective. To increase students' learning motivation and develop practical skills, innovative learning designs such as problem-based learning(PBL) and self-regulated learning (SRL) are on trial in this specific context. We conducted a series of quasi-experiments to examine effects of these designs mediated by a web-based learning environment. Two classes of 106 freshmen in a semester course at Institute of Technology in Taiwan were chosen for this empirical study. Result sreveal that effects of web-enabled PBL, web-enabled SRL, and their combinations, on students' skills of application software have significant differences. The implications of this study are alsodiscussed.

INTRODUCTION

Professionals with a vocational degree represent a major portion of the work force in Taiwan. Vocational education is highly competitive in that it must attract enough student enrollments in the

face of a continually decreasing birthrate and a rapidly increasing number of schools. Students in these schools tend to have lower levels of academic achievement. They spend more time on part-time jobs and do not get involved in their schoolwork adequately. They also care less about their grades. Teaching in such a context, particularly teaching the curriculum of application software, is a great challenge to most educators.

No one doubts the guiding principles of practical applications in the vocational education in Taiwan (Tai, Chen, & Lai, 2003). However, most teaching and learning efforts in this area have been devoted to helping students pass written tests, and, thus, receiving awards or official certificates. Schools, facing the high pressure of market competition, often emphasize the proportion of students awarded such certificates before they graduate instead of quality education. This materialistic aim puts students' attention less on mastering application software and more on preparing for tests through memorization. Consequently, a student who has passed the examination may still be unable to apply what was learned in school, and worse, lack motivation to learn more in the future.

The courses in application software traditionally emphasize memorization by applying short, disjointed, lack-of-context examples. There is a wide gap between what is learned in school and what is required in the workplace (Wu, 2000). In this regard, the computer software education in vocational schools in Taiwan can hardly be deemed as effective. In order to increase students' learning motivation and develop practical skills, problem-based learning (PBL) is considered to be the most appropriate. PBL uses real-world, simulated, contextualized problems in practice to motivate, focus, and initiate content learning and skill development (Boud & Feletti, 1991; Bruer, 1993; Williams, 1993). We believe that PBL could help low-academic-achievement students to develop practical skills of application software through online courses.

Web-based instruction seems to be an ideal learning environment because students can access an almost-unlimited amount of information and apply it in multiple ways (Kauffman, 2004). However, implementing e-learning for low-academic-achievement students inevitably runs high risks. For instance, Internet addiction is quite common among low-academic-achievement students. When students enter the traditional classroom, they are used to logging on to MSN Web Messenger and checking their e-mail first. Many students like to chat with each other frequently via MSN Web Messenger, even though they are in the same classroom. They might browse shopping Web sites, even while a teacher is lecturing in the classroom. Thus, the teacher has to disconnect the network several times in his classroom to focus students' attention. It is even more difficult for students to concentrate on online learning because of this addiction to the Internet and a lack of on-the-spot teacher monitoring. To respond to this challenge, we propose an approach that can help students regulate their learning in a better way.

Success in online courses often depends on students' abilities to successfully direct their own learning efforts (Cennamo, Ross, & Rogers, 2002). It is very critical to develop students' self-regulation of learning before providing online courses to them. In web-based learning environments, physical absence of an instructor and increased responsibility of learners to effectively engage in learning tasks may present difficulties, particularly those with low self-regulatory skills (Dabbagh & Kitsantas, 2005). Student motivation may benefit from Web-based instruction with self-regulated learning (SRL) strategies. Students in the online environment, equipped with SRL competence, become more responsible for their learning and more intrinsically orientated (Chang, 2005). So self-regulation is important, particularly while learning in World-Wide-Web-supported environments (Winnips, 2000).

Although researchers have consistently shown that self-regulation helps high achievers reach their potential (Risemberg & Zimmerman, 1992),

it also makes a difference between failure and success for low achievers (Borkowski & Thorpe, 1994). However, there has been relatively little empirical research on students' SRL with such complex technology-based learning environments (Azevedo & Cromley, 2004). Therefore, this study applies SRL in this study to help vocational school students (particularly the low achievers) concentrate on their learning, leave time for learning after their part-time jobs, and furthermore, take responsibility for their learning.

There are few studies that have discussed effective online teaching methods for low academic achievers. In this area, the restructuring and translation of traditional computer software courses into e-learning has seldom been documented. Thus, this study redesigns a course in application software to integrate innovative teaching methods and learning technologies to help students learn and apply what they have learned. This study specifically explores potential effects of Web-based PBL and SRL on the development of low-academic-achievement students' skills in using packaged software.

AUTHENTIC ASSESSMENT

The traditional teaching approach regarded learners as passive recipients of information. Memorization of the content lectured by the teacher was the main goal of the instructional process. The assessment approach that accompanied this instructional approach was mainly operationalized by the so-called objective test, often standardized and with a true/false or a multiple-choice item format. Such assessments mostly do not seem to assess higher-order cognitive skills, such as problem solving, critical thinking, and reasoning (Segers, Dochy, & De Corte, 1999).

However, the objective tests are often given, even in computer software education. They could not evaluate students' practical skills, but also limit the potential application. Two innovative and practical teaching methods are applied in this study, so the traditional measurement is not appropriate to assess students' learning, particular in computer software education. More appropriate assessment methods for PBL would include reports, practical examinations, construction of concept maps, peer assessment, or oral presentations (Sonmez & Lee, 2003). In this regard, we measure students' skills of application software according to the correctness and completeness of their problem solving and artistry of their design instead of the memorization. The way we assess students' improvement in skills of application software is described in the "Measures" section.

PROBLEM-BASED LEARNING

Problem-based learning (PBL) is a teaching method that may engage students in authentic learning activities by using challenging problems in practice as a starting point, stimulus, and focus for learning (Barrows & Tamblyn, 1980; Boud, 1985; Barrows, 1985, 1986; Walton & Matthews, 1989; Boud & Feletti, 1991). PBL promotes student learning based on the need to solve problems. It not only emphasizes the learning in the subject area, but also provides opportunities for students to practice and apply knowledge and learned skills.

Davis and Harden (1999) indicate that PBL is one of the most effective teaching methods in recent 30 years. The probable reason is that the learning environment of PBL includes all kinds of factors currently known which can improve efficiency of learning, such as activity, cooperation, feedback, adaptability, and accountability. Albanese and Mitchell (1993) reveal that PBL helps students more in knowledge construction and reasoning skills compared with the traditional teaching approach. PBL may help students achieve learning goals, such as professional reasoning;

integration of scientific, academic, and professional knowledge; and lifelong learning skills (Dunlap, 2005).

Many Researchers examine PBL's positive impact on knowledge and skill acquisition and transfer, problem solving, attitudes and opinions about courses and programs, measures of performance, and self-directed learning (Norman & Schmidt, 1992; Albanese & Mitchell, 1993; Berkson, 1993; Vernon & Blake, 1993; Colliver, 2000; Davies, 2000). Furthermore, Polanco, Calderón, and Delgado (2004) point out that students in PBL groups attain significantly higher scores than students in a course from a control group composed of physics, mathematics and computer science.

In the domain of Information Science, Greening, Kay, Kingston, and Crawford (1996) consider that upper-level university students should have developed many key competencies. However, during teaching, they actually find many third-year students not having acquired necessary competencies from information science curriculum. PBL, as an alternative teaching method, demonstrates that it helps to improve students' key competencies. Yip (2001) points out that PBL can enhance competencies, both in professional and information systems education.

PBL is a flexible approach. It has demonstrated its working well with both small teams and large groups. However there might be disagreement whether PBL will be as effective, or even possible, for online learning. In this regard, Chanlin and Chan (2004) examine the effects of PBL in a Web-based approach. Results reveal that students in the PBL treatment group perform better than those from the control group. Therefore, it can be summarized that, in a Web-enabled learning environment, the effects of problem-based learning on students' skills of application software are positive, and higher than those without PBL.

SELF-REGULATED LEARNING

Zimmerman and Schunk (1989) define SRL in terms of self-generated thoughts, feelings, and actions, which are systematically oriented towards attainment of students' own goals. SRL is also defined as a learner's intentional efforts to manage and direct complex learning activities and is composed of three primary components, namely cognitive strategy use, meta-cognitive processing, and motivational beliefs (Kauffman, 2004).

Yang (1993) finds that students who are high self-regulatory skill users, as measured by a preexisting index, score significantly higher than their counterparts, low self-regulatory skill users, regardless of the level of control. Characteristics attributed to self-regulated persons coincide with those attributed to high-performance, high-capacity students, as opposed to those with low performance (or learning disabilities), who show a deficit in these variables (Roces & González Torres, 1998; Zimmerman, 1998; Reyero & Tourón, 2003).

Zimmerman and Martinez-Pons (1986) point out that students' frequency of self-regulated strategy use predicts a substantial amount of variance in their achievement test scores. Nota, Soresi, and Zimmerman (2004) indicate that the cognitive self-regulation strategy of organizing and transforming proves to be a significant predictor of the students' course grades in mathematics and technical subjects of high school, their subsequent average course grades, and examinations passed at the university.

As for the effects of SRL on computer software, Bielaczyc, Pirolli, and Brown (1995) incorporate self-explanation and self-regulation strategies in the attainment of the cognitive skill of computer programming. They find that their treatment group, which incorporates the self-regulation strategies of self-monitoring and clarifying comprehension

failures in conjunction with self-explanation strategies, outperforms a control group that did not have the benefit of instruction in these strategies. Their study implies that, in addition to knowledge acquisition strategies, students benefit from the incorporation of strategies which allow them to plan, monitor, and evaluate their understanding and strategy use (Bielaczyc, Pirolli, & Brown, 1995). In a similar vein, this study provides us an insight that SRL is appropriate to be applied in computer software education.

Researchers have consistently shown that self-regulation helps high achievers reach their potential (Risemberg & Zimmerman, 1992). It also makes a difference between failure and success for low achievers (Borkowski & Thorpe, 1994). Young (1996) identifies that low self-regulatory skill users are found to perform significantly lower in computer-based instruction that applied learner control of the sequencing. It is believed that SRL is effective to the low achievers, not only through face-to-face instruction, but also in computer-based or technology-based instruction. Learners who report more extensive use of SRL strategies show higher academic achievement than learners who use self-regulated learning strategies less often (Zimmerman & Martinez-Pons, 1986). In the same study, students' achievement could be predicted with 93% accuracy from reported use of SRL strategies.

Web-enabled learning environment is an interactive network system consisting of a variety of functions to support a virtual classroom to enhance the quality of teaching and learning activities (Poon, Low, & Yong, 2004). Students can access information at their convenience, are free to work at their own pace, and can revisit information that they find confusing and/or interesting (Lehman, Kauffman, White, Horn, & Bruning, 2001). Nevertheless, providing students with opportunities to integrate their knowledge through Web-enabled instruction may not be effective if they lack the skills needed to regulate their learning. Thus, strategies that prepare students for the rigors of learning at a distance and increase the probability of retention and success must be put into practice (Chang, 2005). Winnips (2000) suggests that self-regulation is particularly important when learning in World-Wide-Web-supported environments. In other words, it is even more critical for students to transform into self-regulated learners in Web-enabled learning. Previous studies have established that self-regulated skills can help foster learning from any instructional method (Weinstein, 1989; Zimmerman, 1990; Lindner & Harris, 1993; Ertmer, Newby, & MacDougall, 1996). To sum up, in the Web-enabled learning environment, effects of self-regulated learning on students' skills of application software are positive and performance higher than those without SRL.

PROBLEM-BASED LEARNING AND SELF-REGULATED LEARNING

In PBL, students work in collaborative groups to identify what they need to learn in order to solve problems. They engage in self-directed learning (SDL), apply knowledge they acquire to the problems, and reflect on what they have learned and the effectiveness of the strategies they have employed. That is, PBL is well-suited to help students become active learners as it situates learning in real-world problems and makes students responsible for their learning (Hmelo-Silver, 2004). PBL helps prepare students for lifelong learning by developing students' ability for self-directed learning and their meta-cognitive awareness (Dunlap, 2005).

PBL is a task-based approach that teachers can apply to support development of SRL. If PBL activities are designed carefully with teachers who provide appropriate modeling and scaffolding, they facilitate and necessitate SRL. PBL provides opportunities for self-directed learning by offering students choice, control of what to work on, how to work, and what products to generate. PBL

facilitates SRL because it places responsibility on the students to discover information, to coordinate actions and people, to monitor understanding, and to reach goals (Paris & Paris, 2001).

Hmelo-Silver (2004) points out that students' approaches to learn from problems are different qualitatively because of their degree of self-regulation. Ertmer et al. (1996) conducts a qualitative study of how veterinary students learned from problems. Low self-regulated learners have difficulty in adapting to the kind of learning required in problem-based instruction. They fluctuate according to their perception of the value of learning from problems. High self-regulated students value learning from problems and tend to focus on the problem analysis and reflection process. In contrast, low self-regulated students tend to focus on fact acquisition. Furthermore, the results of Ertmer et al. suggest that low self-regulated students may have difficulty in dealing with SRL demands of a PBL curriculum.

Combined training with self-regulatory and problem-solving strategies is effective for enhancing self-regulatory competences in solving mathematical problems (Perels, Gürtler, & Schmitz, 2005). However, there are very few studies that discuss the effects of PBL and SRL simultaneously, particularly through teaching Web sites. According to the literature reviewed , it is believed that the learning effect will be even stronger when teachers arouse students' interest, lead them to apply their skills and knowledge to solve problems, and encourage them to self-regulate their learning.

In this research, we hypothesize that students focusing on the aspects of problem-based learning variant with self-regulated learning would gain more development on the skills of application software than students studying the task without these co-existing treatments. We also hypothesize that learning effects of PBL and non-SRL group or non-PBL and SRL group are better than non-PBL and non-SRL group. That is, highest increase of development in skills of applying packaged software is expected in conditions wherein students are confronted with the situation of simulated problems and with self-regulated or self-directed learning style without teachers' pressure. Therefore, we propose the following: in a Web-enabled learning environment, the effects of PBL and SRL intervention on students' skills of application software are positive, and higher, than those without the combined intervention.

METHODS

Participants

Participants in this study are 106 fresh students taking a compulsory course of 'Packaged Software and Application' in an institute of technology in Taiwan. None of them major in the field of information or computer technology. However, in an institution for technological/vocational education, practical applications of technology are guiding principles (Tai, et al, 2003). Students are expected to spend more time and efforts in mastering a variety of technological skills, as compared to those in universities in Taiwan.

Course Setting

The course under study is a semester-long, two-credit-hour class, targeted at first-year college students from different majors. Students solve a series of authentic tasks by applying Microsoft Office (including Word, Excel, and PowerPoint).

Experimental Design and Procedure

In this study we explore whether students in the "Packaged Software and Application" course enhance their skills of application software via e-learning. Based on feedback from earlier research, we have re-designed the course and conducted a series of quasi-experiments to examine the effects of web-enabled PBL, SRL, and their combinations.

Figure 1. Expected effects of variation in instructional methods

	PBL	non-PBL
SRL	The most significant effect (C1 Group)	Medium effect (C3 Group)
non-SRL	Medium effect (C2 Group)	No difference (C4 Group)

The experimental design is a 2 (PBL vs. non-PBL) × 2 (SRL vs. non-SRL) factorial pretest-post-test design (see Figure 1). Students from four groups solve the same task but in different learning conditions. Participants are randomly assigned to one of the four experimental conditions in such a way that each condition contains 24 to 30 students. The PBL and SRL group (C1, n=30), PBL and non-SRL group (C2, n=25), non-PBL and SRL group (C3, n=24) are experimental groups, while non-PBL and non-SRL group (C4, n=27) is the control group. The sample size in this study is large enough to be tested in statistics.

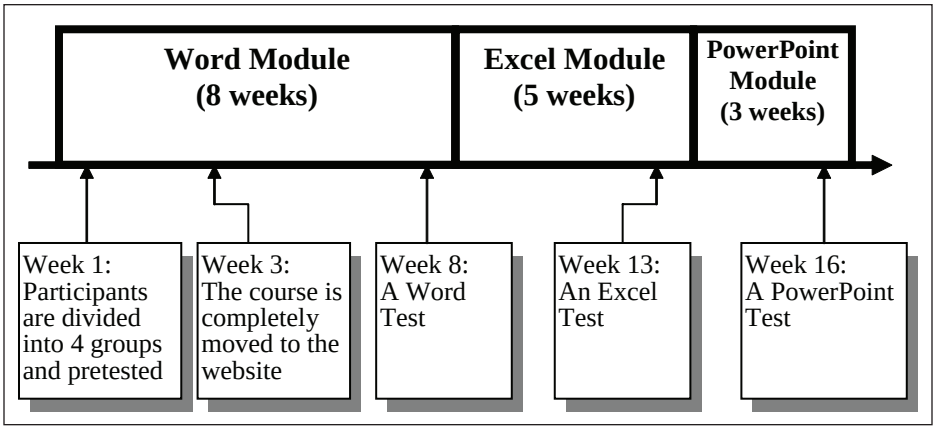
The course design in the present study consists of three subsequent modules: the Word module, the Excel module, and the PowerPoint module. A skill test is administered after the completion of each module. The first test is held during midterm examination (eighth week). The second test is held in the 13th week and the final one in the 16th week. A schedule of teaching of modules and skill tests is depicted in Figure 2.

In the beginning of a course, students are encouraged to adapt and learn via a course Web site. Teaching during this period is through a traditional classroom. A teacher records every session of the lecture first and then converts these lectures into a HTML file with flash, video, and voice. These HTML files are then loaded into the course Web site. Students can preview and review the course sessions on this course Web site. After three weeks, most of the coursework is moved onto the Web site. We help students to adapt to learning through the net and try lessen down a feeling of isolation. Within these three weeks, we adjust students' learning gradually and smoothly.

PBL Treatment

A teacher creates interesting, challenging, and authentic problem situations. In the first Word

Figure 2: Schedule of three modules and skill tests



module, students are required to apply for a job as Marketing Assistant in an online-game company. They are required to design and then build autobiographies and resumes by applying skills of application software that they have just learned. In the Excel module, students play roles as if they are employed by this same software company, and a marketing manager asks them to compare expenses resulting from different distribution channels. They have to survey and then complete a worksheet with some graphs to contrast differences between channels. Additionally, they must come up with a recommendation regarding the best combination of channels. In the last module of PowerPoint, they are promoted as Marketing Managers of higher rank. They are asked to develop a business proposal for a new online game. They have to present this proposal with visual aids to convince the managing director to enter in to a market. Therefore, a persuasive PowerPoint presentation is built at this stage.

A teacher demonstrates first how they could approach the situation and solve the problem with the help of a Web-based multimedia application. In addition to the teaching of skills of application software, similar situations and related applications are also discussed in the class. Latter, the teacher guides students in constructing their own models of problem-solving.

SRL Treatment

There is a SRL group in each class. Two SRL groups from the PBL class and non-PBL class are gathered in a classroom and a two-hour lecture is delivered discussing how to manage study time and regulate their learning. The lecture is given in an after-school course. Content of this SRL course is composed of following four processes addressed by Zimmerman, Bonner and Kovach (1996), that is,

- Self-evaluation and monitoring,
- Goal-setting and strategy planning,
- Strategy implementation and monitoring

Table 1. Teaching and learning activities in different experimental groups

Group	Teaching Activities	Learning Activities
C1	A teacher…	Students…
	• demonstrates how to solve authentic problems and discusses its potential applications.	• take on authentic tasks and learn by problem solving.
	• teaches SRL skills and urges students to study regularly.	• practice SRL and record learning behaviors every week.
C2	Teaching activities are the same as C1 but without SRL lectures.	Students experience authentic situations and solve the problems without extra requirements of SRL.
C3	A teacher…	Students…
	• converts his traditional way of teaching without any modification into an online format.	• receive traditional computer software course through internet.
	• teaches SRL skills and urges students to study regularly.	•practice SRL and record learning behaviors every week.
C4	Teaching activities are the same as C3 but without SRL lectures.	Students experience traditional style of teaching and do not deal with extra requirements of SRL, although teaching is conducted via Internet.

- Monitoring of the outcome of strategy.

Students are taught how to implement these four processes to become more regulated learners.

In addition to the two-hour lecture, students in SRL groups are required to prepare and read the textbook regularly before classes, and to review or practice the skills of application software they have learned after school. They are also required to record their learning behavior every week. Data is recorded on the course Web site instead of their notebooks in order to prevent falsification of records. A teacher cursorily examines students' records.

Treatments in four groups are illustrated and compared in Table 1.

Measures

To examine levels of change manipulated by variations in experimental conditions, we first measured students' skills of application software as a baseline before they entered the class. In the first week, students complete three Word documents as a pretest. We choose Microsoft Word for the pretest because almost every student in Taiwan learns Word before he learns any other software package.

The pretest grades show that computer skills of almost all are low. None of the participants are able to answer any of the pretest questions correctly. This confirms that all participants in the four groups had a little knowledge or a skill of software package. The difference in students' skills of application software at the beginning stage among the four groups was not statistically significant. Therefore, we assume that the students have equal or no computer skills before they take this course. In addition, none of them had any experience in taking a Web-based course. We then evenly and randomly divided the students into the four experimental groups.

Later, skill tests are administered at the end of each module. In this study, we applied two practical and authentic teaching methods. It is very important to measure students' practical skills in problem solving rather than the memorization. In this regard, the lecturer simulated practical problems for students to solve to evaluate their enhancement of skills of application software. For example, students are required to complete a worksheet with some graphs and tables to compare the market share with all competitors. They have to apply several functions to compute the numerals and sort them. Students' grades come from their correctness and completeness of problem solving, and the artistry of their designs. Students will get high grades if they completely solve the problems with exquisite designing.

Before testing, students are assigned random seats. All students are tested at the same time. Every test consists of five to seven questions. A teacher grades and records the results immediately after each test. A surrogate representing skills in application software is averaged from the scores of these three tests. Finally, the enhancement of skills in application software of a module is the result of one's grade minus pretest grade. We test differences in the enhancement of the skills of application software under different conditions.

Results

To examine levels of change manipulated by variants in experimental conditions, we first measured students' skills using application software as a baseline. The pretest confirmed that all participants from four groups had a little or no knowledge and skills in packaged software. The difference in students' skills of application software at the beginning stage among four groups was not statistically significant. In addition, none of them had any experience in taking a Web-based course. We then randomly divided the students into four experimental groups.

Independent sample t-test was used to compare improvement of grades between PBL and non-PBL teaching methods. As shown in Table 2, the improvement in students' grades for Word, Excel and PowerPoint modules in PBL class (67.09) was on an average higher than that in the non-PBL class (56.75). This is also true if one takes the test scores of each module into account separately. Therefore, effects of Web-based PBL on students' skills of application software are positive, and higher than those who do not receive PBL.

Results from Table 3 show that the average grade for Word, Excel and PowerPoint modules in SRL group (66.01) is higher than that in the non-SRL group (58.08). This is also true if one takes test scores of each module into account separately. Thus, the effects of web-based SRL on students' skills in using application software are positive, and higher than those without SRL intervention.

Finally, data from Table 4 shows that combination of PBL and SRL intervention a group results in the highest grades among four groups. The improvement of skills using application software in C1 is significantly higher than C3 and C4, and also higher than C2, though insignificant. Therefore, we conclude that the effects of Web-based PBL and SRL intervention on students' skills in using application software are positive, and higher than those who do not receive PBL or/and SRL.

For those teachers who wish to stick to traditional methods of teaching, directly transforming their teaching material into electronic form may not be a fruitful approach. Students

Table 2. Independent samples t-test: Improvement of grades

		N	Mean	S. D.	F	t-value	df	P
Word	PBL	55	64.29	11.063	2.604	3.147	104	.002**
	non-PBL	51	56.84	13.269				
Excel	PBL	55	64.98	17.948	24.992	3.809	104	<.001**
	non-PBL	51	46.43	30.943				
PowerPoint	PBL	55	72.00	11.617	.029	2.186	104	.031**
	non-PBL	51	66.98	12.016				
Average	PBL	55	67.09	9.779	5.430	4.722	104	<.001**
	non-PBL	51	56.75	12.673				

Table 3. Independent samples t-test: The improvement of grades

		N	Mean	S. D.	F	t-value	df	P
Word	SRL	54	63.11	11.647	.102	2.018	104	.046**
	non-SRL	52	58.21	13.325				
Excel	SRL	54	62.37	23.770	2.483	2.554	104	.012**
	non-SRL	52	49.50	28.021				
PowerPoint	SRL	54	72.54	12.462	.222	2.649	104	.009**
	non-SRL	52	66.52	10.831				
Average	SRL	54	66.01	11.305	.769	3.473	104	.001**
	non-SRL	52	58.08	12.195				

Table 4. One-way ANOVA (Analysis of Variance): Average of the improvement of grades.

Dependent Variable		(I) Group	(J) Group	Mean Difference (I-J)	Std. Error	Sig.
Average	Scheffe	1	2	5.862	2.910	.262
			3	8.436(*)	2.943	.047
			4	17.064(*)	2.851	.000
		2	1	-5.862	2.910	.262
			3	2.574	3.071	.872
			4	11.202(*)	2.983	.004
		3	1	-8.436(*)	2.943	.047
			2	-2.574	3.071	.872
			4	8.628(*)	3.015	.048
		4	1	-17.064(*)	2.851	.000
			2	-11.202(*)	2.983	.004
			3	-8.628(*)	3.015	.048

from the controlled group (C4) received lowest grades among four groups, and differences in grades among them are significant (see Table 4). It is suggested that teachers should redesign their courses and then adopt new instructional methods and technologies to fully exploit benefits of deploying Web-based learning environments.

DISCUSSION

This study provides valuable insights and sheds some lights on new, effective practices that can be used by schools (particularly vocational schools), scholars, and teachers planning to implement or presently engaged in implementing e-learning environments. The implications of this study consist of three-fold:

Firstly, the PBL teaching method was found to play a consistently positive role in enhancing students' skills of using application software (see Table 2). There are significant differences between PBL and non-PBL classes, either in case of individual module tests or in an average of the three module tests. It was demonstrated that PBL is good for computer software education in general and e-learning in particular. Teachers could redesign their courses by simulating meaningful and interesting business situations, and, thus, engaging students' imaginations and interest to solve challenging problems.

Secondly, evidence also supports that the second teaching method based on SRL also enhances students' skills of using application software (see Table 3). There are significant differences between SRL and non-SRL groups on three tests and in average. Interestingly, as we focused on the variation of P-value related to each module, it became clear that the effects of SRL increased over time. This suggests that the effects of SRL among low achievers can be significantly improved, even by a limited amount of intervention, such as two-hour lecture regarding SRL at the beginning of teaching program and later by students' monitoring their own learning.

Finally, this study found some support for the effectiveness of a combination of instructional

methods. As shown in Table 4, results show that the effects of a combination of PBL and SRL intervention on students' skills of application software are positive and higher than those who do not receive PBL or/and SRL, although the difference between C1 and C2 is not statistically significant.

This study contributes to e-learning practices in three different ways: (a) this study specifies how teachers can engage students in improving learning under authentic conditions. At the same time, teachers help students to regulate their learning by applying PBL and SRL instructional methods in a Web-based learning environment; (b) this study is one of the first attempts to explore learning effects of various combinations of PBL, SRL, and Web-based learning; and (c) this empirical study provides evidence that skills of low academic achievers, for using application software can be improved through e-learning.

Teachers face tremendous challenges in implementing e-learning among relatively low academic achievers. For example, Internet addiction is common, and it is not immediately clear how to focus students' attention and improve their learning in a Web-based environment without teachers' on-the-spot monitoring. This study proposes an effective approach in this kind of e-learning environment.

REFERENCES

Albanese, M. A., & Mitchell, S. (1993). Problem-based learning: A review of literature on its outcomes and implementation issues. *Academic Medicine*, *68*(1), 52–81.

Azevedo, R., & Cromley, J.G. (2004). Does training on self-regulated learning facilitate students' learning with hypermedia? *Journal of Educational Psychology, 96*(3), 523-535.

Barrows, H. (1985). *How to design a problem-based curriculum for the preclinical years*. New York: Springer.

Barrows, H. (1986). A taxonomy of problem-based learning methods. *Medical Education*, *20*(6), 481-486.

Barrows, H., & Tamblyn, R. (1980). *Problem-based learning.* New York: Springer.

Berkson, L. (1993). Problem-based learning: Have the expectations been met? *Academic Medicine*, *68*(1), 79-88.

Bielaczyc K, Pirolli P., & Brown A. (1995). Training in self-explanation and self-regulation strategies: investigating the effects of knowledge acquisition activities on problem solving. *Cognition and Instruction*, *13*(2), 221–252.

Borkowski, J. G., & Thorpe, P. K. (1994). Self-regulation and motivation: A life-span perspective on underachievement. In D. H. Schunk & B. J. Zimmerman (Eds.), *Self-regulation of learning and performance* (pp. 45-73). Hillsdale, NJ: Lawrence Erlbaum Associates.

Boud, D. J. (1985). Problem-based learning in perspective. In D. J. Boud, (Ed.), *Problem-based learning in education for the professions* (pp. 13-18). Sydney: Higher Education Research and Development Society of Australasia.

Boud, D., & Feletti, G. (1991). *The challenge of problem based learning.* London: Kogan Page.

Bruer, J. T. (1993). *Schools for thought: A science of learning in the classroom*. Cambridge, MA: MIT Press.

Cennamo, K. S., Ross, J. D., & Rogers, C. S. (2002). Evolution of a Web-enhanced course: Incorporating strategies for self-regulation. *Educause Quarterly*, *25*(1), 28-33.

Chang, M. M. (2005). Applying self-regulated learning strategies in a Web-based instruction - An

investigation of motivation perception. *Computer Assisted Language Learning*, *18*(3), 217-230.

Chanlin, L. J., & Chan, K. C. (2004). Assessment of PBL design approach in a dietetic Web-based instruction. *Journal of Educational Computing Research*, *31*(4), 437-452.

Colliver J. A. (2000). Effectiveness of problem-based learning curricula: Research and theory. *Academic Medicine*, *75*(3), 259-266.

Dabbagh, N., & Kitsantas, K. (2005). Using Web-based pedagogical tools as scaffolds for self-regulated learning. *Instructional Science*, *33*(5–6), 513–540.

Davies, P. (2000). Approaches to evidence-based teaching. *Medical Teacher*, *22*(1), 14-21.

Davis, M. H., & Harden, R. M. (1999). Problem-based learning: A practical guide. *AMEE Education Guide*, 15, http://www.amee.org/index.asp?tm=43

Dunlap, J. C. (2005). Changes in students' use of lifelong learning skill during a problem-based learning project. *Performance Improvement Quarterly*, *18*(1), 5-33.

Ertmer, P. A., Newby, T. J., & MacDougall, M. (1996). Students' approaches to learning from case-based instruction: The role of reflective self-regulation. *American Educational Research Journal*, *33*(3), 719-752.

Greening, T., Kay, J., Kingston, J. H., & Crawford, K. (1996). Results of a PBL trial in first-year computer science. *The 2nd Australasian Conference on Computer Science Education* (pp. 201-206). New York: ACM Press.

Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review*, *16*(3), 235-266.

Kauffman, D. F. (2004). Self-regulated learning in Web-based environments: Instructional tools designed to facilitate cognitive strategy use, metacognitive processing, and motivational beliefs. *Journal of Educational Computing Research*, *30*(1 & 2), 139-161.

Lehman, S., Kauffman, D., White, M., Horn, C., & Bruning, R. (2001). Teacher interaction: Motivating at-risk students in Web-based high school courses. *Journal of Research on Computing in Education*, *33*(5), http://www.iste.org/inhouse/publications/jrte/33/5/lehmans.cfm?Section=JRTE_33_5

Lindner, R. W., & Harris, B. (1993). Teaching self-regulated learning strategies. In M.R. Simonson & K. Abu-Omar, (Eds.), *Proceedings of selected research and development presentations at the annual conference of the Association for Educational Communications and Technology* (pp. 641–654). Ames, IA: Instructional Resources Center, Iowa State University.

Norman, G.., & Schmidt, H. (1992). The psychological basis of problem-based learning. *Academic Medicine*, *67*(9), 557-565.

Nota, L., Soresi, S., & Zimmerman, B. J. (2004). Self-regulation and academic achievement and resilience: A longitudinal study. *International Journal of Educational Research*, *41*(3), 198–251.

Paris, S. G.., & Paris, A. H. (2001). Classroom applications of research in self-regulated learning. *Educational Psychologist*, *36*(2), 89-101.

Perels, F., Gürtler, T., & Schmitz, B. (2005). Training of self-regulatory and problem-solving competence. *Learning and Instruction*, *15*(2), 123-139.

Polanco, R., Calderón, P., & Delgado, F. (2004). Effects of a problem-based learning program on engineering students' academic achievements in a Mexican university. *Innovations in Education & Teaching International*, *41*(2), 145-155.

Poon, W. C., Low, K. L. T., & Yong, D. G. F. (2004). A study of Web-based learning (WBL) environment in Malaysia. *The International Journal of Educational Management, 18*(6), 374–385.

Reyero, M., & Tourón, J. (2003). *El desarrollo del talento: La aceleración como estrategia educativa* [*The development of talent: Acceleration as an educational strategy*]. A Coruña: Netbiblo.

Risemberg, R., & Zimmerman B. J. (1992). Self-regulated learning in gifted students. *Roeper Review, 15*(2), 98-101.

Roces, C., & González Torres, M. C. (1998). Capacidad de autorregulación del aprendizaje [*Ability to self-regulate learning*]. In J. A. González Pienda & J. C. Núñez (Eds.), *Dificultades de aprendizaje escolar* (pp. 239-259). Madrid: Pirámide/Psicología.

Segers, M., Dochy, F., & De Corte, E. (1999). Assessment practices and students' knowledge profiles in a problem-based curriculum. *Learning Environments Research, 2*(2), 191–213.

Sonmez, D., & Lee, H. (2003). Problem-based learning in science, http://www.eric.ed.gov/ERICWebPortal/recordDetail?accno=ED482724

Tai, C. F., Chen, R. J., & Lai, J. L. (2003). How technological and vocational education can prosper in the 21st century? *IEEE Circuits & Devices Magazine, 19*(2), 15-51.

Vernon, D., & Blake, R. (1993). Does problem-based learning work? A meta-analysis of evaluative research. *Academic Medicine, 68*(7), 550-563.

Walton, H., & Matthews, M. (1989). Essentials of problem-based learning. *Medical Education, 23*(6), 542-558.

Weinstein, C. (1989). Teacher education students' preconceptions of teaching. *Journal of Teacher Education, 40*(2), 53-60.

Williams, S. M. (1993). Putting case-based learning into context: Examples from legal, business, and medical education. *Journal of Learning Sciences, 2*(4), 367-427.

Winnips, K. (2000). *Scaffolding-by-design: A model for WWW-based learner support*, Enschede: University of Twente Press.

Wu, T. Y. (2000). Integrative curriculum planning in technological and vocational education in Taiwan, Republic of China. http://www.eric.ed.gov/ERICWebPortal/recordDetail?accno=ED450230

Yang, Y. (1993). The effects of self-regulatory skills and type of instructional control on learning from computer-based instruction. *International Journal of Instructional Media, 20*(3), 225-241.

Yip, W. (2001). Utilisation of the problem-based learning approach facilitated by information technology to teach information on systems development. http://citeseer.ist.psu.edu/535807.html

Young, J. D. (1996). The effect of self-regulated learning strategies on performance in learner controlled computer-based instruction. *Educational Technology Research and Development, 44*(2), 17-27.

Zimmerman, B. J. (1990). Self-regulated learning and academic achievement: An overview. *Educational Psychologist, 25*(1), 3-17.

Zimmerman, B. J. (1998). Developing self-regulation cycles of academic regulation: An analysis of exemplary instructional model. In D. H. Schunk & B. J. Zimmerman (Eds.), *Self-regulated learning: From teaching to self-reflective practice* (pp. 1-19). New York: Guilford.

Zimmerman, B. J., Bonner, S., & Kovach, R. (1996). *Developing self-regulated learners: Beyond achievement to self-efficacy*. Washington, DC: American Psychological Association.

Zimmerman, B. J., & Martinez-Pons, M. (1986). Development of a structured interview for assessing student use of self-regulated learning strategies. *American Educational Research Journal, 23*(4), 614-628.

Zimmerman, B. J., & Schunk, D. H. (1989). *Self-regulated learning and academic achievement: Theory, research, and practice*. New York: Springer-Veri

This work was previously published in International Journal of Distance Education Technologies, Vol. 6, Issue 3, edited by Q. Jin, pp. 69-84, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.18
Globalising Software Development in the Local Classroom

Ita Richardson
University of Limerick, Ireland

Sarah Moore
University of Limerick, Ireland

Alan Malone
Siemens Corporate Research, USA

Valentine Casey
University of Limerick, Ireland

Dolores Zage
Ball State University, USA

ABSTRACT

In the dynamic global economy that exists today the operation and structure of organisations have had to adapt to the reality of the information revolution which has taken place. This has been the case within the software industry where global software development (GSD) has become a popular strategy and software development has become a globally sourced commodity. Given the requirement for graduates to operate in this type of environment, we as educators considered how our teaching methods could be developed and enhanced to instil GSD competencies within our graduates. We provided students with the opportunity to take part in a learning experience that transcended geographical and institutional boundaries, giving them first-hand experience of working within globally distributed software teams. Two separate projects were undertaken.

One was with Siemens Corporate Research which was part of a larger project. The focus of this project was the shadowing of the development of an actual geographically distributed software product. The second project was carried out in collaboration with Ball State University, and the focus of this endeavour was virtual team software testing. Extensive qualitative research was undertaken on the data provided by the students. We identified three specific forms of learning which had taken place: (1) pedagogical, (2) pragmatic, and (3) the achievement of specific globally distributed competencies. Our findings would confirm that mimicking real-work settings creates the possibility of giving rise to the range of learning benefits that are associated with truly problem-based learning environments.

INTRODUCTION

This chapter explores the reality of the software industry today, which is becoming more virtual and globally distributed in its methods of operation. It discusses the educational implications of these strategies and how they impact graduates. It looks at what measures can be taken to prepare students to operate in this dynamic and virtual environment. It outlines two projects in which masters students participated in that transcended geographical and institutional boundaries. The projects and the students' experiences were researched and analysed. The results, which are presented here, demonstrate the benefits associated with utilising a hands-on, truly problem-based learning environment.

GLOBAL SOFTWARE DEVELOPMENT

GSD has given rise to the implementation of new types of development teams and project structures within organisations. In many software development organisations, teams are no longer local, but operate within a virtual team environment. As a result they are fundamentally different in their structure and modus operandi to those of a single site team. For educationalists, the emergence of new team and organisational structures require that graduates from software engineering courses be made familiar with these new methods of operation.

The number of organisations employing virtual,[1] team-based globally distributed software development strategies continues to increase (Powell, Piccoli, & Ives, 2004). GSD in essence allows distributed teams to split up the tasks of a project and distribute them as separate jobs (Grinter. Herbsleb, & Perry, 1999). This allows development decisions about each project task to be made with a degree of independence (Herbsleb & Grinter, 1999). However, managing this type of team is not a straightforward endeavour. Some of the difficulties encountered include the problems of understanding requirements and the testing of systems (Toaff, 2002). These difficulties are compounded by cultural and language differences, lack of communication, distance from the customer, different process maturity levels, testing tools, standards, technical ability, and experience. These issues are further augmented by the lack of "trust-building" communication techniques. Trust is important for software development teams to work together successfully, and it is harder to establish trust within virtual teams than it is with local teams (Robey, Khoo, & Powers, 2000), rising from the fact that face-to-face communication methods that can build trust are generally not present in a distributed team (Pyysiainen, 2003). Equally, established trust gained from co-located experiences can deteriorate over time in a distributed setting (Casey, 2007). To address these substantial issues, project management must change from the traditional to the virtual for a GSD strategy to be successfully implemented.

Making GSD Work

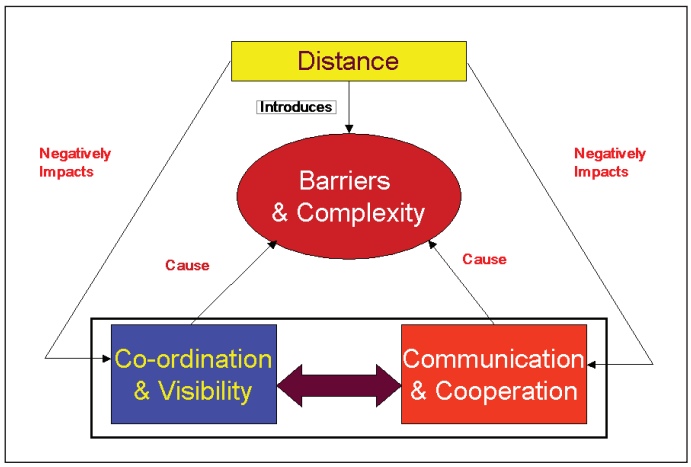
As the GSD team is dispersed across geographical locations, it operates differently to the co-located team. Regardless of the strategy implemented, distance is responsible for the introduction of barriers and complexity. These directly impact on the management and operation of the GSD team. However, other related factors also come into play. Coordination, visibility, communication, and co-operation are all negatively impacted by distance, and if their impact is not identified and correctly managed, they can produce further barriers and complexity within a project (see Figure 1).

Effective *coordination* is a key element in the success of a geographically distributed software development project. This includes undertaking realistic project planning and risk evaluation given the specific requirements of the GSD environment. Work must be partitioned between sites based on the technical needs of the project and on the capabilities and experience of the team members at different geographical locations. Furthermore, there is a requirement for the effective utilisation of technology across sites and between team members. Procedures should be put in place to facilitate and monitor the level of cooperation between team members in all locations. These should also allow for the identification and addressing of problems if and when they arise.

Visibility is another important factor in the successful operation of the global team. It is important for management to ensure that roles and responsibilities are clearly articulated. Each team member should be informed of work product requirements and due dates. This requires effective reporting schedules and a visible reporting strategy. There is a requirement for continuous visibility into the team's activities and operation at all locations. Team members should also be aware of the management structure within the team across all sites.

Communication issues should not be allowed to become a barrier to the successful operation of a distributed software development team. This necessitates the development and implementation of a common vocabulary for all aspects of the project. Effective communication tools, which are utilised and understood by all team members, should be provided. Communication between remote team members is normally electronic and asynchronous in nature with limited opportunities for synchronous contact, depending on the time zone difference between locations. The

Figure 1. Factors in virtual development (Source: Casey, 2007)



global team normally operates in a multicultural and multilingual environment, which may cross organisational boundaries (DeSanctis, Staudenmayer, & Wong, 1999).

Cooperation within global teams is essential to achieve a successful implementation of a GSD strategy. In distributed teams there are limited opportunities for one-to-one contact between remote colleagues. Project managers need to consider how team relationships can best be developed and fostered. The specific implications of cultural diversity on the project operation must be ascertained, monitored, and addressed. All team members should be given cultural training. The identification of a relevant subject matter expert (SME) is important, and team members should be informed who this is. Once identified the SME must be willing and able to provide the required support to local and remote colleagues.

EDUCATING THE GRADUATE

One of our tasks as software engineering educators is to produce graduates for the software market, and with the increase in GSD worldwide, it is incumbent on us as educators to provide these graduates with a background that allows them to work in the global environment.

To operate in the global environment students need to be aware of the factors involved in GSD. Through research carried out by the authors (Casey, 2007; Casey & Richardson, 2005), specific competencies that are needed have been identified. From the perspective of the graduate, these can be broadly divided into two categories: (1) competencies required by both the software developer and manager and (2) those required more significantly by the software development manager (see Table 1).

Competencies Required by the Software Developer and Software Manager

When working in the global environment, *communication and language* takes on greater importance than when working in a local team. English has become the international business language and is extensively used in GSD teams. However, many co-workers will not have English as their first language, therefore, misunderstandings can occur, as there may be different dialects of English and/or a number of regional accents in use (Kiel, 2003). A further consideration is the *use of communication tools* which must be done correctly and in such a way that it benefits and not hinders the project. Global teamwork is often

Table 1. Competencies required by the software graduate

Software Developer and Manager	Software Development Manager
Communication and language	Visibility and coordination
Use of communication tools	True cost of GSD
Culture	Technology transfer and knowledge management
Temporal issues	Roles, responsibilities, and competencies management
Cooperation	Partitioning of work
Software process	Management of fear and trust
Use of process tools	Motivation
Reporting schedule	“Teamness”
	Risk management

carried out through asynchronous communication tools like e-mail (Boland & Fitzgerald, 2004). This is not the environment in which graduates or locally experienced software developers are used to working—they need to learn and understand how to use language, communication, and communication tools efficiently and effectively in the global development team.

An advantage of having a variety of *cultures* involved in one team is diversity and bringing diversity together to work toward a common goal can increase the innovation within a project (Ebert & De Neve, 2001). However, those working within global software teams must learn to understand and appreciate cultural diversity. Cultural differences include values, language, approach to authority, concepts of space, regard for material goods, and time keeping (Carmel, 1999; McDonough, Kahn, & Barczak, 2001). Cultural differences, unless understood by all involved, can have a negative impact on the operation of the global team.

Those working on GSD teams should also have an appreciation of the *temporal issues* that may arise. A strategy which has been employed in establishing global teams has been to have team members in two or three time zones around the globe. For individuals working on such teams, this requires them to ensure that their working hours overlap somewhat with their remote colleagues (Casey & Richardson, 2004; McDonough, Kahn, & Barczak, 2001). Furthermore, responses to communications need to be made in a timely manner. Response delays can cause frustration and decreased motivation among team members. These difficulties can be exasperated if the person required is remotely situated (Herbsleb & Grinter, 1999).

As mentioned previously, cooperation is required for a global team to be successful. An advantage for GSD is that, given the diversity of global teams, the acquired mix of skill sets and life experiences within a team can result in improved coordination among GSD team members (Ebert & De Neve, 2001). However, geographical distance reduces the level of informal contact that can help build better working relationships, and consequently, support cooperation (Herbsleb, Mockus, Finholt, & Ginter, 2000). In the global environment, and without this level of informal contact, software developers need to develop cooperation between themselves and local and remote team members. They need to be able to modify their locally focused competencies to work successfully in the global environment.

With the increase in use, internationally, of software *processes* and the implementation of models such as the CMMI[2sm] and ISO15504 among others, software developers are becoming more au fait with the implementation of processes. However, processes in the global environment are typically different to those in the local co-site operation. They need to be set up with global information flow, global reporting, and the effective division of labour in mind. For example, if a wiki[3] is being used, project members need to continually keep it updated as they will not have the informal contact which, often in the local situation, allows team members to know what is happening on a project. *Process tools* can help local teams in managing processes such as configuration management, risk management, and testing. In the global environment, they can ensure that even though team members are not discussing issues on a face-to-face informal basis, processes can be efficiently controlled. Along with the technical use of such tools, team members need to understand the tools' effectiveness and requirements.

As mentioned previously, visibility of project activities is important. Culture has an effect on whether overruns in project tasks are reported. Furthermore, teams may be set up so that the project manager is based in one location only. To ensure visibility and to minimise difficulties arising through differences across global teams, it is important that *regular scheduled reports* are made to all team members. These reports should

be timely and accurate and submitted by all team members.

Competencies Required by the Software Manager

In addition to the competencies required by the software developer, the software manager working in the global environment must be able to ensure that the team for which he/she has responsibility can work effectively. This requires them to bring additional competencies and requirements to the team.

They are responsible for ensuring the *visibility* of, for example, tasks, team members, project structure, and reporting structures across the global team. They are likely to be co-located with some team members and may never meet face to face with all their global team. To coordinate the work of a global team in this environment may cause difficulties if they are not dealt with in an effective manner.

Companies are not always aware of the "true cost" of GSD, and indeed there is no "silver bullet" that can provide them with that information. However, it is something about which the software development manager should be aware. Negative influences within a global team are high risk and can cause decreases in productivity. For example, lack of communication can reduce the effectiveness of a project and can occur easily through the misuse of communication tools (Casey & Richardson, 2004).

Moving from a local co-site team structure to a global team environment requires *technology transfer* and *knowledge management*. The global software development manager has to understand how they can do this effectively. If new project members are being employed at an off-site location (normally global), how can these people be given the same transfer of technology and knowledge as would happen in a local environment where there are informal discussions and the availability of experts? This needs to be correctly managed with an effective training strategy put in place.

The global project manager must be able to decide upon the *roles and responsibilities* within the project team so that the team can work successfully together. In doing this, they must recognise the variety of competencies that individuals are bringing to the project, and they need to *manage these competencies* for the project's benefit. Considering the lack or limited opportunity for informal contact between teams, the impact this has in a geographically dispersed group is extremely important, and the failure to be cognisant of these factors can cause the team's effectiveness to be reduced (Casey, 2007).

Once the project team has been structured, the project manager must be able to *partition the work* in such a manner that the team's effectiveness is maximised. In some cases, the partitioning can be based around the software development life cycle, where the team members at each location undertake one element of the life cycle, for example, development or testing, while control is maintained at a central location. Another model, which is becoming increasingly popular, is to set up virtual teams, where team members work as one team even though they are located globally. This is not an easy task for the project manager to undertake (Nidiffer & Dolan, 2005).

Within countries, such as Ireland, who no longer are considered a low-cost economy, people who work on project teams can perceive the implementation of GSD as a step in the transfer of their jobs to low-cost economies in Eastern Europe and the Far East. This can, in turn, lead to an element of *fear and mistrust* among the project team members, which would not exist in a local team. The role of the software development manager is to understand these fears and address them where possible. The objective is to ensure that the team does not become de-motivated about the work they are expected to do. Maintaining a level of *motivation* is an important task that should not

be overlooked by the manager. This in turn will facilitate the development of "teamness" within the group, which has to be built across communities and geographical locations. It cannot be done in the informal manner in which local teams can gel together, through casual meetings on the corridor, or through more company-based structured activities, for example, starting a sports and social club (Linnane & Richardson, 2006).

The GSD manager has to consider all the risks involved in setting up and operating their global software development team. While *risk management* is an important element within a co-site software process and it is a level three process area of the Capability Maturity Model Integrated (CMMI[4sm]). The additional risks involved in managing a GSD team are significant and need to be specifically acknowledged and addressed.

Providing GSD Competencies to the Graduate

Given the requirements emerging from GSD, the graduate of the 21st century will not only be required to have competencies that are useful in the local development environment; they must also, for the success of GSD initiatives, demonstrate those other competencies that allow them to operate effectively in the global environment. As educators, we considered how our teaching methods could be developed and enhanced to instil these GSD competencies within the graduate. Our experience has shown that we must be prepared to introduce experiential learning to the student through extending education across international boundaries, cooperating globally between educational institutions and others, thus giving the graduate an educational experience which could not be provided in the local classroom. In the light of these principles the projects described in subsequent sections were undertaken.

When introducing an experiential dimension to student learning, the features of the learning environment can become different and ultimately more beneficial than they might otherwise be. By creating a learning experience that mimics the dynamics, pressures, and puzzles of "real-life" GSD environments, we present a learning context that has the potential of addressing some of the problems that may be associated with more conventional learning modes. In the light of these principles the projects described in the section, *Global Software Development in the Classroom,* were undertaken.

EDUCATIONAL OBJECTIVES ASSOCIATED WITH INTERVENTION

The GSD team structure and process created a problem-based learning environment for students, the benefits of which are well documented, for example, in Savin-Baden (2004) and Barrett (2005). In particular, the benefits of group work in which several students work together to solve a problem or to achieve a task have been evidenced in the educational literature. Joy (2005) has recently shown the particular way in which group working experiences can be beneficial in the context of a computer science curriculum by highlighting learning and experiential outcomes such as the application of knowledge, motivation, advanced cognitive competencies (or deep learning), and self-direction. The importance of ideological development is also relevant in the way that Reynolds (1994) has suggested. It is argued that working on real problems in a group setting can prepare students to become more collaboratively orientated and subsequently more willing to participate in collective effort.

By setting up virtual teams for software engineering students, we also argue that it is possible to inject several more specific learning benefits than the ones that have been briefly outlined. By learning from direct and relevant experience, students' own sense about the plausibility of their learning environment becomes stronger. Levitt and March (1988) and Luo and Peng (1999) are

among many who highlight the importance of real-world experience that is applied in the context of a genuinely and realistically problematic task. Being part of a virtual team in which other pedagogical support is also available can bestow a reflective dimension on learning that can allow students to learn more about themselves through reflection of the ways in which they engaged with the process.

Reflective practice in a "safe" environment is a process that can have both psychological and pedagogical benefits. It dismantles defensive reactions to problems and errors and allows participants to examine them in order to enhance understanding and cognitive command of their features. Indeed, defensiveness in many work environments often prohibits the capacity for learning in ways that are damaging to individuals, teams, and their organisations (Tjosvold, Yu, & Hui, 2004). Setting up early professional experiences in which teams can effectively reflect on and analyse their mistakes as well as their successes in a supported learning environment is one of the beneficial features that we hoped would prevail in this particular learning innovation.

We hypothesised that this type of innovation would create professional precedents that would make it more likely that participating students would engage professionally in more learning-oriented ways. This innovation went beyond just simulating an environment; it allowed students to become contributory members of GSD teams. This could have also created insights about how a sense of responsibility to deliver their part of the task is experienced, and how this sense of responsibility interacts with their learning. These are the kinds of integrated insights that might not otherwise have been possible using other modes of learning. We argue then that the sense of engagement and responsibility was automatically enhanced. These dynamics of learning are ones that have been highlighted as crucial to the creation of self-directed, responsible learners in higher education (e.g., Hwarng, 2001).

These real GSD experiences also have the potential to raise student awareness of the importance of related competencies that they might not otherwise have considered important. Such competencies include those related to communication and global interaction; language and intercultural awareness; and functioning effectively as part of a team (self awareness; creativity; formulation and implementation processes; problem solving; time management; and task completion).

By designing the learning experience described in this chapter, we hoped that we would be able to give rise to the learning benefits outlined previously. The following section captures the insights that participating students provided when asked to reflect on their experiences as part of the GSD team to which they had been assigned.

GLOBAL SOFTWARE DEVELOPMENT IN THE CLASSROOM

The University of Limerick (UL) delivers a Master of Science course in software engineering. Participants have previously completed a related undergraduate degree course. To date, one-third of the students are working in software development and are part-time. The remaining students are pursuing the degree in full-time mode—85% of these have no prior industrial experience, having commenced the degree directly after completing their undergraduate studies. When pursuing the MSc degree, students study a variety of modules which include software engineering quality, software engineering requirements, human computer interaction, software design, software development paradigms, software evolution, software engineering system design, and software engineering fundamentals. These modules account for 60% of course credits, and the remaining 40% of course credits are obtained through a final dissertation, encompassing a major study undertaken by the student.

As educationalists, and given the importance and growth of GSD within both multinational and small to medium-sized companies in Ireland, we strive to expose graduates from this course to GSD. We do this in some cases through lecturing on the topic. However, as we aim to maximise the pedagogic experience for students we have implemented two specific GSD projects for the course, where students were given the opportunity to participate in situated GSD projects. One of these projects, Ard na Croise,[5] is carried out in conjunction with Siemens Corporate Research, USA and has been running for 2 years. In each year, there has been one team of five people from the MSc class involved. Of these, nine had no prior industrial experience. The other project, Ainm an Eolaithe,[6] is a collaborative project with Ball State University, USA and has run during one course, with 12 full-time students participating in UL. Ten of these students had no prior industrial experience. Four of these students also participated in Ard na Croise.

In implementing these projects we aimed to equip students with the competencies required for software developers and managers as given in Table 1, while additionally expecting that at least some of the competencies required by software development managers in working with globally dispersed teams will be attained by the participants.

PROJECT DESCRIPTIONS

Ard na Croise with Siemens Corporate Research

Siemens Corporate Research, Inc. (SCR) has been conducting research aimed at developing a better understanding of the issues and impact of various practices with respect to GSD. By making this investment SCR hopes to establish itself as a GSD centre of excellence that can assist other Siemens operating companies to cope with the unique complexities of conducting GSD. SCR, in conjunction with Harvard Business School, University of Limerick, Carnegie Mellon University, Monmouth University, Technical University of Munich, the Indian Institute of Information Technology Bangalore, University of Toronto, Penn State University, and the Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) in Brazil have set up an experimental geographically distributed development project (GSP) using student teams to study associated issues. The project ran for 2 years, with different students involved in the teams each year within the participating institutions.

Organisation of the Global Studio Project

The project simulates a hub-and-spoke model. The hub is the central team at SCR in Princeton orchestrating the effort, and the spokes are the remote teams consisting entirely of participating students. The central team is responsible for project management, requirements, architecture, testing, and integration of the system. The remote teams are responsible for design, development, and unit tests for defined work packages that correspond to code modules or subsystems defined by the central team. Interactions between the central and remote teams are managed by the role of a *supplier manager*. There is a supplier manager for each remote team and the incumbent of this role is a member of the central team.

In the first year of the project the central team was staffed by four members of SCR's technical staff (part time) and two masters students (full time) working onsite at SCR. The team consisted of several roles: architect, requirements engineer, project manager, supplier manager, and build meister. The build meister handled the system testing, integration, configuration management, and delivery of artefacts from the teams.

For the second year of the project, the structure of the central team changed. It was staffed

with four interns working full time on the project along with one member of SCR's technical staff working part time in a supervisory capacity. Team roles were similar to the first phase but evolved to include an infrastructure manager, quality assurance manager, process manager, integrator, and wiki administrator. The role of a build meister was abolished as it encapsulated several key responsibilities that could be split and assigned to various central team members. Three graduates from the University of Limerick with an MSc in software engineering were central team members in the second year, one of whom became the central team contact for UL.

Remote development teams were set up at five universities across the globe in the first year. The teams consisted primarily of masters or graduate students in the field of software engineering. UL, Carnegie Mellon University, Technical University of Munich, and the Indian Institute of Information Technology each had one team while Monmouth University had three teams. In the second year of the project, an additional student team was established in PUCRS in Brazil. Each team member committed a specified number of hours each week on the project, taking the academic commitment within their own university into account. Each team had a direct contact on the central team.

Ard na Croise: UL's Involvement in the Global Studio Project

Participation by the MSc in software engineering students at UL in the Global Studio project has been implemented as part of the MSc dissertation process. Students are expected to research a topic relevant to their degree. Students who chose to participate in the GSP project were expected to become reflective practitioners within the project and use this analysis to compare with their chosen aspect of GSD in the "real world." For example, dissertations included a comparison of project management in the local versus the global, and how communication is carried out within global project teams.

In each year, five students volunteered to participate in the SCR GSP project. In the first year, none of the students had previous industrial experience and did not know each other prior to commencing the course a few weeks earlier. Each of the students assumed a role within the team—project leader, software architect, technical support, tester, and quality assurance. These roles were decided on by the team members themselves without any input from the supervisor. Internally, within the university, students were given the software and hardware that they required. This was located in a laboratory shared with students studying this course and other postgraduate courses. The UL team was assigned a supplier manager who was based in SCR. All contact for the first 2 months was via Internet or telephone. The supplier manager visited UL at that stage.

In the first year, as with all participating teams, the UL teams were engaged in the definition phase of the project. Initial tasks were assigned with the goal of bringing the teams up to speed with the building automation domain. Work breakdown was done considering several factors. The schedule, available weekly student hours, and the geographic proximity of the teams to the central team were all considered when defining the original work breakdown. During the first year, it was decided (as part of the research project) not to allow the UL team to communicate with any other team. Their communication was only with the central team. Work packages were defined and distributed to all of the teams. The initial work packages contained: requirements specification, high level architecture, market intent, specifications, high level project plan, and a description of the tasks to be completed by the team for the first engineering release.

In year 2, a new team was formed. One part-time student had significant experience, and he became the project leader of the team. Other roles

assumed by each of the team members were defined by the SCR central team. The UL team was also provided with a space that was private to the project team. The SCR central team contact for UL had been the project leader on the UL team the previous year. Prior to this, no member of the UL project team knew him, and none of them had met him. Furthermore, the restriction on the UL team communicating only with the central team was removed. A wiki, set up by the central team, was continually updated by students and central team members. Otherwise, the project structure was similar to that in year 1.

Ainm an Eolaithe with Ball State University

As with other aspects of GSD, distributed testing is becoming important to the global software development industry. Researchers from UL and Ball State University (BSU) have been involved in studying management and technical issues involved in the distribution of testing across countries within a multinational company who are based in Ireland, USA, and Malaysia (Casey & Richardson, 2005; Zage, Zage, & Wilburn, 2005). As part of this research project, we set up a collaborative teaching project involving MSc students from both UL and BSU. While we were interested in establishing "what works" and "what does not work" when implementing global testing, a further aim of this project was to provide knowledge and education on GSD to individual students in both locations.

The time frame for this joint project began with a week of instruction for the UL students. The projected testing time line of the project was to commence with black-box testing[7] (UL) in weeks 2 through 4. During the black-box testing time frame, it was expected that the white-box testers (BSU) would review the actual code, would review the unit testing environment, and prepare test scripts. At the end of the semester each BSU team was expected to submit the combined defect reports (UL and BSU), the white-box testing scripts, the white-box test cases accompanied by the testing resources, and a summary report. The black-box and white-box test cases were to be recorded in a collaborative testing tool (GATE), which was used by students from both locations. Within GATE, students had access to a wiki where they were free to enter discussions with each other and where they could add information about themselves.

Ainm an Eolaithe: UL's Involvement in the Testing Project

At UL, all full-time students studying software quality engineering (one module) as part of their MSc in software engineering course were randomly selected to participate in testing teams, which comprised two or three students based at UL. Many of these students did not know each other prior to the course, so in some cases they were working with people they had not met before they attended these classes. However, they had been in the MSc class together for 8 weeks before this project commenced. These teams were then paired, again, at random, with student teams from BSU, thus creating virtual teams.

Initially some difficulties arose with the provision of the product to be tested. UL students were given a week-by-week oral update of status and were also given regular updates through the class Web page. This impacted on the start of the initial projected time line for testing outlined perviously. The Irish students testing assignment started 3 weeks later than anticipated, which meant that their training was completed 3 weeks before they had an opportunity to start testing.

RESEARCH METHODOLOGY

Having instigated and managed both of these projects within the MSc class, both of which required extra collaboration and input from the

university lecturers, we were interested to establish the success or otherwise of both of these global projects. We collected and analysed data throughout the lifetimes of each of the projects described previously. The results presented in this chapter are based exclusively on the analysis of data obtained from UL participants.

Each member of the two UL Ard na Croise teams participated in a semistructured interview with one of the authors. These interviews focused on the expectations and experiences of each of the participants. When the interviews were conducted, five students had completed their dissertations, the remaining five had completed 3 months' involvement in the project.

During the Ainm an Eolaithe UL/BSU collaboration, UL students were required to maintain a log book describing their participation. As part of their continuous assessment submission, they were required, in teams of two or three, to write a reflective analysis on the experience of being involved in a GSD testing team.

While the educational objectives (outlined later on) were of interest to the researchers, a semistructured, grounded approach was adopted, so that students were facilitated in identifying learning-related issues in their own words and on their own terms. However, students were guided in the provision of responses and insights by a variety of prompts, which, for example, in the interviews that were conducted included questions like: Why did you participate in the project? What were your expectations? Were your expectations fulfilled? What did you learn? What was good? What was bad? What was done well? What was done badly? What should be changed? What should I keep? Would you participate again? These prompts had two functions. Firstly, they were designed to help students to make more sense of their experiences and to integrate those insights into their learning. Secondly, they allow us to capture and disseminate (via this commentary) issues that may be of use to teachers and academics planning similar interventions within their own contexts.

The interview notes, log books, and the reflective analysis was analysed by the researchers. A qualitative content analysis of these sources of data took place in order to identify overarching themes associated with the students' experiences of this learning intervention. A broad analysis of the frequency of different concepts was conducted based on an interrogation of the available data.

STUDENT INSIGHTS

The data gathered from students about their insights and experiences were rich and diverse, expressed mainly in their own words in response to various prompts and delivered as part of the reflective requirements of the programme of study on which they were enrolled. We categorised the feedback, reflections, and ideas that they provided, while also attempting to stay true to the student voices and the priorities that they reported. Throughout this process we maintained awareness of the emphases that we had established as educators. We observed certain insights, focusing on either pedagogical process or GSD requirements. Oftentimes, these observations were so intertwined, that they were essentially inseparable in the eyes of the students themselves. Therefore, this section organises these student insights into categories sequenced in order of the frequency and emphasis with which certain response categories appeared in the data. There is no explicit division between pedagogic and GSD issues.

Our analysis of the reflective, qualitative data reveals seven overarching categories of student insight into the learning experiences associated with the GSD project process. These categories can broadly be summarised under the following headings:

1. The importance and subtleties of communication and timing of contact.
2. Interpersonal awareness and team dynamics.

3. The movement from incompetence and uncertainty to confidence and command over ambiguous dynamics.
4. Issues associated with energy and emotion: stress, isolation, commitment, morale, satisfaction, and motivation.
5. Awareness of and concern with how students present themselves as individuals and teams, and how they represent their institution
6. Career development advantages.
7. Empowerment, responsibility, and decision making.

Each category is discussed in further detail. This discussion is supported by respondent statements (presented in italics).

Importance and Subtleties of Communication and Timing of Contact

A content analysis of student insights shows that the issue of communication was the most frequently invoked overarching "insight category." A variety of subcategories also emerged here which mainly focused on the inherent value, the subtleties, and the timing of communication. Participants frequently referred to the overall importance of having virtual team communication that was regular and positive, while also revealing perceptions that this was difficult to achieve. General comments about communication often revealed how important the participants considered (or came to consider) it to be:

Communication would be the big one.

Communication can be helped with emails. Phone calls, Wiki's, instant messenger etc but require effort and participation from distributed teams.

The pragmatic issue of timing of communication was seen generally as a problematic aspect of the experience, but also often recognised both as a realistic and inevitable challenge that needed to be managed. In addition, many of the insights that students provided suggested that "waiting" for responses from the team members in the remote location may have given rise to frustrations and anxieties that affected the group and its work.

We were told by SCR to sit tight and wait.

Timing can be a problem. If the [local] team finds an error at 8am then it won't be until the next day that team B will tackle the error, leading to delays.

One student made a link between early communication problems and a feeling of ambiguity and noted that addressing this would lend clarity and possibly more momentum to the work that they had been assigned:

There should be more phone conferences early in the req phase. This probably would help clear up ambiguity.

Another respondent recognised the importance of the proximity of certain types of expertise by saying:

The remotely located team may have different knowledge and experience than others and therefore communication is a key player to resolve this issue.

Subtleties of communication and the nuances, conflicts, and misinterpretations to which it can often be vulnerable, were also regularly invoked in this response category. Several respondents talked about the importance of face-to-face contact and seemed to suggest that tone and patience are more important when communicating with remote team members.

Learning point: Know what to say and what not to say.

Team in US was not as helpful as they could have been. They made us feel bad about it "Where is it?" I like meeting and talking to people.

With the teams so dispersed there was no face to face warming of the teams. This led to a slow start in stimulation of the participants.

The sense of interdependence between the locations was also invoked:

Coordination of effort also came into play after the black-box testing phase as the American teams used our completed tests to form a basis for white box testing. Our performance would obviously have an impact on that.

These comments echo with communication competencies that we identified in *Educating the Graduate* section as essential to facilitate the software developer and manager functioning effectively in the GSD environment. The student teams articulated insights about communication in a global classroom setting, a fact that should equip them to operate more effectively in subsequent GSD work environments.

Interpersonal Awareness and Team Dynamics

Commonly, students' reflective statements about their experiences as a part of the project, invoked the issue of interpersonal awareness and team dynamics. These statements were coded under this category when they specifically referred to issues about "getting to know" the remote team members or themselves more accurately. They were linked to issues of communication too, but the following statement types were deemed to be sufficiently different and more focused on team dynamics and interpersonal awareness issues to merit their own category:

We got on very well as a team. We did have problems. Some worked better with each other. Some were really pessimistic—could only see the downside of things. I tried as a team leader to see the upside—sometimes seeing the downside can stop a project too—it does stop the whole thing from happening, or stop the momentum but it gives a reality check.

I started to understand the importance of group work. I have more of an understanding of different personalities—what we had to deal with—laid back or not.

In particular, one of the respondents indicated a need to "get the measure" of people in the remote locations and refers directly to "making up" personalities. *[With this project] you never get a sense of what the other guys are like. We have made up a personality for [him] (S8).* This suggests the conscious "construction"of individual images along with their own mistrust in their theories about other people.

There also seem to emerge "theories" about team functioning. Several comments from respondents suggested that they had generated overall hypotheses about how teams should work, based directly on the experiences that they had with this task:

Teams should be more inquisitive.

When there are 5 people there are always problems of someone not there.

The development of a sense of "teamness" among all members of the virtual team was also highlighted more particularly as problematic through the following insights:

At no stage was a sense of teamness developed. In fact at times it felt very much a case of running test cases and hearing nothing back.

The aforementioned statements reflect a rich engagement with the development of insights about team characteristics, and an emerging sense of understanding gained about the strengths of different members of the team. Some of the statements provide evidence of a fine-grained development of knowledge about which team member's strengths relate to which aspect of the task. In addition, respondents demonsrated the feeling of getting to know different emotional orientations towards the processes in which they were immersed (for example, *some people could only see the downside of things*). It is difficult to see how such fine-grained knowledge of team dynamics could be achieved without the direct participation that they had experienced, and it is unlikely that the curriculum relating to their programme of study specifies this level of understanding and knowledge of the nuances and complexities of team functioning. The statement provided by respondent S8, reveals something even more complex—that is, a sense of caution and awareness about the assumptions made relating to personalities of people that had not yet been met in person. *We have made up a personality for [him]* is a statement that may reveal the need for team members to apply characteristics to people they have not yet met, accompanied by a recognition that the assumptions are only provisional and cannot be confirmed from a distance. In terms of interpersonal awareness and insight, such statements demonstrate strong and complex learning of some of the intangible aspects of GSD as a process and a task.

The Movement from Incompetence and Uncertainty to Confidence and Command Over Ambiguous Dynamics

There is plenty of evidence among the reflective statements provided in this study that a developing sense of competence and confidence started to emerge as part of the GSD experience. Statements that signalled a gradual unfolding of knowledge and engagement with the task included the following:

We did not know what they meant. As we went through the course more we got to know what they meant.

We learned from our mistakes and adapted.

It is often said that we learn more from our mistakes than our successes and in that case it has been fair to say that it has been a worthwhile experience.

It also seemed clear that part of the journey to competence included encountering an experience of ambiguity and confusion, as revealed by the following kinds of statement:

Hard to pin down how code was going to work in a new system.

A component was changed so all the work we did before Christmas was wasted.

Requirements were not good. We constantly had to read between the lines.

Insights about emerging confidence also appeared frequently in students' responses and reports, particularly in response to prompts that encouraged them to think specifically about what they had learned from the process. Familiarity, comfort, and confidence seem to have replaced the sense of the task having been "daunting" and of feeling out of their depth. The recognition of the importance of experience and the role that it has played in reinforcing and developing their learning seems to be clear:

Well now I've seen that (req doc) before so it won't be as daunting .

It all comes down to experience. If we could go back and do it again, we'd do it differently.

The fundamental purpose of education is to facilitate the journey towards competence. It is clear from the previous statements that students appeared to navigate a kind of pedagogical "journey" to competence. When looking back on this experience, their insights suggest that the negotiation of confusion and ambiguity was central, if not essential, to the learning processes in which they were involved. Again this suggests that there was something very crucial about the simulated GSD experience, and that in conventional learning environments the movement from ambiguity to clarity might not be as significant or as realistic as those experienced by this group of students. A link between these insights and the developing sense of "teamness" reflected in other comments suggests that the developing sense of competence might also occur in tandem with a developing sense of teamness. This may be evidence of the students' enhanced understanding of the sophisticated and complex dynamics of GSD.

Issues Associated with Energy and Emotion: Stress, Isolation, Commitment, Morale, Satisfaction, and Motivation

Another common statement category among students suggested that they were readily able to identify "emotionally relevant" aspects of the experience of working in a GSD team. The ability to signal the relevance of difficult feelings associated with stress and isolation as well as the more positive emotions associated with commitment, morale, and motivation demonstrate another dimension of self-awareness that may have developed and become enhanced over the course of the experience.

Student references to experienced stress and isolation included the following types of statement:

We felt isolated. We felt a bit alone as a small development team.

It was time consuming and stressful but at the same time I enjoyed it.

Some of the stress was linked to the time deadlines associated with the learning experience, while one respondent also highlighted that the emotional engagement with the task could be experienced as simultaneously positive and negative.

In addition, responding students also seemed to demonstrate an awareness of the commitment required in order to complete the task successfully and on time. Some of them demonstrated this awareness by highlighting that among their team members that commitment was not always present, while others took commitment as one of the learning outcomes of the process:

Make sure they understand the commitment they need to give to the project. People not coming in [on] time affects morale.

Are you going to be in at 9 a.m. during the 3 week break. It is worth it in the end if you can do it.

Others noted how the experience induced commitment by creating inherent motivators into their lives as students. Statements demonstrating this insight included the following:

This gave me something to get up and come into college for [even when my lecture schedule did not require it].

I have something to push me along.

The realisation of the effective dimension of working seems to have had an effect on students' recognition of how feelings can impact on projects. An explicit recognition of the added value of intangibles like commitment shows that students

are developing a more subtle comprehension of how team dynamics work and how emotions impact on those dynamics in sometimes fundamental ways.

Awareness of and Concern with How Students Present Themselves as Individuals and Teams, and How They Represent Their Institution

A very clear category of statements emerged from respondents that related specifically to the ways in which the local student team presented itself and appeared to the remote members of the GSD teams. A concern for how they came across to their corresponding team members in the other locations emerged as something that was important to students:

We didn't want to look like a bunch of muppets and you were our supervisor.

You don't want to be seen to be proven wrong. Very pleased when we were able to say the other team was wrong.

We didn't want to be seen as a failure.

These statements express that students were prevented from availing themselves of useful learning moments during the process. They were reluctant to clarify, question, and learn. This was not because of pressure from lecturers, project supervisors, or central team members but because of an internal pressure that they put on themselves and a concern regarding how they might be viewed by other stakeholders in the process.

It also seems clear that once they had met face to face with a representative of the remote team members, this reluctance and concern diminished significantly:

Once they met the guy from Siemens they were much happier.

This year's team haven't got that problem at all because the links are stronger.

Furthermore, the implementation of the wiki, which is used as a remote communication tool within the Ard na Croise project during year 2 has helped communication—*Wiki may be helping too—to break taboo about asking "stupid" questions.* The wiki contains questions from all students internationally—so UL students have visibility into queries from other international student teams. This was not the case in year 1 of the project, when UL students were isolated from communicating with other teams. This was a decision of the research project team.

The most striking aspect of this category of statements lies in its identification of a learning paradox within the student teams. On one hand, the concern for projecting themselves in a positive light led to them wanting to work hard and appear to be performing at high levels. This may have boosted their motivation and engagement with the task. On the other hand, the need to be seen in a positive light seemed, at least occasionally, to block good learning, as the students were reluctant to ask important questions at key stages in the process.

Career Development Advantages and Practical "Real-Life" Experience

Students tended also to invoke the pragmatic and instrumental benefits associated with having participated in the process. Many of them mentioned specifically how being able to say they had participated in the project would be helpful for their career opportunities, indicating particularly that it would give them "the edge" in an interview setting.

In addition the benefits of having been involved in an activity that reflected the real life pressures and structures of GSD was something that they invoked explicitly as being positive and beneficial:

While we did not hypothesise that this experience would be seen in such pragmatic ways by the participants, our research shows that the pragmatic benefits of participating in this kind of educational experience must not be underestimated. We as educators tend to look at the "whole curriculim" as contributing to the overall employability of our students. However, it seems in this instance that the specific GSD experience reported here was heavily weighted by students as having particularly useful career development effects. It is worth noting that out of five participants in one of the GSD projects, three were subsequently employed in GSD environments, a series of choices and opportunities that may not have arisen without their participation while studying.

Empowerment, Locus of Control, and Decision Making

Students also highlighted the struggles and issues that they encountered in the development of a more "internal locus of control" for the work that they were doing. They highlighted key questions and problems that seem to indicate that a gradual emerging of empowerment occurred over the course of the project. The worries associated with who was responsible for what were indicated in the following types of statements:

I'd feel a good bit responsible.

I didn't think we should go to Ita.

One student articulated an emerging conviction that the decision making relating to the task ultimately lay within their own team:

We had to learn to figure things out for ourselves.

Several noted the issue of responsibility and locus of control as something that needed to be defined and clarified:

Something like a contract would be good at the start.

It's important to assign responsibilities. If we don't, no-one will take it on for themselves.

Another intangible but important aspect of learning and work performance is relevant to this issue of locus of control. Understanding levels of responsibility and empowerment influences individual and group decision making in very significant ways. In grappling with the issue of locus of control and decision making in GSD contexts, students also seem to have gained a possibly deeper level of insight about their engagement with the tasks and processes.

The pragmatic effects of student participation can be seen explicitly when we analyse their career development advantages. However, there were other pragmatic effects observed in the study, for example, following their participation in the global projects, they recognise the importance of teamwork (*I learned a lot of things that are valuable like working with the group*), how they need to change their work practices (*It involved changing my mindset from haphazard to process*), and how to deal with responsibility (*Need to set ground rules and clarify responsibility*).

As illustrated in Table 1, graduates are expected to demonstrate competencies for use in the GSD environment. Through involvement in the projects discussed, students had exposure to all of the competencies required by the software developer. Through the discussion on interpersonal awareness and team dynamics, students developed communication and language and worked with other cultures. They also had exposure to communication tools and how these are used differently by different cultures (for example, *Differences in the Irish and American teams—Irish used email more, but US used discussion boards*). The reality of temporal issues became clear (*Communication may have been delayed as a result of the time difference*). The students were

also exposed to cooperation issues. Not only did they have to work together within local teams, they also had to cooperate with internationally dispersed teams. Furthermore, in the Global Studio Project, in particular, they were required to report to a "management" team in Siemens Corporate Research. As discussed earlier, this was not always an easy task and participation ensured that the students learned how to cooperate. In our interviews, statements from the students such as—*One would make breakthrough, others would put shoulders to the wheel*—illustrate this. In the global environment, as in other development environments, software developers should also be aware of processes and process tools. Participants in these projects were exposed to processes such as requirements and configuration management and were expected to continue development even with changing requirements. Global tools were implemented and used by the students. Students were expected to make regular reports to both their internal university supervisor and to the teams with whom they were working.

In addition to competencies required by the software developer, students were also exposed to some of those required by the software manager. In particular, they experienced roles, responsibilities, and competencies management. In one case, when the students stated: *We didn't want to look really bad. We all thought that if we kept ringing with questions, we did not want to seem incompetent;* they learned that to survive in the global environment, then they needed to do exactly the opposite and have their questions answered.

CONCLUSION AND OBSERVATIONS

Through the qualitative analysis of key reflective statements conducted for this chapter, it has been possible to extract several overarching themes that demonstrate participants' own perceptions of the learning concerns, experiences, benefits, and outcomes associated with being part of a virtual GSD team. We now bring these themes together in an effort to map the student experiences with pedagogical interventions and aims associated with this project.

Moore and Ryan (2006) remind educators that "*Important truths that seem obvious to experienced practitioners can often only be learned by students if they have been encountered in a 'deep' as opposed to a 'shallow' sense*." In the complex experience of joining an international, virtual GSD team, students gained insights, learned lessons, and acquired understanding that would otherwise have been very difficult, if not impossible to achieve. In an analysis of students' own words we have seen evidence of both cognitive and affective breakthroughs achieved directly as a result of their experience with a GSD environment and tasks. While students could be given vicarious insights (through lectures, tutorial sessions, and discussion forums) about the importance of knowing the subtleties of communication, team dynamics, and perceptions, we find it difficult to imagine how the range of work-based comprehension that they have reported could possibly be achieved outside of the experiential frame provided by this learning experience. In sketching out the framework of competencies that the literature highlights as central to the needs of GSD developers and managers, we are struck by the parallels found between this competence set and those identified by the students when discussing and explaining their learning outcomes.

For several decades now, there have been significant criticisms of conventional higher education environments. Biglow (1983) highlighted a direct link between university classroom norms and the problems faced by graduates on entering the workplace. Passivity, dysfunctional competition, and lack of a sense of responsibility are, he argues, all facilitated by inactive, disengaged educational environments. While conventional educational norms can be active, engaged, and

excellent, it is arguably much easier to create these conditions in more experiential settings like the ones described and evaluated in this chapter.

By observing the seven outcomes identified previously, it is possible to argue that there are three forms of learning impact experienced by students who participated in this project: (1) pedagogical, (2) pragmatic, and (2) achievement of specific GSD competencies. The pedagogical effects can be tracked most strongly through observed references to the journey from incompetence to competence. The pragmatic effects, while they may derive from this journey, are more closely evidenced by participants' own insights about the practical and optical value of participation. The specific GSD competency development is demonstrated through the students' own reported sense of achievement, but still requires confirmation through the normal curriculum assessment process. We do, however, argue that the sense of competence that the students report sets the scene for more positive and engaged learning, even in didactic contexts.

In summary, we champion the application of the learning experiences we have described in this chapter, while recognising that the implications of this for GSD learning are that educators need to work interinstitutionally and in tandem with industry where relevant. Mimicking real work settings creates the possibility of giving rise to the range of learning benefits that are associated with truly problem-based learning environments. Adding pedagogical support, opportunities for reflection, and the capacity to revise and develop ideas using experienced teachers as a sounding board creates an added-value environment that we believe should become a standard part of software engineering education curricula.

ACKNOWLEDGMENT

The research in this chapter has been funded by Science Foundation Ireland through the Principal Investigator, B4Step project (Grant no. SFI 02/IN.I/108) and Cluster project GSD for SMEs (Grant no 03/IN3/1408C) within the University of Limerick, Ireland; by the National Science Foundation (U.S.) within the Software Engineering Research Center, Ball State University, Muncie, Indiana, U.S. (Grant no: EEC-0423930); and by Siemens Corporate Research, Princeton, New Jersey, U.S.

REFERENCES

Augar, N., Raitman, R., & Zhou, W. (2004, December 5-8). Teaching and learning online with Wikis. In R. Atkinson, C. McBeath, D. Jonas-Dwyer, & R. Phillips (Eds.), *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference* (pp. 95-104).

Barrett, T. (2005). Understanding problem based learning. In Barrett, MacLabhrainn, & Fallon (Eds.), *Handbook of enquiry and problem based learning: Irish case studies and international perspectives.* AISHE: Galway.

Bigelow, R. (1983, May). *The importance of engagement and responsibility in higher education.* Paper presented at the University of Iowa Educational Development Conference, Iowa City.

Boland, D., & Fitzgerald, B. (2004). Transitioning from a co-located to a globally-distributed software development team: A case study and Analog Devices Inc. *ICSE International Workshop on Global Software Development,* Edinburgh, Scotland.

Carmel, E. (1999). *Global software teams: Collaboration across borders and time zones.* Upper Saddle River, NJ: Prentice Hall.

Casey, V. (2007). PhD to be published, University of Limerick, Ireland.

Casey, V., & Richardson, I. (2004). Practical experience of virtual team software develop-

ment. *European Software Process Improvement (EuroSPI) 2004,* Trondheim, Norway.

Casey, V., & Richardson, I. (2005, September 26-30). Virtual software teams: Overcoming the obstacles. *Third World Congress on Software Quality,* Munich, Germany, 1-63 to 1-70, Volume 1.

DeSanctis, G., Staudenmayer, N., & Wong, S. S. (1999). *Interdependence in virtual organisations, The virtual organization (Trends in organizational behavior).* Chichester, England, UK: John Wiley & Sons.

Ebert, C., & De Neve, P. (2001). Surviving global software development. *IEEE Software, 18*(2), 62-69.

Grinter, R. E., Herbsleb, J. D., & Perry, D. E. (1999). The geography of coordination: Dealing with distance in R&D work. In *Proceedings of ACM SIGGROUP Conference on Supporting Group Work* (pp. 306-315). Phoenix, AZ.

Hayes, I. S. (2002). Ready or not: Global sourcing is in your IT future. *Cutter IT Journal, 15*(11), 5-11.

Herbsleb, J. D., & Grinter, R. E. (1999). Splitting the organisation and integrating the code: Conway's law revisited. *Twenty-first International Conference on Software Engineering,* Los Angeles: IEEE Computer Press.

Herbsleb, J. D., & Mockus, A., Finholt, T. A., & Ginter, R. E. (2000). Distance, dependencies and delay in a global collaboration. *ACM conference on Computer Supported Cooperative Work,* Philadelphia: ACM Press.

Herbsleb, J. D., Paulish, D. J., & Bass, M. (2005, May 15-21). Global software development at Siemens: Experience from nine projects. In *Proceedings of the 27th International Conference on Software Engineering, ICSE '05* (pp. 524-533). New York: ACM Press.

Hwarng, H. B. (2001). A modern simulation course for business students. *Interfaces, 31*(3), 5-11.

Jarvenpaa, S. L., & Ives, B. (1994). The global network organization of the future: Information management opportunities and challenges. *Journal of Management Science and Information Systems, 10,* 25-57.

Joy, M. (2005). Group projects and the computer science curriculum. *Innovations in Education and Teaching International, 42*(1), 15-25.

Kiel, L. (2003). Experiences in distributed development: A case study. *ICSE International Workshop on Global Software Development,* Portland, OR.

Levitt, B. B., & March, J. G. (1988). Organisational learning. *Annual review of Sociology, 14,* 319-340.

Linnane, S., & Richardson, I. (in press). Distributed software development—Difficulties for the SME. In Perspectives in *Software Quality, Proceedings of Software Quality Management Conference, SQM2006,* (pp. 113-128). Southampton, UK.

Lipnack, J., & Stamps, J. (1997). *Virtual teams: Reaching across space, time and organisations with technology.* New York: John Wiley & Sons.

Luo, Y., & Peng, M. W. (1999). Learning to compete in a transition economy: Experience, environment and performance. *Journal of International Business Studies, 34,* 52-68.

McDonough, E. F., III, Kahn, K. B., & Barczak, G. (2001). An investigation of the use of global, virtual, and collocated new product development teams. *Journal of Product Innovation Management, 18,* 110-120.

Moore, S., & Ryan, A. (2006). Learning to play the drum: An experiential exercise for management

students. *Innovations in Teaching and Learning International,* 435-444.

Nidiffer, K. E., & Dolan, D. (2005). Evolving distributed project management *IEEE Software, 22,* 63-72.

O'Brien, J. A. (2002). *Management information systems managing information technology in the business enterprise* (6th ed.). McGraw-Hill Irwin.

Organisation for Economic Co-operation and Development (OECD). (2004). *International investment perspectives.* France: Author.

Powell, A., Piccoli, G., & Ives, B. (2004). Virtual teams: A review of current literature and direction for future research. *The DATA BASE for Advances in Information Systems, 35,* 6-36.

Pyysiainen, J. (2003). Building trust in global interorganizational software development project: Problems and practices. *ICSE Workshop on Global Software Development.*

Reynolds, M. (1994). *Groupwork in education and training.* London: Kogan Page.

Riley, M. (2005, September 13). The proof is out there. *Software Development.* Retrieved February 2006, from http://www.sdmagazine.com/documents/s=9889/sdm0510b/0510b.html

Robey, D., Khoo, H. M., & Powers, C. (2000). Situated learning in cross-functional virtual teams. *IEEE Transactions on Professional Communications, 43*(1), 51-66.

Savin-Baden, M. (2004). Understanding the impact of assessment on students in problem based learning. *Innovations in Education and Teaching International, 41*(22), 221-233.

Sommerville, I. (2001). *Software engineering* (6th ed.). Harlow, UK: Addison-Wesley.

Technology Foresight Ireland. (2001). *Information and communications technologies.* Report from the Information and Communications Technologies Panel. Dublin, Ireland: Forfás.

Tjosvold, D., Yu, Z., & Hui, C. (2004). Team learning from mistakes; The contribution of cooperative goals and problem solving. *Journal of Management Studies, 41*(7), 1223-1245.

Toaff, S. S. (2002). Don't play with "mouths of fire" and other lessons of global software development. *Cutter IT Journal, 15*(11), 23-28.

Zage, D., Zage, W., & Wilburn, C. (2005, April). *Test management and process support for virtual teams* (Tech. Rep. No. SERC-TR-271). Muncie, IN: Ball State University.

ENDNOTES

1. A virtual team may be formally defined as *"A team whose members use the Internet, intranets, extranets and other networks to communicate, coordinate and collaborate with each other on tasks and projects even though they may work in different geographical locations and for different organisations*" (O'Brien, 2002).
2. [sm] CMMI is a service mark of Carnegie Mellon University.
3. A wiki (from the Hawaiian word meaning quick) is a user-editable Web site. Users can visit, read, reorganise and update the structure and content of a wiki as they see fit. An Internet connection and Web browser are all that are required to read and edit a wiki, (Augar, Raitman, & Zhou, 2004).
4. [sm] CMMI is a service mark of Carnegie Mellon University
5. Ard na Croise (height of the cross) is a hydro-electric power station built by Siemens, Germany and is located about 8km from the

University of Limerick. Its 75th anniversary was celebrated in 2005.

[6] Anim an Eolaithe (name of the scientist): In recognition of the scientific achievements in Ireland and the U.S., each of the teams was named after two celebrated scientists—one Irish and one American.

[7] Black-box or functional testing is an approach to software testing where tests are derived from the program or from its specification. Success or failure of the test can only be determined from studying its inputs and related outputs (Summerville, 2001).

This work was previously published in Information Systems and Technology Education: From the University to the Workplace, edited by G. Lowry and R. L. Turner, pp. 82-104, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 5.19
A Requirement Elicitation Methodology for Global Software Development Teams

Gabriela N. Aranda
Universidad Nacional del Comahue, Argentina

Aurora Vizcaíno
Universidad de Castilla-La Mancha, Spain

Alejandra Cechich
Universidad Nacional del Comahue, Argentina

Mario Piattini
Universidad de Castilla-La Mancha, Spain

INTRODUCTION

Failures during the elicitation process have been usually attributed to the difficulty of the development team in working on a cooperative basis (Togneri, Falbo, & de Menezes, 2002), but today there are other points that have to be considered. In order to save costs, modern software organizations tend to have their software development team geographically distributed, so distance between members becomes one of the most important issues added to the traditional problems of the requirement elicitation process (Brooks, 1987; Loucopoulos & Karakostas, 1995).

So far, literature has widely analysed real life Global Software Development (GSD) projects and pointed out the main problems that affect such environments, especially related to communication. As a complementary view, we have focused our research on analysing how cognitive characteristics can affect people interaction in GSD projects, especially during the requirement elicitation process, where communication becomes crucial.

In this article, we present the main characteristics of requirements elicitation in GSD projects and introduce a cognitive-based requirement elicitation methodology for such environments.

BACKGROUND

Advantages and challenges of GSD have been widely analyzed in literature. As part of the advantages, the most cited are:

- Taking advantage of time difference to extend productive hours (Herbsleb & Moitra, 2001);
- Minimizing development costs (Lloyd, Rosson, & Arthur, 2002);
- Locating developers closer to the customers (Damian & Moitra, 2006); and
- Taking advantage of diversity of stakeholders' knowledge and experiences (Ebert & De Neve, 2001).

On the other hand, the challenges that GSD must face are (Damian & Zowghi, 2002):

- the loss of communicative richness, affected by the lack of face-to-face interaction;
- the time difference between sites, that introduce delays in the project;
- cultural diversity, as a source of misunderstandings; and
- knowledge management, because of the need of maintaining information from many distributed sources.

Looking for solutions to improve communication in GSD, concepts from CSCW (Computer-Supported Cooperative Work) become important because this research area concerns the development of software for enabling communication between cooperating people (*groupware*), that can be simple systems (like e-mail or plain-text chat), more complex ones (like videoconferencing), or the combination of more than one of them. To be more specific, when talking about groupware we follow a convention: We refer to every simple communication technology (e-mail, chat, videoconference) as groupware tools, and to the systems that combine them as groupware packages (Gralla, 1996). Doing so, the most common groupware tools used during multisite developments are e-mails, newsgroups, mailing lists, forums, electronic notice boards, shared whiteboards, document sharing, chat, instant messaging, and videoconferencing (Damian & Zowghi, 2002; Gralla, 1996).

Another research area related to the distributed requirements elicitation process is Cognitive Informatics (CI), a transdisciplinary research area that encompasses informatics, computer science, software engineering, mathematics, cognition science, neurobiology, psychology and philosophy, and knowledge engineering (Chiew & Wang, 2003). In CI, there is a bidirectional relationship between cognitive sciences and informatics (Wang, 2002):

1) using computing techniques to investigate cognitive science problems like memory, learning, and thinking; and
2) using cognitive theories to investigate informatics, computing, and software engineering problems.

In our research, we have followed the second point of view, using concepts from cognitive psychology to improve the requirement elicitation process. Doing so, our research focused on learning styles models (LSMs), a cognitive psychology theory based on Jung's theory of psychological types published in 1921 (Miller & Yin, 2004), that classify people according to the ways they perceive and process information. These models have been discussed in the context of analyzing relationships between instructors and students, but we propose applying them to a virtual team that deals with a distributed requirement elicita-

tion process, considering an analogy between stakeholders and roles in LSM, because during the elicitation process everybody "learns" from others (Martin, Martinez, Martinez, Aranda, & Cechich, 2003), and stakeholders play the role of students or instructors alternatively, depending on the moment or the task they are carrying out.

After analyzing five LSM in Martin et al. (2003), we found out that every item in the other models was included in the model proposed by Felder-Silverman (Felder & Silverman, 1988), so that we may build a complete reference framework choosing this as a foundation. The Felder and Silverman (F-S) model classifies people into four categories, each of them further decomposed into two subcategories (*Sensing – Intuitive*; *Visual – Verbal; Active – Reflective; Sequential – Global*). To know their cognitive profile, people must fill in a multiple-choice test (available at http://www.engr.ncsu.edu/learningstyles/ ilsweb.html), that returns a rank for each category. Depending on the circumstances people may fit into one category or the other, being, for instance, "sometimes" active and "sometimes" reflective, so preference for each category is measured as *strong*, *moderate*, or *mild*.

Most of related works use learning and psychological style models with educational purposes, while few works use them to solve problems in software engineering. One work which uses cognitive styles as a mechanism for software inspection team construction is described in Miller and Yin (2004). They use the MBTI method, an instrument similar to the F-S model. Their intent is different from ours because they use the cognitive styles to set which people seem to be more suitable to work together, while we try to give the best solution (concerning technology) for an already chosen group of people.

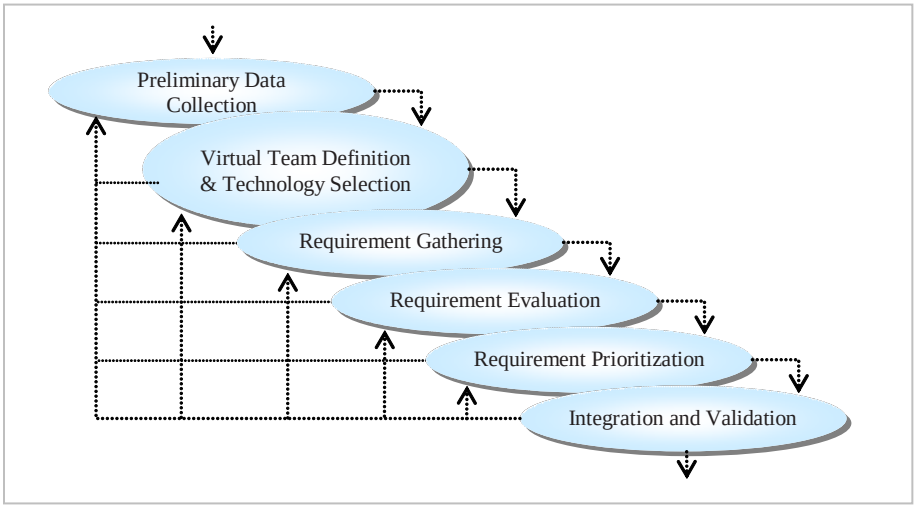
OUR METHODOLOGY: RE-GSD

In order to define the basis for a methodology for requirement elicitation in GSD projects, we have analyzed methodologies used in colocated development and proposed extending them from a cognitive point of view, using the F-S model as a basis for defining a model for technology selection.

We have called our methodology RE-GSD (Requirement Elicitation for Global Software Development projects) and, as a starting point for it, we have selected the models proposed by Christel and Kang (1992) and Hickey and Davis (2003). This selection is due to the fact that both models share a generic view of the selection of requirement elicitation techniques, which fits our intention of defining what to use according to stakeholders' personalities. Both models have been extended and adapted to a distributed environment, so that our methodology can be expressed as follows (see Figure 1):

1. Preliminary data collection: Further decomposed into two categories: (1) about the stakeholders, and (2) about the system and the domain.
2. Virtual team definition & Technology selection: Before starting the requirement gathering, it is important to determine who is going to participate in that stage, because not all the stakeholders in the project are required to participate in every iteration of the elicitation process. Then, the selection of appropriate technology should be carried out, that means, choosing the most appropriate set of requirement elicitation techniques and groupware tools for a given group of people, by taking into account their personal characteristics.

Figure 1. RE-GSD methodology



3. Requirement gathering: Once technology has been defined, it is time to apply the requirement elicitation techniques (which are combined with appropriate groupware tools) to obtain a new list of requirements, trying to answer "*what*" is to be built (Christel & Kang, 1992).
4. Requirement evaluation: In this stage, requirements lists must be analyzed in order to determine consistency between different statements.
5. Requirement prioritization: Once requirements are defined, it is important to give them an order of relative importance so as to know when they should be addressed in relation to other requirements (Christel & Kang, 1992). Specially designed tools for distributed requirement inspection (Lanubile, Mallardo, & Calefato, 2003), which allow synchronous and asynchronous discussion, voting, and so forth, can be used to address both this step and the previous one.
6. Requirement integration and validation: In this step, the new requirement list must be integrated to the requirements collected in the previous iterations, looking for inconsistencies also with the system's goals and organizational factors initially defined.

The following sections present the first two phases of our model that are related to our proposal of technology selection according to the stakeholders' cognitive styles.

PHASE 1: PRELIMINARY DATA COLLECTION

a) About the stakeholders
   1) Identify people whose participation is important for the requirement elicitation process, including people from different levels of the organization.
   2) Get personal information about stakeholders, using the form shown in Figure 2. Some important points for distributed environments are, for instance, distinguishing which is the given name and the family name, because different cultures use different order (for

Figure 2. Stakeholder's personal information form

Stakeholder's Personal Information Form						
Complete Name (as written in the ID card)						
Given Name		Family Name				
Nickname (optional)						
Birthday						
Gender						
Mother Language		Country of Origin				
		Country of Residence				
Academic degree		University / College		Years of study		
For each foreign language (mark with an X your level of knowledge)	<Language>	low	low-interm	interm	high- interm	high
	Writing					
	Reading					
	Speaking					
Felder and Silverman preferences		Active Reflexive	Sensitive Intuitive	Visual Verbal	Sequential Global	

instance, in China the family name goes first, while in most of occidental countries, the family name is the last one). Also, recording information about the country of origin and the country of residence is important to identify their mother language and possible differences in cultural background, as well as the foreign languages stakeholders know to choose a second language.

3) Get information about stakeholders' job, roles, responsibilities and schedules. And because they are distributed, obtaining information about each team member's location (time difference with other sites, work hours, lunch time, etc.) is relevant for other members to know how to contact each other. The form is shown in Figure 3.

b) Get information about the structure, culture, and internal politics of the organization (SWEBOK, 2004), to answer the following questions:

4) *About the groupware tools*: Which groupware tools are commonly used in the organization? Have stakeholders received training in the use of groupware tools? Which ones do they know better? Which ones have they not used before? Is there any policy that limits the use of groupware tools? Are stakeholders willing to learn how to use other groupware?

5) *About the requirements elicitation techniques*: Which requirement elicitation techniques are commonly used in the organization? Have stakeholders received training in the use of requirement elicitation techniques? Which ones do they know better? Which ones have they not used before? Are stakeholders willing to learn new techniques?

6) *About the organizational culture*: Do organizational policies allow stakeholders to communicate with others in the virtual team freely, or is there a person that must act as a mediator?

Figure 3. Stakeholder labour information form

Stakeholder's Labour Information Form						
Role during RE						
Job description	Position			Time in such a position: ……..... years ……... months		
	Place			Time difference (GM)		
Daily timetable	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Arrival time	◷	◷	◷	◷	◷	◷
Dismissal time	◷	◷	◷	◷	◷	◷
Coffee-break(s)	◷	◷	◷	◷	◷	◷
Lunch break	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to
Time I prefer to be called	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to	◷ from ◷ to
Contact information (write the number or user login name)	❍ **Telephone** (number) Country code …….. City code …… Numbers (1) ……. (2) ……. (3) …….			❍ **Fax** (number) Country code …….. City code …… Numbers (1) ……. (2) ……. (3) …….		
	❍ **E-mail** (user name) (1) ……. (2) ……. (3) …….			❍ **Instant messaging** (user name) MSN: ……..……..…. Yahoo messenger: ……..…… Skype: …...……..…… Other: ……...….		
I have also the possibility to use (check)	❍ videoconference ❍ audio conference ❍ others: ……					
If I can choose, I prefer using (put a value of preference between 1 and 10)	…….. e-mail …….. telephone …….. instant messaging …….. discussion forums			…….. shared whiteboards …….. audio conference …….. videoconference		

c) About the system and the domain
   7) Get information about the domain and the system in construction.
   8) Determine the system goals.
   9) Identify similar systems

PHASE 2: VIRTUAL TEAM DEFINITION AND TECHNOLOGY SELECTION

As we have mentioned before, we aim at defining strategies that analyze the personal characteristics of stakeholders with the objective of selecting the best groupware tools and requirement elicitation techniques for them. For that reason, selecting appropriate groupware tools is related to our goal of providing the stakeholders with the possibility of communicating with others in a manner closer to the way in which they perceive and process information. That means giving them the chance to feel comfortable with the way in which they interact (synchronously or asynchronously) and the kind of information they interchange (based on words, based on diagrams, etc.).

Even when the technology selection is done for a given virtual team before the elicitation process, obtaining the preference rules is a previous step.

Preference rules are obtained from a generic set of people that works on GSD projects, and then the resulting sets can be used in many different GSD projects as well as different virtual teams, at the same time that new information can be added to improve the source of knowledge of the fuzzy logic system.

Obtaining Preference Rules

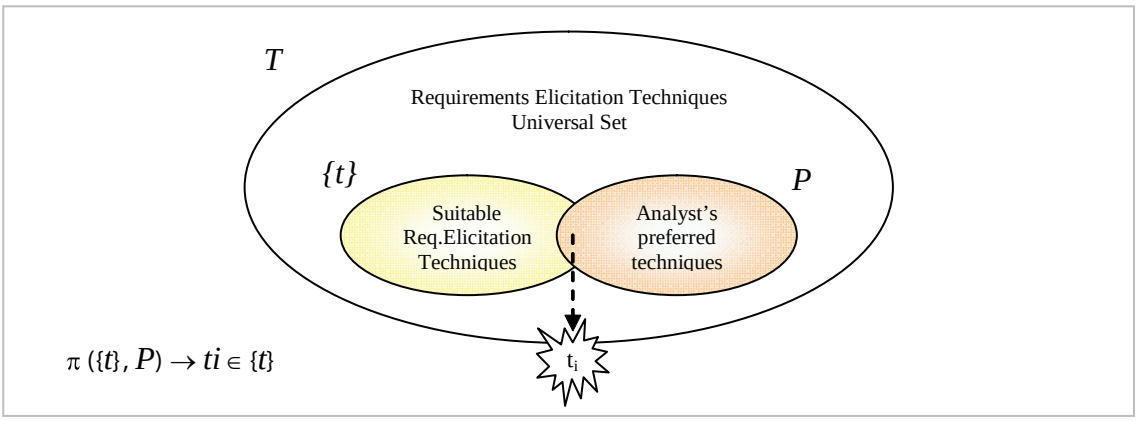
In order to support personal preferences toward groupware tools, in Aranda, Cechich, Vizcaíno, and Castro-Schez (2004) a model based on fuzzy logic and fuzzy sets to obtain rules from a set of representative examples has been proposed. The obtained rules represent patterns of behavior that indicate the preferences of stakeholders in their daily use of groupware tools and requirements elicitation techniques, according to their classification in the F-S model. To do so, we have collected examples of people preferences and applied a machine learning algorithm. The algorithm we chose (Castro, Castro-Schez, & Zurita, 1999) finds a finite set of fuzzy rules to reproduce the input-output system's behavior. Using this machine learning algorithm over a set of examples that represent the preferences of many stakeholders, we obtained a set of rules (Aranda, Vizcaíno, Cechich, Piattini, & Castro-Schez, 2006). The resulting set of preference rules can be applied to choose the best suite of groupware tools for a group of people by analyzing the results and combine them appropriately, as we will explain in the next section.

The Technology Selection Strategies

According to the analysis of real life projects, analysts are those who choose the techniques for requirement elicitation (Hickey & Davis, 2003). To model such a selection, Hickey and Davis' model considers a personal selector function π that returns a technique t_i, from a given a set of techniques $\{t\}$ and a set P that represents the personal preferences of the analyst. That means only the analyst's preferences are taken into account, as it is shown in Figure 4.

A first attempt to adapt the previous generic model to our cognitive point of view analyzes the preferences of all the stakeholders and chooses the technique that has more adherents (Aranda, Vizcaíno, Cechich, & Piattini, 2005). This extension of the π function, called π^*, is shown in Figure 5. In this formula PS_i represents a set of techniques that fit the *i-th* stakeholder's preferences (defined by the mechanisms based on fuzzy logic and fuzzy

Figure 4. Requirement technique selection according to analyst's preferences (π)



sets we have described previously), and $t_i \in \{t\}$ is the technique that appears in most of the PS_i. In this case, analysts are considered without any priority over the rest of the stakeholders.

A later improvement of π^* considers the relative importance of stakeholders' preferences by means of weighting them. Its purpose is that, if some stakeholders' preferences are stronger than the rest, the preferences that should be primarily considered are those of the first group of stakeholders. Also, the different weights might be used to prioritize preferences according to stakeholders' roles. The resulting function, called π^{**}, is shown in Figure 6. In this case, ws_i represents the weight (how strong the preferences are), and the resulting t_i is a technique that is appropriate for the current situation and is also appropriate for the stakeholder whose personal preferences are the strongest.

Figure 5. Requirement technique selection according to the most common preference (π^*)

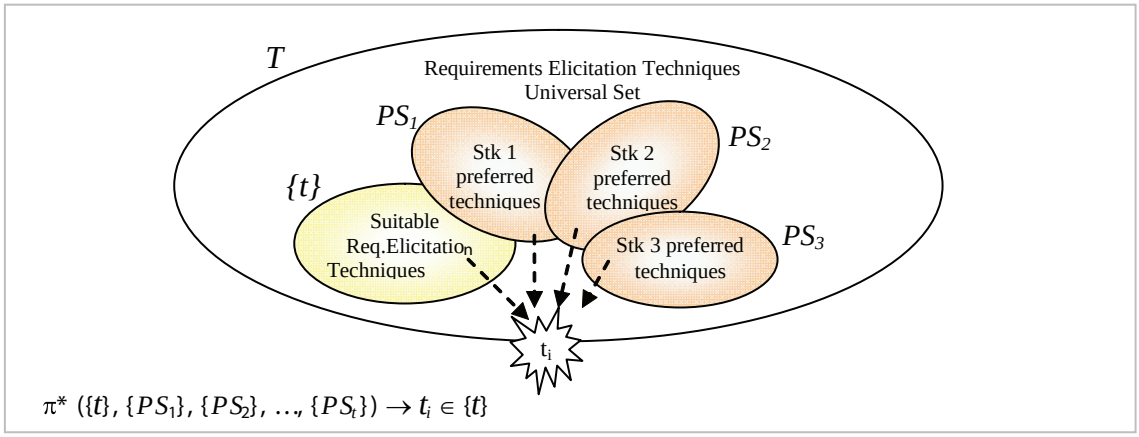
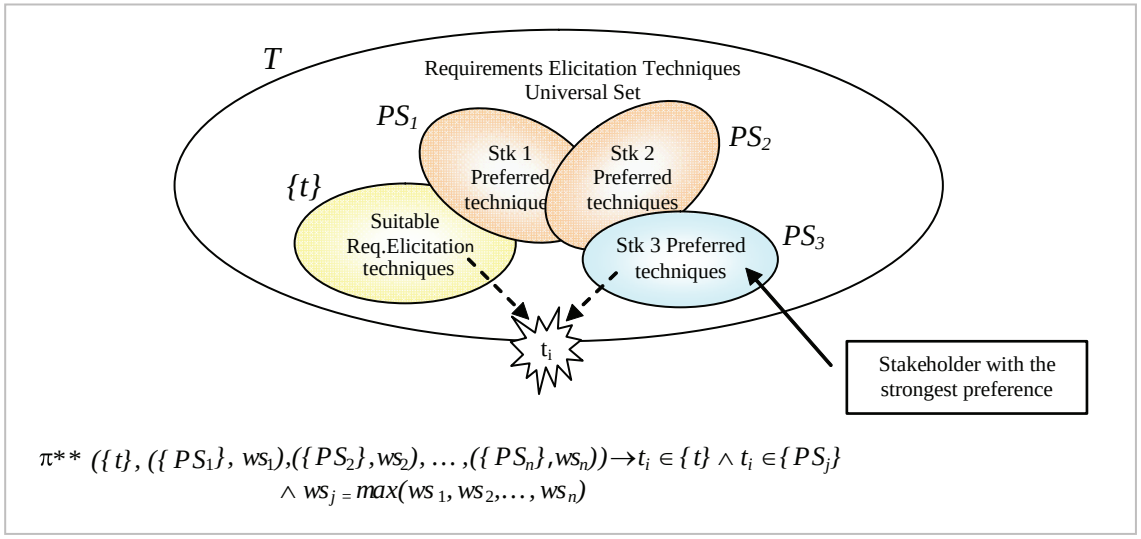


Figure 6. Requirement technique selection according to the strongest preferences in the group (π^{**})



PHASE 3: REQUIREMENT GATHERING

At the moment of deciding how to conduct the requirement gathering, the first step is deciding which of the possible techniques is best, given a current state of knowledge and a particular situation. To do so, the Hickey and Davis model defines the selector function σ (shown in Figure 7).

In a scenario where stakeholders are distributed along many geographically distanced sites, the selector function σ must also consider other aspects. The most important are time difference and the level of knowledge of a common language.

Time difference is important because when timetable does not overlap, or overlaps for a very short period of time, synchronic communication is not possible, then the selection process should prioritize those techniques that work better on asynchronous basis. Similarly, when stakeholders do not share the same language, some of them would need more time to read, think quietly, look for some vocabulary in the dictionary, and so forth.

With those considerations in mind, we extended the selector function σ as it is shown in Figure 8, where T_i (time difference) indicates in which degree synchronous communication is pos-

Figure 7. Definition of the requirements elicitation techniques most suitable for a given situation in a collocated environment (σ)

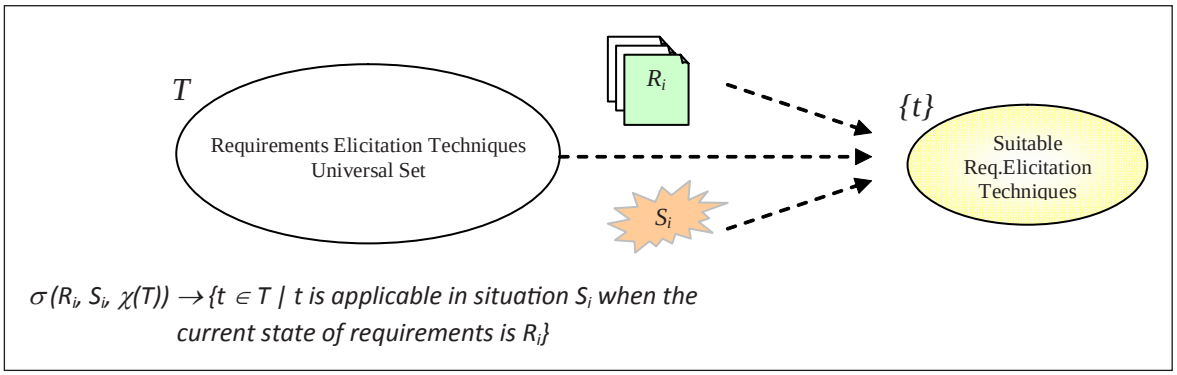


Figure 8. Definition of the requirements elicitation techniques most suitable for a given situation in a distributed environment (σ)*

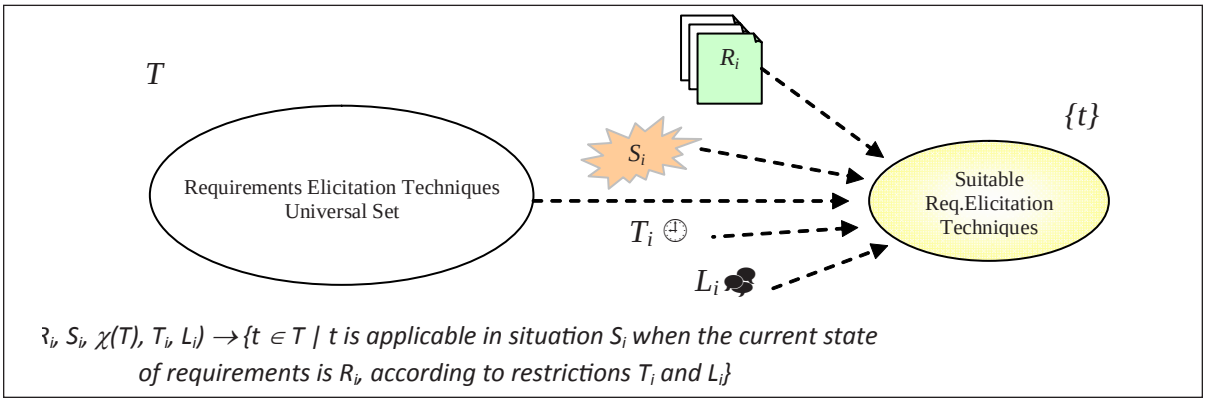
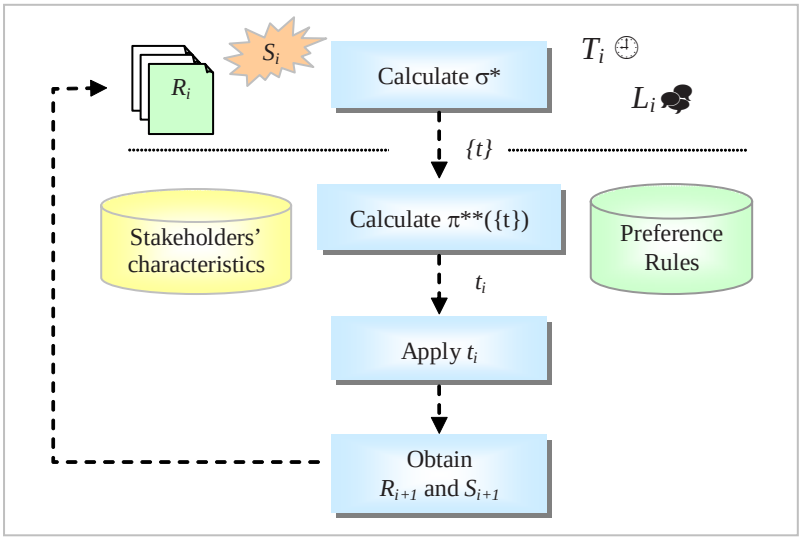


Figure 9. Requirement elicitation in GSD, as an iterative process of technique selection and application



sible between the sites that need to interact; and L_i (degree of knowledge of a common language) indicates the level of fluency of communication.

Finally, a graphical representation of the requirement elicitation process in a distributed environment is shown in Figure 9.

FUTURE TRENDS

For many years, research on GSD has focused on improving technology as a supporting media for spreading global communication. Those technology topics are really necessary indeed, but not enough for a society challenged by cultural diversity and worldwide-located working teams. Currently, research efforts are looking at GSD as a human-intensive process, where new strategies to deal with socio-cultural differences and distance are bringing psychological and cognitive aspects into the arena. Human factors seem more complex than technological ones, so probably we will see many strategies in the future trying to find the most effective way people understand, choose, and communicate during GSD. Certainly, cognitive informatics will be one of them.

CONCLUSION

GSD projects are a common way of work these days. Looking for solutions to improve communication in virtual environments has led us to analyse human interaction and how to apply well-known techniques from the field of cognitive psychology (called learning style models) as a base for technology selection.

Based on a cognitive-based technology selection approach, we propose a methodology for requirement elicitation in GSD projects, which focuses on improving communication between distant stakeholders. Considering cognitive aspects of stakeholders is significant because the selection of requirement elicitation techniques according to personal characteristics of all the members of a virtual team (instead of just the

analyst) might affect positively the quality of the information gathered during the requirement elicitation process. In addition, as stakeholders might feel more comfortable expressing themselves when using a groupware tool closer to the way they perceive and reason about the world, communication in virtual teams is expected to be more fluid and personal satisfaction higher.

In this article, we show the first three phases of such a methodology, called RE-GSD, which are the most clearly different from traditional requirement elicitation methodologies. Current work is focused on defining experiments, in academic and industrial scenarios, to analyze its performance.

ACKNOWLEDGMENT

This work is partially supported by the ENIGMAS (PIB-05-058), and MECENAS (PBI06-0024) project, Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, and the ESFINGE project (TIN2006-15175-C05-05) Ministerio de Educación y Ciencia, Dirección General de Investigación, Fondos Europeos de Desarrollo Regional (FEDER), from Spain. The CompetiSoft project (CyTED 3789); and the 04/E059 project, Universidad Nacional del Comahue, Argentina.

REFERENCES

Aranda, G., Cechich, A., Vizcaíno, A., & Castro-Schez, J. J. (2004). Using fuzzy sets to analyse personal preferences on groupware tools. In *Proceedings of the 10th Argentine Congress of Computer Science, CACIC 2004*, San Justo, Argentina, (pp. 549-560).

Aranda, G., Vizcaíno, A., Cechich, A., & Piattini, M. (2005). A cognitive-based approach to improve distributed requirement elicitation processes. In *Proceedings of the 4th IEEE International Conference on Cognitive Informatics (ICCI'05*), Irvine, USA, (pp. 322-330).

Aranda, G., Vizcaíno, A., Cechich, A., Piattini, M., & Castro-Schez, J. J. (2006). Cognitive-based rules as a means to select suitable groupware tools. In *Proceedings of the 5th IEEE International Conference on Cognitive Informatics (ICCI'06),* Beijing, China, pp.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer, 20*(4), 10-19.

Castro, J. L., Castro-Schez, J. J., & Zurita, J. M. (1999). Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems. *Fuzzy Sets and Systems, 101*(3), 331-342.

Chiew, V., & Wang, Y. (2003). From cognitive psychology to cognitive informatics. In *Proceedings of the Second IEEE International Conference on Cognitive Informatics, ICCI'03*, London, UK, (pp. 114-120).

Christel, M., & Kang, K. (1992). *Issues in requirements elicitation*. Pittsburgh, PA: Carnegie Mellon University.

Damian, D., & Moitra, D. (2006). Guest editors' introduction: Global software development: How far have we come? *IEEE Software*, *23*(5), 17-19.

Damian, D., & Zowghi, D. (2002). The impact of stakeholders geographical distribution on managing requirements in a multi-site organization. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE'02*, Essen, Germany, (pp. 319-328).

Ebert, C., & De Neve, P. (2001). Surviving global software development. *IEEE Software*, *18*(2), 62-69.

Felder, R., & Silverman, L. (1988). Learning and teaching styles in engineering education. *Engineering Education*, *78*(7), 674-681.

Gralla, P. (1996). *How Intranets work.* Emeryville, CA: Ziff-Davis Press.

Herbsleb, J. D., & Moitra, D. (2001). Guest editors' introduction: Global software development. *IEEE Software, 18*(2), 16-20.

Hickey, A. M., & Davis, A. (2003). Requirements elicitation and elicitation technique selection: A model for two knowledge-intensive software development processes. In *Proceedings of the 36th Annual Hawaii International Conference on Systems Sciences (HICSS),* (pp. 96-105).

Lanubile, F., Mallardo, T., & Calefato, F. (2003). Tool support for geographically dispersed inspection teams. *Software Process: Improvement and Practice, Wiley InterScience, 8*(4), 217-231.

Lloyd, W., Rosson, M. B., & Arthur, J. (2002). Effectiveness of elicitation techniques in distributed requirements engineering. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering, RE'02*, Essen, Germany, (pp. 311-318).

Loucopoulos, P., & Karakostas, V. (1995). *System requirements engineering.* New York, USA.

Martin, A., Martinez, C., Martinez, N., Aranda, G., & Cechich, A. (2003). Classifying groupware tools to improve communication in geographically distributed elicitation. In *Proceedings of the Ninth Argentine Congress on Computer Science, CACIC 2003*, La Plata, Argentina, (pp. 942-953).

Miller, J., & Yin, Z. (2004). A cognitive-based mechanism for constructing software inspection teams. *IEEE Transactions on Software Engineering, 30*(11), 811-825.

SWEBOK. (2004). *Guide to the software engineering body of knowledge*.

Togneri, D. F., Falbo, R. d. A., & de Menezes, C. S. (2002). Supporting cooperative requirements engineering with an automated tool. In *Proceedings of the Workshop em Engenharia de Requisitos, WER02*, Valencia, España, (pp. 240-254).

Wang, Y. (2002). On cognitive informatics. In *Proceedings of the First IEEE International Conference on Cognitive Informatics, ICCI'02,* Calgary, Alberta, Canada, (pp. 34-42).

KEY TERMS

Cognitive Informatics: It is an interdisciplinary area that applies concepts from psychology and other cognitive sciences to improving processes in engineering disciplines, such as informatics, computing, and software engineering.

CSCW (Computer-supported Cooperative Work): It is a research area that focuses on how people work together and the design and development of software (groupware) that may help their work as a group.

Distributed Software Development: It is a way of developing software that allows the stakeholders to be distributed in geographically distanced sites.

Global Software Development: When the distribution of the members of a distributed software development team exceeds the frontiers of a country.

Groupware: Software that supports and improves group work. It can be a simple text-based technology like e-mail or more sophisticated like videoconferencing.

Learning Style Model: It is a cognitive psychology theory that classifies people according to a set of behavioural characteristics pertaining to the ways they perceive and process information. They can be used to improve the way people learn a given task.

Requirements Elicitation: It is the first stage in the process of understanding the problem the

software has to solve. It is crucially based on human communication between the development team and the customer. Other terms that are used as synonyms are "requirements capture," "requirements discovery" and "requirements acquisition."

Stakeholders: They are all the actors that have some interest on a system. They can be the people that pay for it, work on its development or people whose task will be affected by the system, among others.

This work was previously published in Encyclopedia of Information Science and Technology, Second Edition, edited by M. Khosrow-Pour, pp. 3273-3282, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 5.20
Project Quality of Off-Shore Virtual Teams Engaged in Software Requirements Analysis:
An Exploratory Comparative Study

Dhruv Nath
Management Development Institute, India

Varadharajan Sridhar
Management Development Institute, India

Monica Adya
Marquette University, USA

Amit Malik
Management Development Institute, India

ABSTRACT

The off-shore software development companies in countries such as India use a global delivery model in which initial requirement analysis phase of software projects get executed at client locations to leverage frequent and deep interaction between user and developer teams. Subsequent phases such as design, coding and testing are completed at off-shore locations. Emerging trends indicate an increasing interest in off-shoring even requirements analysis phase using computer mediated communication. We conducted an exploratory research study involving students from Management Development Institute (MDI), India and Marquette University (MU), U.S.A. to determine quality of such off-shored requirements analysis projects. Our findings suggest that project quality of teams engaged in pure off-shore mode is comparable to that of teams engaged in collocated mode. However, the effect of controls such as user project monitoring on the quality of off-shored projects needs to be studied further.

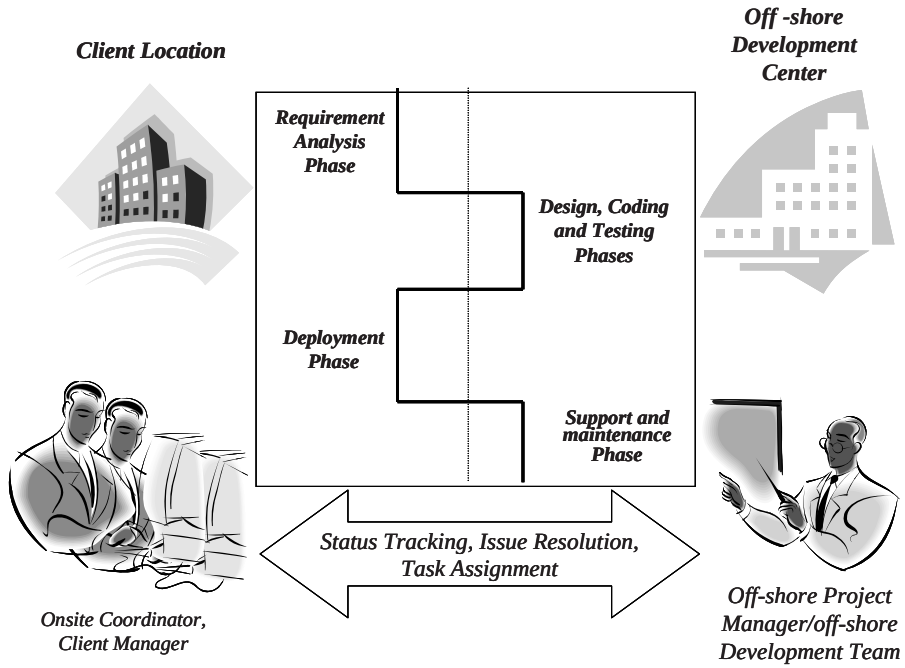
INTRODUCTION

The past two decades have witnessed significant globalization of the software development process with development rapidly moving away from the traditional collocated model, often called *on-site development,* to the off-shoring model. With the availability of increasingly skilled, flexible, and economical IT workforce in countries such as India, Malaysia, and China, it makes financial sense for United States and European client organizations to execute a significant portion of software projects in these countries. This growing trend towards off-shoring has, in turn, spurred growth in many Asian nations, creating improved economic and IT infrastructure and enhancing the viability of these countries as software service providers. For example, India has emerged as a dominant off-shore software development industry with revenue of about $16.7 billion, which is projected to reach $60 billion by the year 2010 (Carmel, 2006; National Association of Software and Service Companies, 2005).

The Indian off-shore software industry has matured over the years, and process capability has been steadily improving. Coordination and communication problems typically encountered in off-shore development (see Battin, Crocker, Kreidler, & Subramanian, 2001, for an extended discussion), are mitigated by the use of processes such as rational task assignments and liaisoning, and tools such as centralized bug reporting system and software configuration management platforms. A case in point is India's Infosys Technologies, which has significantly leveraged time zone differences with its clients by modifying its organizational culture, processes, and communication technologies (Carmel, 2006).

The typical off-shore development model, followed successfully for over a decade by many Indian software companies such as Infosys, Wipro, TCS, and Satyam, is illustrated in Figure 1.

Figure 1. The off-shore software development model



Requirements analysis refers to that stage of the system development life cycle wherein the information and information processing services needed to support select objectives and functions of the organization are (i) determined and (ii) coherently represented using well-defined artifacts such as entity-relationship diagrams, data-flow diagrams, use cases, and screen prototypes (Hoffer, George, & Valacich, 1999). As suggested in Figure 1, typically this phase is conducted at the client location, since this phase requires frequent and significant interaction between users and developers. Business and systems analysts are physically located at the client site to perform this activity. Global projects consultant teams from off-shore location travel to the user site to gather and analyze requirements in face-to-face meetings (Damian & Zowghi, 2002). The consultants then communicate the requirements to the development staff in the offshore site. Depending on the nature of the project, high-level design is conducted in both on-site and off-shore mode due to comparatively lower interaction needs with the client. Detailed design, coding, and testing are executed at the off-shore site. Off-shore vendors also deploy liaisons who coordinate activities between on-site users and the off-shore development team. These liaisons are critical for effective communication and coordination between users and developers (Battin et al., 2001).

Increasingly, both client and software providers are now considering the possibility of off-shoring the requirements analysis phase, traditionally done on client site, away from the client location. In such a scenario, analysts and developers located at the off-shore location would interact in a virtual mode with the clients situated at their premises to determine and structure the requirements. Such a shift could potentially improve the cost arbitrage of the projects for instance by cutting down travel costs incurred for sending analysts to the client site for face-to-face meetings. In an extreme case, the entire team of analysts and developers could be based in off-shore location such as India while the client could be in Europe or the United States. Requirements gathering would then be conducted between these virtual teams using existing computer-mediated communications such as chat, e-mail, and video conferencing. The questions of research interest then are:

1. Can requirements analysis conducted by collocated teams using face-to-face communication be comparable or better than those produced by virtual off-shore teams using computer-mediated communication?
2. What forms of control are necessary to facilitate high-quality outcomes from virtual requirements analysis undertakings?

Using theories of social presence, media richness (Burke & Chidambaram, 1999), as well as control theory (Kirsch, 2002), we develop and test hypotheses regarding these questions.

Traditionally, user involvement in IS projects has been an important contributor to project success (Hartwick & Barki, 1994; Foster & Franz ,1999; Lin & Shao, 2000; Sridhar, Nath, & Malik, in press). Lack of user proximity in a virtual setting can potentially limit the quality of requirements elicitation due to limitations of communications media. In order to mitigate these limitations and the absence of analysts and developers at customer premises, user involvement is expected to take the form of close project monitoring and control to ensure that requirements and project goals are met. Control theory provides the required theoretical foundations for analyzing the effect of different types of controls on teams (Crisp, 2003). In this study, we specifically consider user project monitoring as a behavioral control mechanism and examine its impact on project quality during requirements analysis phase of off-shored software projects. Further, we explore the intersection of media richness and control theories to find early answers to the research questions raised earlier.

This study is exploratory in nature. Without loss of generality, we restrict our attention to the requirements analysis phase as defined in the structured systems analysis and design (SSAD) methodology as defined by Hoffer et al. (1999). We define requirements analysis as subsuming the following two phases:

1. Requirements determination: The process by which the analysts determine the requirements of the system from the users through discussions and interviews and exchanging forms, reports, job descriptions and other necessary documents.
2. Requirements structuring: The process by which the analysts coherently represent the information gathered as part of requirements determination using process modeling and logic modeling tools as described in SSAD.

Our interactions with managers in client firms engaged in software development indicate that off-shoring of requirements analysis is still uncommon. Hence it is not practical to analyze this phenomenon of pure off-shoring of requirements in real-life setting. It is also difficult to do in-depth longitudinal or cross-sectional case studies. Given these arguments, an exploratory research study was conducted in an academic setting involving management students enrolled in a graduate-level information systems course at Management Development Institute (MDI), India, and management students enrolled in a graduate level IT Project Management course at Marquette University (MU), United States. MU students role-played as virtual users/project managers while MDI students were software developers for MU teams as well as user clients for collocated MDI teams. Prior to a full description of our undertaking, we first discuss existing literature on virtual teams in software projects. We then describe the theoretical foundations of this study and elaborate on our research design. Next, we discuss our measures and discuss study outcomes. The article concludes with implications for future research in this context.

VIRTUAL TEAMS IN SOFTWARE PROJECTS

In a pure off-shore mode, users at the client location and the developers at the off-shore location never meet face to face and hence operate as virtual teams, primarily linked through technology across national boundaries. It is in this context that we review previous research on such virtual teams, specifically those engaged in software development projects. Virtual teams are becoming the norm in most corporate environments such as consulting firms, technology products, and e-commerce (Lurey & Raisinghani, 2001) and are being increasingly examined in academic literature (see Powell, Piccoli, & Ives, 2005 for a comprehensive survey of virtual teams). Battin et al. (2001) described how Motorola deployed global virtual teams across six different countries for a Third Generation Cellular System product development. Software development in Alcatel was handled by a central group of several thousand engineers distributed throughout the world (Ebert & De Neve, 2001).

Few studies however, have, examined the use of virtual teams for requirements analysis. Edwards and Sridhar (2005) studied the effectiveness of virtual teams in a collaborative requirements analysis practice. In that study virtual teams at near and far locations participated in requirements analysis phase of the project. This typically is applicable in collaborative global product development exercises as described in Battin et al. (2003). In contrast, in this study we look at the requirements analysis phase of off-shored software projects in which the two protagonists are (i) users who specify the requirements, and (ii) developers who determine and document these requirements together constituting a col-

laborative virtual teams. Damian and Zowghi (2002) studied the interplay between culture and conflict and the impact of distance on the ability to reconcile different viewpoints with respect to "requirements negotiation" processes. They found that lack of a common understanding of requirements, together with reduced awareness of local context, trust level, and ability to share work artifacts significantly challenge effective collaboration among remote stakeholders in negotiating a set of requirements that satisfies geographically dispersed customers. Damian, Eberlein, Shaw, and Gaines (2000) examined the effect of the distribution of various stakeholders in the requirements engineering process. They found that highest group performance occurred when customers were separated from each other and collocated with the facilitator or system analyst. Our study further contributes to the literature on virtual teams engaged in off-shored software requirements analysis.

THEORETICAL FOUNDATIONS AND HYPOTHESES DEVELOPMENT

Social Presence and Media Richness Theories

Social presence is the extent to which one feels the presence of a person with whom one is interacting. Short, Williams, and Christie (1976) suggested that some media convey greater social presence than others. For instance, face-to-face interaction is considered to be high in social presence, primarily because of the capacity of the medium to transmit proximal, facial, and other nonverbal cues relative to other media. In contrast, computer-mediated communication such as e-mail exhibit inherently lower bandwidth than face-to-face interaction, thus permitting transmission of fewer visual and nonverbal cues and restricting socio-emotional communication (Rice & Love, 1987). In addition to differences in social presence, media richness theory proposes that, given their limited cue-carrying capacity, leaner media such as e-mail, will be less effective for groups performing ambiguous tasks which require a variety of cues to be exchanged. However, Burke and Chidambaram (1999) pointed out that despite some support for media characteristics-dependent theories, overall empirical evidence has been mixed.

Quality of Off-Shored Projects vs. Collocated Projects

Teams engaged in pure off-shored projects primarily rely on computer-mediated communications (synchronous such as chat, audio and video conferencing as well as asynchronous such as e-mail) for interaction. However, collocated teams have the luxury of rich face-to-face communication. Based on the social presence and media richness theories, we formulate the following hypothesis:

H1: *Collocated teams using face-to-face communication will produce higher quality project artifacts compared to virtual teams using computer-mediated communication during the requirements analysis phase of software projects.*

In a subsequent section, we define *quality of project artifacts* and how it is measured. To the best of our knowledge, quality of projects and performance of virtual teams engaged in the software requirements analysis has not been studied in the literature thus far. Although several researchers have compared performances of traditional collocated teams with that of virtual teams, the conclusions have been mixed. While one study reported greater effectiveness for virtual teams (Sharda, Barr, & McDonnell 1988), others such as McDonough, Kahn, and Barczak (2001) have found that virtual teams could not outperform traditional teams. Andres (2002) reported that teams working in face-to-face settings experienced greater productivity compared to those supported using videoconferencing. Generally,

computer-mediated teams exhibit lower frequency of communication than face-to-face teams, although they tend to exchange more task-oriented messages as a proportion of total communication (Burke & Chidambaram, 1999; Chidambaram, 1996). This enhanced communication leads to comparable or even higher performance of virtual teams as compared to collocated teams (Burke & Chidambaram, 1999). Consistent with these findings, Schmidt et al. (2001) reported that virtual teams are more effective in new product development decisions as compared to face-to-face teams. However, a majority of the early work has detected no difference between the two types of teams (Burke & Aytes, 1998). Other studies have found no significant differences between traditional and virtual teams when examining decision quality (Archer, 1990; Chidambaram & Bostrom, 1993) as well as the number of ideas generated by decision making teams (Archer, 1990; Lind, 1999; Sharda et al., 1988). Walther (2005) further suggested that complex human processes such as negotiation actually improve between physically distributed individuals who communicate using media low in richness. Studies comparing performance of virtual and collocated teams in software requirements analysis phase are even fewer. Damian et al. (2000) found that groups in face-to-face meetings performed no better than the electronically mediated groups in the requirements negotiation phase of the software development life cycle.

Control Theory

Control is defined as the set of mechanisms designed to motivate individuals to work in such a way that desired objectives are achieved (Kirsch, 1996). *Formal controls* rely on mechanisms that influence the controllee's behavior through performance evaluation and rewards (Choudhury & Sabherwal, 2003). Controllers utilize two modes of formal control: behavior and outcome (Kirsch, 2002). In behavior control, appropriate steps and procedures for task performance are defined by controllers, and then controllees' performance is evaluated according to their adherence to the prescribed procedures. In outcome control, controllers define appropriate targets and allow controllees to decide how to meet those output targets. Controllees' performance is evaluated on the extent to which targets were met, and not on the processes used to achieve the targets (Kirsch 2002).

Informal control mechanisms utilize social or people strategies to reduce goal differences between controller and controllee. Self-control, one mode of informal control, occurs when an individual sets up his or her own goals, self-monitors goal achievement, and rewards or sanctions him- or herself accordingly (Kirsch, 2002). *Clan control*, the other type of informal control, is implemented through mechanisms that minimize the differences between controller's and controllee's preferences by "promulgating common values, beliefs and philosophies within a clan, which is defined as a group of individuals who are dependent on one another and who share a set of common goals" (Choudhury & Sabherwal, 2003). Kirch et al (2002) extended the control theory to the role of client liaisons, exercising control of IS project leaders to ensure that IS projects meet their goals. The study examined the conditions under which client liaisons of IS development projects choose various modes of control. In a related work, Choudhury and Sabherwal (2003) examined the evolution of portfolio of controls over the duration of outsourced IS development projects. They conclude that in outsourced software projects outcome controls are exercised at the start of the project. Behavioral controls are added later in the project. Clan controls are used when the client and vendor had shared goals, and when frequent interactions lead to shared values. Both these studies analyzed the evolution and choice of controls in IS projects and not on the effect of these controls on project outcome. In this study we focus on the effect of formal modes of

control (both outcome and behavior) on the quality of project artifacts produced by virtual teams engaged in software requirements analysis. Project monitoring provides opportunities for both forms of formal control previously described through tracking, interpretation and transmission of status information (Crisp, 2003). In this study, we define *user control* to include not only monitoring the project plan (a form of behavioral control) but also the evaluation of the formal artifacts produced (a form of outcome control) during the requirements analysis process. Monitoring of costs is excluded as requirements analysis is often part of a large IS outsourcing project. Though cost monitoring is vital, it does not assume much significance when considered for only one phase of the project and hence is excluded. Based on the control theory and literature review of virtual teams, our second hypothesis is as follows:

H2: *Developer teams that are closely monitored by their users in a virtual team mode will produce higher quality of artifacts as compared to developer teams that are not closely monitored by their users.*

RESEARCH DESIGN

To test both the aforementioned hypotheses, we conducted two overlapping quasi experiments involving students at MU and MDI in controlled settings. Such experimental settings have been actively used in distributed software engineering laboratories and business schools to conduct virtual team exercises in their courses (Powell et al., 2005). A controlled experimental approach provides three benefits. Firstly, it makes available several teams that work in parallel, thereby generating rich data for drawing conclusions. Secondly, it permits researchers to experiment with newer approaches, which may not yet have been explored by the industry. Finally, it equips and trains software engineering students to understand and to handle the challenges of working in global software teams (Favela & Pena-Mora, 2001). A survey on virtual team research by Powell et al. (2005) cited 28 academic experiments and only 13 case study research papers. Our experimental setup is illustrated in Figure 2 and described in greater detail next.

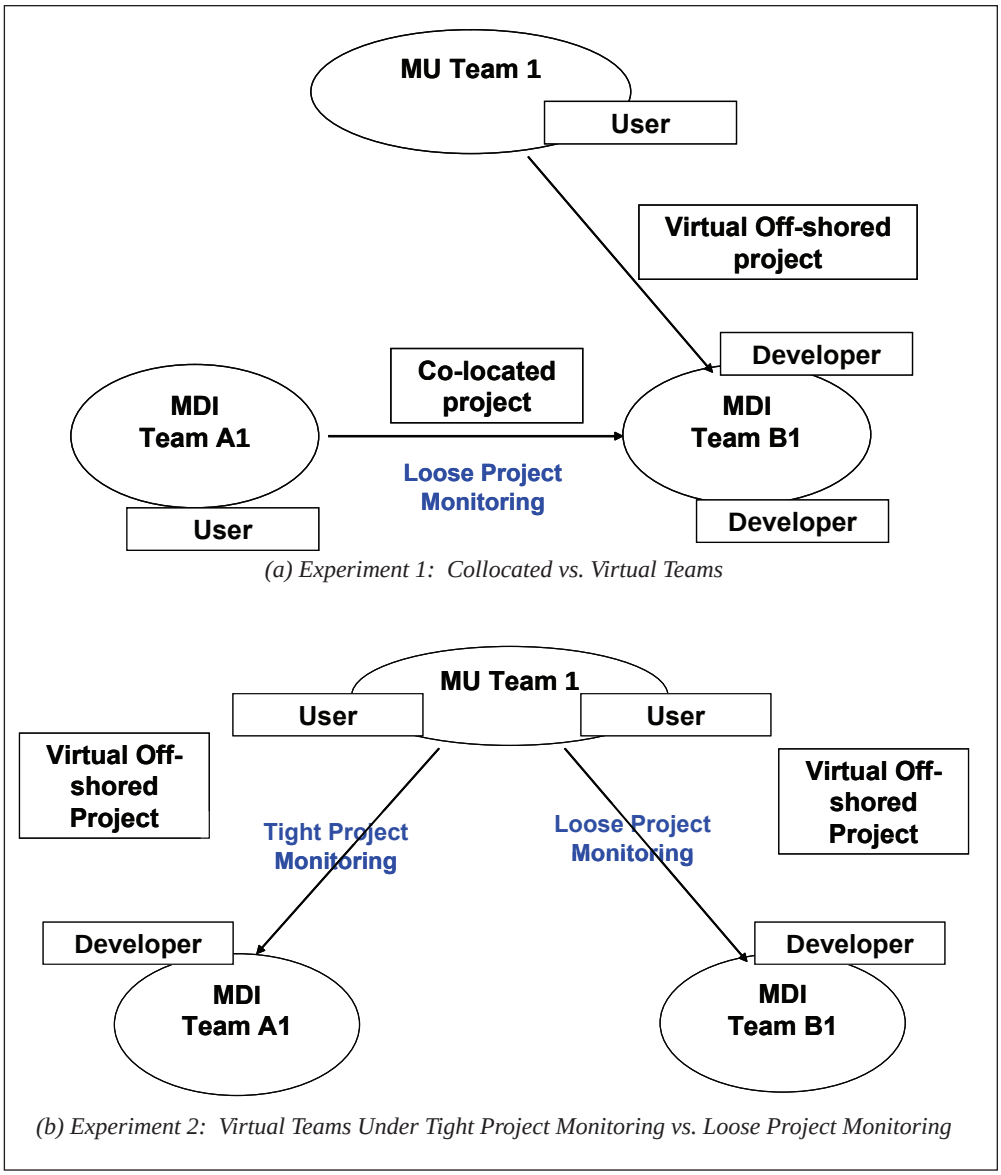
Experiment 1—Testing H1: The Impact of Media Richness on Project Quality

For hypothesis 1 (H1), we compared the quality of projects produced by collocated teams with those that were produced by virtual teams. The collocated teams were students of the postgraduate program in management (equivalent to an MBA) who were attending a core course in management information systems (MIS) at MDI. One hundred and twenty-seven students were divided into two roughly equal sections, section *A* and section *B*. Students from section *A* were grouped into 10 teams of 5 or 6 students each. Each team played the role of users for the collocated project. Figure 2(a) shows one such team, referred to as MDI team A1. Students from section *B* were also grouped into 10 teams of 5 or 6 students each. Each of these teams formed developer teams for the collocated project. Figure 2(a) shows one such team, referred to as MDI team B1. Each MDI *A* team was then paired with one of the MDI *B* teams, as shown in Figure 2(a). Thus MDI team A1 served as users to MDI team B1, the developers in the collocated project. Similarly, MDI team A2 was the user for MDI team B2, and so on.

Setting for the Virtual Teams

MU students, enrolled in a graduate elective course in IT project management, assumed the role of virtual users. Twenty-eight students divided into 10 teams (each with a team size of 2-3 members), referred to as *MU Teams*. Figure 2(a) shows one of these MU teams, team 1. Each MU

Figure 2. The experimental set-up



(a) Experiment 1: Collocated vs. Virtual Teams

(b) Experiment 2: Virtual Teams Under Tight Project Monitoring vs. Loose Project Monitoring

team was paired with one of the MDI B teams. Thus MDI B teams became the off-shore development teams for the associated MU user teams. These teams consisting of users and developers worked in virtual team mode. In summary, each MDI team *B* was involved in the following two projects: (i) collocated project with MDI user team *A* and (ii) virtual off-shored project with MU user team.

In both projects, the MDI *B* teams were required to submit a project plan at the beginning of the project, detailing various activities and timelines. The final delivery date was predetermined by the instructors based on the course schedule. Project

monitoring was voluntary between MDI B and MDI A user teams, so as to minimize the impact of any other variables on the experiment. The MDI B development teams communicated with their corresponding user teams at MU through online means such as e-mail, instant messaging, and voice chats such as Skype and with their MDI A user teams through face-to-face meetings while having face-to-face interactions with their collocated MDI *A* teams. It must be noted that each developer teams (i.e., MDI *B* teams) had 5 or 6 members, thus controlling for the effects of team sizes on the quality of the project.

Experiment —Testing H2: The Impact of Project Monitoring on Project Quality

To test H2, we compared the quality of two sets of virtual teams, one in which the users imposed project monitoring (referred to as *tight* monitoring), and the other one in which user project monitoring was voluntary (referred to as *loose* monitoring). For this purpose, we used a portion of the data collected as part of experiment 1. Recall that in experiment 1 we already had a set of virtual teams, namely the teams formed by MU user team and the MDI *B* developer teams, operating in voluntary project monitoring mode. We then formed another set of virtual teams by pairing each MU user team with MDI *A* teams. However, in this experiment MDI *A* teams performed the role of developers for their corresponding MU user teams (compared to the role of users they played in experiment 1). MU user teams were required to tightly monitor their projects with MDI *A* teams. This is illustrated in Figure 2(b), where MU team 1 was the user for MDI team A1, under tight project monitoring, and was also the user for MDI team B1, under loose project monitoring (part of experiment 1). Similarly, the MU team 2 was the user for MDI team A2 and B2, and so on. Once again, each of the developer teams (i.e., MDI *A* and *B* teams) had 5 or 6 members, thus controlling for effects of team size on success of the project.

Tight and loose control was implemented as follows: In the case of virtual teams operating under imposed tight project monitoring (MU and MDI *A* teams), the developers were told to submit weekly project reports to their respective user teams. The user teams were required to review and ask for changes/actions as required, thus implementing behavioral control. In addition, MDI teams were required to conduct requirements analysis in iterative model, returning a set of intermediate artifacts which would also be reviewed and commented on by their users, thus implementing outcome control. This formed the control group in our experiment. In contrast, teams operating under voluntary user project monitoring did not have to submit regular project status reports nor any intermediate artifacts to their users. They received requirements specifications from their users, asked for clarifications where necessary, and submitted the final artifacts at the end of the project. Any communication between these teams and their users was strictly on a need-be basis. This formed the experimental group in our research design. MU teams were graded partly on the communication plans and weekly project status reports they developed for monitoring their MDI *A* teams. This ensured that MU team users spent more time and effort in monitoring their associated MDI *A* teams than MDI *B* teams. This design resulted in the two overlapping experiments 1 and 2 described previously. Table 1 illustrates the roles of MDI and MU teams in these experiments.

All student teams were formed in such a way that the technical background and average work experience of group members were almost the same across groups, thereby controlling team member heterogeneity. Table 2 provides ANOVA results comparing means of various parameters across teams. Results suggest no significant differences in the means of various parameters across teams confirming their homogeneity. Students

Table 1. Experimental set-up

	MU Teams	MDI *A* Teams	MDI *B* Teams	Treatment	Hypothesis Tested
Experiment 1	Users	Users	Developers	MU Users <-> MDI *B* Developers, Virtual Teams MDI *A* Users <-> MDI *B* Developers, Collocated Teams	H1
Experiment 2	Users	Developers	Developers	MU Users <-> MDI *A* Developers, Virtual and tightly controlled Teams MU Users <-> MDI *B* Developers, Virtual and loosely controlled Teams	H2

Table 2. ANOVA comparing means of variables across teams

Variable	*F*	Significance
Work Experience	0.601	0.795
Experience in Programming	1.356	0.213
Experience in Participating in Virtual Teams	0.803	0.614
Experience in Software Project Management	0.973	0.465
Experience in Systems Analysis and Design	0.543	0.841

had sufficient stake in the virtual team project as up to 30% of the course grade was assigned to the project.

Our research design adopts the quasi experiment approach where the participants are allotted to teams, based on certain criterion, as explained previously, and not randomly. Hence the limitations of quasi experimentation as explained in Campbell and Stanley (1966) applies to our research setting as well.

Tasks

Virtual Team Exercise

The virtual team interactions (in both experiments) were broken down into two phases: (1) *socialization*, which permitted the teams to develop relationships and negotiate communication terms and requirements; and (2) *project execution*, which allowed requirements gathering, clarifications, and exchange of analysis artifacts.

Phase 1: Socialization

It is an increasingly common practice in virtual teams to engage in formal socialization before embarking on virtual projects in order to understand each others' work styles and expectations, negotiate communications strategies and protocols, and build trust for sustained relationships (Jarvenpaa & Leidner, 1999). In our experiment, this was not feasible due to resource and other restrictions, not unlike those faced by organizations new to off-shoring as well as those involved in small, preliminary initiatives. Furthermore, our objective was to draw benchmark conclusions regarding effects of user project monitoring on teams engaged in a fully virtual team environment. Therefore we encouraged the MU and MDI teams to communicate and socialize with each other on-line before initiating actual work on the project. The virtual teams—MU, MDI *A*, and MDI *B*—socialized with each other using on-line media such as e-mail, Internet chat,

bulletin boards, and e-groups for a period of 2 weeks. Project details were withheld from all teams till conclusion of the socialization phase in order to ensure that communication was more personalized and oriented towards relationship and trust building (Sarkar & Sahay, 2002) rather than requirements exchange.

Phase 2: Project Execution

Subsequent to socialization, the projects were initiated, and team roles were detailed. Marquette University has a service learning office that obtains information systems projects from nonprofit organizations and small businesses in and around Milwaukee. Such real-life projects were given to MU users. Examples of these projects include a donation management system for a nonprofit organization, a volunteer management system, an alumni website, a tracking system for battered and abused women, and a book inventory management system.

The MDI teams elicited project requirements from MU teams through various on-line media, as described previously. SSAD methodology was used in the experiment. The gathered requirements were structured using process modeling tools such as context analysis diagram (CAD), data flow diagrams (DFDs) and process specifications. MDI teams also modeled the data and associated relationships using entity relationship diagrams (ERDs). MDI teams also created screen-based

Table 3. Artifacts submitted by the different teams for the virtual team projects

Artifact	**MDI *A* Teams for the Virtual Team Projects under tight Project Monitoring**	**MDI *B* Teams for both the Virtual and Collocated Projects under loose Project Monitoring**	**MU Teams for the Virtual team Projects (to be submitted to the instructors)**
Context Analysis Diagram	✓	✓	
Data Flow Diagrams	✓	✓	
Entity Relationship Diagrams	✓	✓	
Process Specifications	✓	✓	
Screen shots	✓	✓	
An intermediate version of all the above artifacts	✓		
Weekly Development Status Report	✓		
Communication Plans			✓
Risk Assessment			✓
Contingency Plans			✓
Weekly Project Status Report (to the Instructors)			(only with MDI *A* teams)
Project Closure Report			✓
Team *A* and *B* Assessment			✓

prototypes as part of the requirements analysis exercise. These artifacts were submitted by the MDI teams to MU user teams as part of the deliverables.

In addition, the MDI A development teams that experienced tightly monitored projects submitted the following additional artifacts to the users:

a. A weekly status report of the project, explaining reasons for delays and plans for overcoming any slippages.
b. Any modifications to the project plan.
c. A draft (intermediate) version of all the above artifacts, midway through the project Based on their requirements, users provided feedback and corrections, which were incorporated by the developers into the final version.

Details of all these deliverables submitted by the different teams for this virtual team exercise are shown in Table 3. The table also shows several artifacts/reports that the MU teams had to submit to the course instructors.

Collocated Exercise

For the collocated team exercises, each MDI *A* team had at least one member who had prior work experience of 2 to 3 years. These individuals were asked to select an information system project they had encountered at work, to ensure realism and familiarity with system features. Each collocated team developed requirements analysis artifacts for these projects. The instructors had discussions with each group and scoped the projects such that the project complexity was almost the same as that of the virtual teams. The MDI *B* teams were asked to submit to MDI team *A* artifacts identical to those submitted to MU teams during the virtual team project (see Table 1). The entire project duration for both virtual and collocated projects was 8 weeks.

OUTCOME MEASURES

Quality of Projects

Quality of MU-MDI projects were determined through (i) expert evaluation of project artifacts produced by developer teams and (ii) user perceptions about the project deliverable quality. Quality of project artifacts was measured on several dimensions—namely, correctness of the artifacts (e.g., whether the data flow diagrams were drawn correctly, whether or not they satisfied user requirements), adherence of the artifacts to user requirements, and consistency of the artifacts with each other.

i. Completeness and Adherence of the Artifacts to User Requirements

Completeness and adherences were analyzed by an external expert who was not part of the MU-MDI teams. This expert had 2 to 3 years of experience in software projects and had taken courses in SSAD. The expert evaluated the completeness and adherence of each of the following artifacts:

1. Context analysis diagram
2. Data flow diagrams (DFDs)
3. Process specifications
4. Entity-relationship diagrams (ERDs)
5. Screen shots of the proposed system

The expert analyzed and scored the above artifacts for each project on a 7-point Likert-type scale. Though the expert had only 2 to 3 years of experience, by following a standard evaluation procedure such as the one outlined previously, this individual was able to arrive at an objective assessment of project quality. This evaluation was validated for consistency and accuracy by a second expert who had more than 20 years of SSAD industry experience, thus reducing possible

biases in the evaluation process. The average of these scores across all artifacts for each project was taken as a measure of completeness and adherence of project artifacts to user requirements. By making the team assignments to the projects blind to the expert, we minimized subjective bias of the expert during the assessment.

ii. Consistency of the Artifacts

The expert also analyzed the consistency of the screen prototypes submitted by development teams with the DFDs and ERDs submitted. Using a 7-point Likert-type scale, the expert analyzed and scored for each project the consistency across

1. Screen prototypes and DFDs
2. Screen prototypes and ERDs

Using the same evaluation and validation procedure described in (i), an average score measuring the consistency of the project artifacts was generated.

iii. User-perceived quality

User perceptions about the quality of artifacts submitted by the developer teams were also collected through a survey questionnaire as the third measure of team performance. A 7-point Likert-type scale was used to elicit response from the user team members. Items adapted from Edwards and Sridhar (2005) are detailed in Appendix I. Scores given by all the users to a particular development team were averaged and were treated as measure of user-perceived quality. Therefore, there was one rating/score per user teams. Based on measures of quality already mentioned, hypothesis H1, which was constructed in the previous section, can be refined and are presented in Table 4.

By specifying the two dimensions of completeness and adherence as well as consistency, any errors in the assessment of the quality of the projects was thought to be minimized.

Table 4. Detailed hypotheses based on different measures

Research Question	Hypotheses
Quality of Projects of Virtual Teams vs. Collocated Teams	**H1a:** Adherence and completeness of the requirements analysis artifacts produced by the collocated teams using face-to-face communication will be better than those produced by the virtual teams using computer-mediated communication. **H1b:** Consistency of the screen shots and requirements analysis artifacts produced by the collocated teams using face-to-face communication will be better than those produced by the virtual teams using computer-mediated communication. **H1c:** The users will perceive the quality of project artifacts produced by collocated teams using face-to-face communication to be better than those produced by the virtual teams using computer-mediated communication.
Impact of User Project Monitoring on of the Quality of Projects	**H2a:** Quality of project artifacts (as defined by the three measures of completeness & adherence, consistency, and user perception) produced by the developer teams that are closely monitored by their associated users in a virtual team mode will be better than those produce by the developer teams that were not closely monitored by their users.. **H2b:** Quality of project artifacts (as defined by the three measures of adherence & completeness, consistency, and user perception) produced by of the developers that perceived higher levels of project monitoring by their users will be better than those produced by the developer teams that perceived lower levels.

User Project Monitoring

We also measured perceived project-monitoring practices of all users and developers involved in both tight and loosely monitored projects. Responses were elicited on a 7-point Likert-type scale at the end of the project. Items are shown in Appendix I. In order to capture the responses for perceived quality and user project monitoring based on the roles they played (user/developer) and the team (collocated/ virtual) with which they did the projects, different versions of the survey was prepared and administered to students at MDI and at MU. The various versions included same items for each construct but were worded differently, depending on the roles the participants played. Based on the experimental measure of perceived project management practice, hypothesis H2 can be further articulated as in Table 4.

ANALYSIS, RESULTS, AND DISCUSSIONS

A principal component analysis was performed on the items constructed for the previously mentioned measures with Varimax rotation and Kaiser normalization; the results are given in Table 5. Reliability of all these measures of (i) completeness and adherence of artifacts, (ii) consistency of project artifacts, (iii) user-perceived quality, and (iv) perceived user project monitoring practices are given in Table 6. Cronbach's alpha values of 0.70 and higher indicate construct reliability.

Performance of Collocated vs. Virtual Teams

To test hypothesis **H1,** a one-way ANOVA test was performed on the three measures of project quality, as were previously described, between virtual and collocated teams that participated in Experiment 1. Note that in this case the project artifacts are produced by the same developer teams, and the project complexity of both the virtual and collocated projects were moderated by the instructors to be almost the same. However, due to constraints in conducting the experiment, the user teams could not be the same. User project monitoring was kept loose for both virtual and collocated projects. ANOVA results are represented in Table 7.

Results indicate that all the variations (H1a, H1b and H1c) of hypothesis H1 can be rejected. Although two of the mean quality measures of collocated teams are better than that of virtual teams, they are not significantly different. This is contrary to expectations that the quality of projects that are produced by collocated teams and that benefit from higher social presence, media-rich face-to-face communications is no better than that produced by virtual teams that use lean media. This potentially suggests that the requirements analysis phase of software projects may be successfully off-shored in full and conducted in virtual team mode without significantly affecting the quality of projects.

Effect of User Project Monitoring

To test H2a, we compared mean values of the quality measures between the tightly monitored control group and the loosely monitored experimental group. Results presented in Table 8 indicate that the completeness and adherence of project artifacts produced by the control group were significantly superior to those produced by the experimental group, suggesting that close project monitoring by users had a positive impact on this measure of project quality. However, neither the consistency of project artifacts nor the user-perceived quality differed significantly across the two sets of teams. As expected, participants in the control group perceived that their projects were indeed closely monitored, compared to those in the experimental group.

Table 5. Principal component analysis of various constructs indicating factor loadings of survey items

Item No	Adherence and Completeness of Project Artifacts	Consistency of Project Artifacts	User-Perceived Quality	Perceived User Project Monitoring
1	.792	.892	.882	0.663
2	.869	.885	.935	0.845
3	.699		.871	0.700
4	.400		.956	0.615
5	.680			0.759
6				0.548
7				0.686

Note. Extraction method: principal component analysis; rotation method: varimax with kaiser normalization

Table 6. Reliability coefficients (Cronbach's Alpha) of constructs

Constructs (Number of items)	Cronbach's Alpha Value
Completeness and Adherence of Project Artifacts (5)	0.70
Consistency of Project Artifacts (2)	0.71
User-Perceived Quality (4)	0.93
Perceived User Project Monitoring (7)	0.73

Table 7. ANOVA Results (Collocated vs. Virtual teams)

Construct	Mean (Collocated team)	Mean (Virtual team)	*F*-value (significance)
Completeness and Adherence of Project Artifacts	4.92	4.56	0.551(0.467)
Consistency of Project Artifacts	6.32	6.51	1.025(0.323)
User-Perceived Quality	4.91	4.75	0.616(0.435)

Table 8. ANOVA results (tight vs. loose project monitoring)

Construct	Mean (Control group-imposed tight user project monitoring)	Mean (Experimental group—voluntary loose user project monitoring)	*F*-value (significance)
Completeness and Adherence of Project Artifacts	**5.60**	**4.50**	**4.6(0.044)**
Consistency of Project Artifacts	6.39	6.51	0.314(0.582)
User-Perceived Quality	4.61	4.75	0.076(0.785)
Perceived User Project Monitoring	**5.18**	**4.35**	**37.2 (0.000)**

Table 9. ANOVA results (perceived user project monitoring)

Construct	Mean (Perceived HIGH user project monitoring)	Mean (Perceived LOW user project monitoring)	*F*-value (significance)
Completeness and Adherence of Project Artifacts	**5.30**	**4.73**	**6.18(0.044)**
Consistency of Project Artifacts	6.41	6.49	0.107(0.768)
User-Perceived Quality	**5.01**	**4.17**	**8.91(0.003)**

Table 10. Pair-wise correlations between input and output variables

Construct	Quality of Projects		
	***Completeness and Adherence of Project Artifacts* (p)**	***Consistency of Project Artifacts***	***User-Perceived Quality***
Perceived User Project Monitoring	**0.215 (0.021)**	0.042(0.643)	**0.281(0.002)**

Table 11. Summary of results of teams engaged in software requirements analysis

	Collocated Teams vs. Virtual Teams in Off-Shore Mode	User Project Monitoring of Off-Shored Projects in Virtual Team Mode	
		Control/ Experimental	Perceived
Completeness and Adherence of Project Artifacts	-	TPM > LPM	HUPM > LUPM
Consistency of Project Artifacts	-	-	-
User-Perceived Quality	-	-	HUPM > LUPM

Note. TPM = tight project monitoring; LPM = loose project monitoring;

HUPM = high user project monitoring; LUPM = low user project monitoring

Mean values of the perceived monitoring of the virtual team were then computed. We categorized those responses that were above the mean value as *high perceived user project monitoring* and those that were below as *low perceived project monitoring.* The performance measured on all the three dimensions were then compared across these two sets, using a one-way ANOVA test. The results as presented in Table 9 indicate that artifacts produced by developers who perceived higher levels of user project monitoring practices were better on the two dimensions of completeness and adequacy, as well as user-perceived quality, as compared to those who perceived low user monitoring.

A pair-wise correlation was carried out between perceived project monitoring and the three measures of project quality, which further confirmed these findings. (These correlations in presented in Table 10.)

It is important to understand the difference between imposed project monitoring as defined in the control and experimental groups and perceived project monitoring. Though ANOVA results in Table 8 indicate that the mean values of perceived project monitoring of the control group were significantly higher compared to that of the experimental group, the mean of the experimental group was significantly higher (4.35) in the Likert scale. We also observed that in the

experimental group, some of the MU teams, along with their corresponding MDI *B* teams, had voluntarily adopted closer project monitoring practices. These MDI *B* teams had been submitting their project plans and intermediate artifacts to their MU user teams, thus resulting in higher levels of perceived project monitoring. From an experimental perspective, there was a positive impact of both imposed project monitoring as well as perceived project monitoring on adherence of artifacts. At the same time, there was a positive impact of perceived project monitoring on user-perceived quality, possibly because of the close working relationship adopted by the users and developers. This could have occurred through informal behavioral control mechanisms such as clan control deployed by the MDI *B* teams and their corresponding MU user teams. However this issue warrants further analysis. Table 11 gives a summary of the results.

CONCLUSION

In this article we have described an exploratory study that examines two aspects of virtual teams in off-shored software development projects, specifically in the requirements analysis phase. First, we examine whether the quality of projects produced by virtual teams engaged in pure off-shore mode is at par with that of traditional, collocated teams. Secondly, within the ambit of virtual teams, we examine whether user monitoring of the projects has an impact on the quality of projects.

Contributions of the Study

Our study is one of the few to apply social presence, media richness and control theories to develop and test a research model of the antecedents of quality of software requirements analysis projects conducted in off-shore virtual team environment. As client and vendor organizations are increasingly considering off-shoring parts of requirements analysis phases, our early conclusions might enable organizations to design communications and governance structures that might facilitate virtual requirements analysis. Considering the rapid leaps in technological infrastructure globally, technology will become a moot point in this facilitation. From an academic perspective, the introduction of these two theories in an offshore context lays the foundations for extended empirical research.

We find that there is no significant difference in the quality of projects produced by virtual teams that used lean media and that by collocated teams that used rich face-to-face communications. This is similar to findings reported in Burke and Chidambaram (1999) where, despite the persistently lower social presence of leaner media, distributed groups performed better than face-to-face counterparts. Possibly, a more task-focused approach and limited social interaction may have enabled teams to generate higher quality outputs. This could be a potentially important result because it implies that off-shoring, which was so far restricted to the lower level phases of system development (such as low-level design, coding, and testing) could successfully be extended to the requirements analysis phase as well. A key benefit, of course, is that software firms could save significantly on costs by locating their business and systems analysts in off-shore locations and facilitating interactions with users through virtual channels. While this may currently be challenging, our study highlights the need for future research in improving these virtual interactions between users and off-shored development teams.

The effect of user project monitoring (control/experimental) on the quality of off-shored requirements analysis projects is ambiguous. Formal behavioral and outcome control implemented through the experimental set up had a positive effect on one measure of quality. It did not have any effect on the other two measures. Piccoli and Ives (2003) pointed out that behavior control mechanisms, which are typically used in

traditional teams, have a significantly negative impact on trust in virtual teams. It was reported that behavior control mechanisms increase vigilance and create instances in which individuals perceive team members failing to uphold their obligations. On the other hand, the perceived user project monitoring had significant positive effect on two dimensions of quality (one assessed and one perceived).

We also infer that, even when project management practices were not enforced, teams might have adopted these practices to improve their performance through clan control. This observation, though anecdotal based on class observations and our analysis of perceived user project monitoring, has important implications. It provides clues that, apart from forced formal controls, informal controls existed between the users and developers when they share common goals (Choudhury & Sabherwal, 2003).

Our findings have important implications for the industry as well. Companies engaged in off-shore software development have produced strong processes around their global delivery model. However, whether the same process and project monitoring discipline will lead to success of projects conducted in pure off-shore mode in virtual team setting during the early stages of system development work has not been explored. Our research indicates that teams engaged in virtual teamwork might develop their own informal control mechanisms and even bypass the forced control mechanisms necessitated by the standard operating procedures while doing their projects. The firms (viz. both the clients and software developers) engaged in off-shore work should develop a conducive climate for team members to develop these informal controls that seem to affect project quality. Apart from this, our study fills the gap in the literature in the area of analysis of quality of projects implemented by virtual teams engaged in off-shore system requirements analysis. Further research is needed to confirm our exploratory findings.

Limitations of the Study: Opportunities for Future Research

Use of Experiments

Literature in the area of virtual teams has mainly followed three research methodologies—case studies, industry surveys, and experiments. Experimental methods make possible the careful observation and precise manipulation of independent variables, allowing for greater certainty with respect to cause and effect, while holding constant other variables that would normally be associated with it in field settings (Damian et al., 2000). They also encourage the investigator to try out novel conditions and strategies in a safe and exploratory environment before implementing them in the real world (McGrath, 1984). The industry is yet to adopt off-shoring of the requirements analysis phase. This precludes the use of case study or industry survey for this research. Hence, we used experiments where we can explore this emerging phenomena.

In our experiment, MDI *A* teams played the roles of both users (in Experiment 1) and developers (in Experiment 2). The dual roles could have created conflicts that might have affected (positively or negatively) their project quality. The same is true with MDI *B* teams, who performed the roles of consultants for both MU teams as well as MDI *A* teams. MU teams also had to manage two projects: one with tight monitoring (with MDI *A* teams) and the other with loose monitoring (with MDI *B* teams). To remove the confounding effects of dual roles played by the teams, it is recommended that a true controlled factorial experiment be conducted to verify our findings.

Use of Students as Surrogates

There are criticisms for the use of students in academic experiments as surrogates. However, MBA students have been used as surrogate us-

ers in a range experiments conducted (see, e.g., Briggs, Balthazard, & Dennis, 1996; Hazari, 2005). Even in requirement negotiation phase, students with work experience were taken as users for developing a small system (Damian et al., 2000). Remus (1986) argued that graduate students could be used as surrogates for managers in experiments on business decision making. Students often represent a typical working professional and organizational member due to the variety of backgrounds and goals (Dipboye & Flanagan, 1979). Studies in industrial organization psychology and organization behavior have found that results obtained from students were similar to those from managers (see, e.g., Locke, 1986). Despite the fact that users and developers in our experiments had 2 to 4 years of work experience, limitations of using students as surrogates are still applicable in our study. As the industry evolves, we suggest the extension of these experiments to business settings.

Complexity of Projects

Requirements analysis is intensive, and hence it is not possible to completely replicate in student experiments. However, our objective was to study the research questions on comparable, relatively well-defined small projects in which complexity of requirements analysis is not high.

Though the experiments were carefully designed, the projects were limited in scope and size compared to large-scale industrial projects. Furthermore, no formal measures of complexity were used in the study so that we could compare the projects used in the experiments with real-world industrial projects. Further research is needed to assess the impact of these findings on large-scale industrial projects with complex requirements.

Future Research Directions

One way of dealing with the lack of realism in laboratory experiments is to use multiple methods (McGrath, 1984) so that strengths of some compensate weaknesses of others. To truly test the predictive ability of the research results, the studies must also involve a multiplicity of research methodologies in order to avoid biases due to the methods used (Jarvenpaa, Knoll, & Leidner, 1988). Simulated laboratory negotiations could be complemented by field studies or validations (whose strength is realism), if the lack of realism is an issue. In our research, internal validity of results was established through conducting experiments in a controlled environment. We expect to conduct external validity through industry survey.

Finally, while we have explored one variable of project control, quality of projects can be affected by other variables such as team motivation, trust, cohesion, coordination, and communication (Chidambaram, 1996; Jarvenpaa et al., 1998; Lurey & Raisinghani, 2001). Hence, a comprehensive model that defines all factors affecting the quality of off-shored software requirements analysis projects must be developed. Further research is required to determine how informal controls develop between the virtual team members. One cause may be the amount of initial online socialization, when the teams familiarize with each other before the start of the project, for the design of such experiments in the future. Since it may not always be feasible to make experimental and control groups adhere to experimental requirements in a classroom setting, a flexible approach is needed in experimental design.

REFERENCES

Andres, P. (2002). A comparison of face-to-face and virtual software development teams. *Team Performance Management, 8*(1/2), 39-48.

Archer, N. P. (1990). A comparison of computer conferences with face-to-face meetings for small group business decisions. *Behavior & Information Technology, 9*(4), 307-317.

Battin, R., Crocker, R., Kreidler, J., & Subramanian, K. (2001). Leveraging Resources in global software development. *IEEE Software, 18*(2), 70-77.

Briggs, R. O., Balthazard, P. A., & Dennis, A. R. (1996). Graduate business students as surrogates in the evaluation of technology. *Journal of End User Computing, 8*(4), 11-17.

Burke, K., & Aytes, K. (1998). A longitudinal analysis of the effects of media richness on cohesion development and process satisfaction in computer-supported workgroups. In *Proceedings of the 31st Hawaii International Conference on Systems Sciences* (pp. 135-144).

Burke, K., & Chidambaram, L. (1996). How much bandwidth is enough? A longitudinal examination of media characteristics and media outcomes. *MIS Quarterly, 23*(4), 557-580.

Campbell, D. T., & Stanley, J. C. (1966). *Experimental and quasi-experimental designs for research*. Chicago: Rand McNally.

Carmel, E. (2006). Building your information systems from the other side of the world: How Infosys manages time zone differences. *MIS Quarterly Executive, 5*(1), 43-53.

Chidambaram, L. (1996). Relational development in computer-supported groups. *MIS Quarterly, 20*(2), 143-163.

Chidambaram, L., & Bostrom, R. (1993). Evolution of group performance over time: A repeated measures study of GDSS effects. *Journal of Organizational Computing, 3*(4), 443-469.

Choudhury, V., & Sabherwal, R. (2003). Portfolios of control in outsourced software development projects. *Information Systems Research, 14*(3), 291-314.

Crisp, C. B. (2003). Control enactment in global virtual teams. *Dissertation Abstracts International*. (UMI No.)

Damian, D. E, Eberlein, A., Shaw, M. L. G., & Gaines, B. R. (2000). Using different communication media in requirements negotiation. *IEEE Software, 17*(3), 28-36.

Damian, D. E., & Zowghi, D. (2003). An insight into interplay between culture, conflict and distance in globally distributed requirement negotiations. In *Proceedings of the 36th Hawaii International Conference on System Sciences.*

Dipboye, R. L., & Flanagan, M. F. (1979). Research setting in industrial and organization psychology: Are findings in the field more generalizable than in laboratory. *American Psychologist, 34*(2), 141-150.

Ebert, C., & De Neve, P. (2001). Surviving global software development. *IEEE Software, 18*(2), 62-69.

Edwards, K., & Sridhar, V. (2005). Analysis of software requirements engineering exercises in a global virtual team setup. *Journal of Global Information Management, 13*(2), 21-41.

Favela, J., & Pena-Mora, F. (2001). An experience in collaborative software engineering education. *IEEE Software, 18*(2), 47-53.

Foster, S., & Franz, C. (1999). User involvement in information systems development: A com-

parison of analyst and user perceptions of system acceptance. *Journal of Engineering Technology Management, 16*(3-4), 329-348.

Hartwick, J., & Barki, H. (1994). Explaining the role of user participation in information system use. *Management Science, 40*(4), 440-465.

Hazari, S. I. (2005). Perceptions of end-users on the requirements in personal firewall software: An exploratory study. *Journal of Organizational and End User Computing, 17*(3), 47-65.

Hoffer, J., George, J., & Valacich, J. (1999). *Modern systems analysis and design*. Reading, MA: Addison Wesley.

Jarvenpaa, S., Knoll, K., & Leidner, D. (1998). Is anybody out there? Antecedents of trust in global virtual teams. *Journal of Management Information Systems, 14*(4), 29-64.

Jarvenpaa, S., & Leidner, D. (1999). Communication and trust in global virtual teams. *Organization Science, 10*(6), 791-815.

Kircsh, L. J. (1996). The management of complex tasks in organizations: Controlling the systems development process. *Organizational Science, 7*(1), 1-21.

Kircsh, L., Sambamurthy, V., Ko, D., & Purvis, R. (2002). Controlling information systems development projects: The view from the client. *Management Science, 48*(4), 484-498.

Lin, W., & Shao, B. (2000). A relationship between user participation and system success: A simultaneous contingency approach. *Information & Management, 37*(6), 283-295.

Lind, M. (1999). The gender impact of temporary virtual work groups. *IEEE Transactions on Professional Communication, 42*(4), 276-285.

Locke, E. A. (1986). *Generalizing from laboratory to field setting: Research finding from industrial organization, organization behavior, and human resource management*. Lexington, MA: Lexington Books.

Lurey, J., & Raisinhgani, M. (2001). An Empirical study of best practices in virtual teams. *Information & Management, 38*(8), 523-544.

McDonough, E., Kahn, K., & Barczak, G. (2001). An investigation of the use of global, virtual, and collocated new product development teams. *The Journal of Product Innovation Management, 18*(2), 110-120.

McGrath, J. (1984). *Groups: Interaction and performanc*e. Upper Saddle River, NJ: Prentice Hall.

National Association of Software and Service Companies. (2005). *Indian IT industry*. Retrieved March 3, 2005, from http://www.nasscom.org/

Piccoli, G., & Ives, B. (2003). Trust and the unintended effects of behavior control in virtual teams. *MIS Quarterly, 27*(3), 365-395.

Powell, A., Piccoli, G., & Ives, B. (2004). Virtual teams: A review of current literature and directions for future research. *The DATABASE for Advances in Information Systems, 35*(1), 6-36.

Remus, W. E. (1986). An empirical test of the use of graduate students as surrogates for managers in experiments on business decision making. *Journal of Business Research, 14*(1), 19-25.

Rice, R. E., & Love, G. (1987). Electronic emotion: Socio-emotional content in a computer-mediated communication network. *Communication Research,* 14(1), 85-108.

Sharda, R., Barr, S. H., & McDonnell, J. C. (1988). Decision support system effectiveness: A review and an empirical test. *Management Science, 34*(2), 139-157.

Schimdt, J. B., Montoya-Weiss, M. M., & Massey, A. P. (2001). New product development decision-making effectiveness: Comparing individuals,

face-to-face teams and virtual teams. *Decisions Sciences*, *32*(4), 575-600.

Sarkar, S., & Sahay, S. (2002). Information systems development by US-Norwegian virtual teams: Implications of time and space. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences* (pp. 1-10).

Short, J., Williams, E., & Christie, B. (1976). *The social psychology of telecommunications*. London: Wiley.

Sridhar, V., Nath, D., & Malik, A. (in press). Analysis of user involvement and participation on the quality of IS planning projects: An exploratory study. *Journal of Organizational and End User Computing.*

Stevenson, W., & McGrath, E. W. (2004). Differences between on-site and off-site teams: Manager perceptions. *Team Performance Management*, *10*(5/6), 127-132.

Townsend, A., DeMarie, A. M., & Hendrickson, A. R. (1998). Virtual teams: Technology and the workplace of the future. *Academy of Management Executive*, *12*(3), 17-29.

Walther, J. (1995). Relational aspects of computer-mediated communication: Experimental observations over time. *Organization Science*, *6*(2), 186-203.

This work was previously published in Journal of Global Information Management, Vol. 16, Issue 4, edited by F. Tan, pp. 24-45, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.21
A Case Study on the Selection and Evaluation of Software for an Internet Organisation

Pieter van Staaden
Media24 Ltd., South Africa

ABSTRACT

The author conducted research to determine whether IT managers, IT auditors, users, management, etc. (all decision-makers) use a certain evaluation and selection process to acquire software to meet business objectives and the requirement of users. An argument was used that the more thorough the software evaluation and selection process, the more likely it would be that the organisation will chose software that meets these targets. The main objective of the research was therefore to determine whether Media24 uses evaluation methods and obtains the desired results. The results confirmed that Media24 uses suggested protocol as noted in the theory for software acquisition correctly during most stages.

INTRODUCTION

There is a wide variety of methods that can be used for selection of software in various fields of business (e.g., manufacturing, service providers, insurance, wholesale, retail, etc.). This software is used for a variety of purposes in businesses. However, selecting the software that meets organisational requirements and business objectives could prove to be a challenge considering the number of vendors and software available.

Choosing the right software for your company can be bewildering. There are thousands of titles to choose from, and programs and their functionality differ frequently. (Buyerzone.com, 2002)

A hurried, uneducated choice could lead to various problems in the company. Some of these are

failing to support an important business process, supporting a process inaccurately or inefficiently, unhappy customers, disgruntled employees, loss of sales, and poor financial performance.

Competition in the Western Cape requires good performance in all aspects of the electronic publication industry. Bad judgments or decisions in terms of software acquisition could cause a company some losses and complications in their daily operations. Choosing the right software is therefore important and can be achieved by using pre-determined evaluation and selection guidelines.

EVALUATION AND SELECTION OF A COMMERCIAL SOFTWARE SYSTEM

Decisions Made Prior to the Software Evaluation Process

As mentioned by Capterra's software selection methodology (2002), certain procedures should be completed before the actual evaluation is conducted. They suggest that the company should start off by interviewing some staff members, addressing corporate vision, analysing existing systems limitations and features, and looking at present policies and procedures. The company should also determine whether new software will help the business and if it will increase competitive advantage.

They argue that when the decision is made to purchase software, a project plan should be developed to evaluate and list the evaluation criteria that will be used during the process. A project team should also be selected to carry out the evaluation. This team must include representatives from all levels the organisation. If the proposed software incorporates financial aspects, the audit team should also be included.

Determine Requirements for the New Software Package

The purpose would be to create a comprehensive and prioritised list of requirements to help evaluate the software. Base Consulting Group (BCG) (2000) state that the requirements definition should consist of several processes (such as managerial requirements (budget/timing, reporting requirements), functional requirements (stated business needs, technical requirements), and IS standards (data flow diagrams, system interfaces, and hardware and network requirements with emphasis on capacity).

They also note that some companies do not develop detailed requirements and as a result may be dissatisfied with the final outcome. Romney and Steinbart (2000) support this statement and suggest that one or any combination of four strategies (listed below) should be used to determine requirements for the new software:

- Survey end-users to determine what their requirements for software is by using questionnaires, personal interviews, and focus group discussions.
- Analyse the existing system and eliminating requirements that have already been defined.
- Examine how the existing software is used, helping to determine the shortcomings of the system, and identifing any new requirements needed by users.
- Pilot demonstrations of applications/software systems could be utilised when there is a problem in identifying requirements.

Document the Requirements

The systems requirement document or software requirement specifications should be the starting point for measuring performance of the final

Table 1. Capterra's methodology (2002)

COLUMN NAME	DESCRIPTION
Functional department, business processes, and process.	This creates the requirements hierarchy and ensures that all processes are covered (e.g., would be creditors department, cheque printing, approving cheques for printing, etc.)
Requirement type	This identifies the requirement as functional, technical, vendor related, or contractual.
Requirement description	This details the requirement itself and should be as descriptive as possible.
Priority and ranking	*This could be used during the evaluation.*
Objective addressed	This could be used to match the requirement to a business objective.
Comments	This can be used for any additional comments or justifications.

system (Shelly, Cashman, & Rosenblatt, 1998). Users must understand the document to be able to improve the final version. The content of this requirements document will depend on the type of company and the complexity of the new system. BCG (2000) states that the requirements document is the cornerstone to evaluate the software and should be used to identify requirements. Capterra (2002) argues that there is a methodological approach available to help with requirement analysis. This is listed in Table 1.

Selecting Vendors

Michell and Fitzgerald (1997) argue that the range of services offered by IT vendors is large and growing rapidly. They also note that while searching for the "best" vendor, it should be borne in mind that the process of selection and evaluation of a vendor is important. Base Consulting Group (2000) suggests that the project team's first step should be to identify the vendors who offer software solutions that could be used. It could be a high-risk approach not to properly evaluate vendor companies. The sources used to compile the list of vendors should be recent and reliable. These sources include software user groups, databases, industry groups, research firms, consulting firms, trade shows, seminars and conferences, current users, personal recommendations and contacts, competitors, IT, and business magazines, as well as Web sites.

PRELIMINARY EVALUATIO OF VENDOR COMPANIES AND THEIR PRODUCT

Ward (2001) argues that selecting software from a vendor should be a simplified process. She suggests that vendors be invited to participate in a software demonstration because it reduces time

spent on evaluations. Companies could work toward solving the business problems earlier resulting in faster return on investment. Base Consulting Group (2000) states that inviting too many vendors to participate increases the costs and timelines of the project. Team members may also lose focus after seeing too many product demonstrations. In order to shorten the time of the process send the request for proposal (RFP) to a shortlist of 5 vendors, and do a preliminary evaluation of the vendors.

The evaluation team should look at things like:

- The standard functionality and key features of the product
- Technology requirements (hardware, additional software, database, operating system, network, development tools)
- Product considerations such as viability, stability, and cost
- Products targeting different, smaller, or much larger companies or industries should be eliminated
- Eliminate products that are in development or a recent release.
- Licensing and support costs are examined and products that are over/under priced should be eliminated (licensing escalation must be considered).
- Develop a request for proposal (RFP).

According to Levinson (2001), an RFP guides buyers through a process of tying business needs to technical requirements (e.g., as the particular platform on which the software needs to run on or the systems with which the solution must interface). It clarifies why they are undertaking a particular project. Schwalbe (2000) suggests that the RFP should include: statement of the purpose, background information on the company issuing the RFP, basic requirements for the products and/or services being proposed, HW and SW environment, description of the RFP process, statement of work (SOW), and other information added (as appendices). The SOW should describe the work required for the procurement of the software and help vendors determines if they can deliver required goods and services.

EVALUATION PREPARATIONS

Gather and Organise Resources

Lars and Matthew (2002) note that a reason for not detecting errors early is because the inadequacy of the test used by the team. The quality assurance of the evaluation project is jeopardised. To prevent this, the test team should ensure they have the resources to detect errors present. The adequacy of resources gathered should be determined at the same time potential vendors are identified. Resources could be added or updated to support the evaluation.

Determine the Evaluation Approach/ Technique

Restrictions on Evaluating Software

Dean and Vigder (2002) state that, while purchasing software, there are some unique constraints on the ability to conduct effective testing. In general, it should be assumed that there is no access to the source code. If the source code is available it could not be modifiable and it means that the executable part cannot be tested internally and this rules out white box testing. Documentation should consist of user manuals and advertising materials and is not directed at evaluating the software (e.g., it does not describe the behaviour of the software in response to abnormal input).

Evaluation Techniques and Methods

Romney et al. (2000) suggest benchmarking while processing times of software are measured. Software with the lowest time is normally judged the most efficient. Oberndorf, Brownsword, Morris, and Sledge (1997) engaged scenario-based testing methods to represent typical procedures for the software to be programmed and not the software undergoing tests. Test procedures are developed based on scenarios and each is evaluated against a set of criteria. In this case, the initial scenarios are established using preliminary operational definitions. The results of this will serve as confirmation that the software performed satisfactory against set parameters.

Romney et al. (2000) suggest a point scoring technique to evaluate the vendor. Each criterion is assigned a weight based on its relevancy. The vendor is assigned a score based on how their proposal measures up to each criterion. The vendor with the highest score is then judged the "best." They argue that "requirements" costing is an alternative where the total cost of the proposed software is calculated. This provides an equitable basis for comparison.

Another method suggested by Voas, Charron, and McGraw (1997) is the use of fault injection techniques. This is effective when buyers do not have access to the source code. The method consists of inserting erroneous values into the control stream and checking the results. This technique is an example of evaluating (for discovery) to determine unknown or unexpected reactions of the product under evaluation.

Beizer (1995) suggests *black box testing* to allow a tester to treat each module as a unit that can be defined by its inputs and outputs (the interfaces to the module) without considering the route by which an input is transformed into a particular output. Visibility into the internal workings of the code module is not necessary and source code not required. An example of black box testing is boundary value analysis where inputs are supplied to the software to be tested (these values represent valid, invalid and parameters). The outputs are measured and accepted if they fall within expected limitations. This type of testing is used during acceptance testing and is the basis of validation testing, confirming the software performed the required functions.

Other techniques (Hausen & Welzel, 1993) include analysing product documentation, presentations, using trial versions, scheduling demonstrations, or attending training of the software. They suggest that one or more of the previously mentioned techniques could be used to supplement the evaluation of software. The project team should use discretion when selecting evaluation techniques as a company's approach and resources may vary.

Evaluation Considerations

Hausen and Welzel (1993) mention that some of the following principles should be taken into consideration:

- Repeat testing of the same product using the same product specifications with the same testing techniques must deliver similar results (Repeatability).
- Repeat evaluation of the same product to the same product specifications by different parties must deliver similar results (Reproducibility).
- The evaluation is free from bias while achieving any particular result (Impartiality).
- The result is obtained with minimum subjective judgment (Objectivity).

Product Evaluation

Hausen and Welzel (1993) state that the evaluation process should consider software features (compared to the requirements document), product

information (acquired from the RFP, product demonstration, information gathered from investigating vendors, etc.), evaluation techniques, and process information (e.g., results obtained from the testing techniques).

Capterra's methodology (2002) states that all software should be evaluated to determine if it meets requirements (functional and technical). Any additional (functional/technical) requirements should be listed and re-calculated. Missing requirements should be listed and cost incurred to add these features should be calculated. Price and maintenance levels of the product have to be evaluated by totalling cost and maintenance levels. Firms must consider initial product costs (also long-term costs (such as training, implementation costs, maintenance, and upgrading costs). Project teams should keep in mind that software is expensive and by picking the wrong one could have costly repercussions.

Final Evaluation of Vendor Companies Providing Possible Software Solution

Pollard (1999) suggested the evaluation of the support and maintenance staff of the vendor. He notes that it is necessary to know the number of people in customer support. The response time can be measured by calling the customer support department. The availability and quality of the implementation support also ought to be evaluated. The new software could have bugs and other problems (e.g., not meeting the required deadline). The vendor must provide training because users want to use the system properly.

He suggests following up on customer references, reviewing case studies and finding out how many companies are using the software. The financial stability of a vendor is an aspect to consider. Pollard (1999) supports this by suggesting the examination of the financial history and the long-term financial stability of the vendor.

Selecting the Software System

The total score of the software should be recorded on a scoring sheet when a point scoring technique is used. All the software must be listed from the highest to the lowest. The software with the highest score would represent the best fit for the organisation. Although the software with the highest score might represent the best solution for the company, there may be reasons unrelated to the requirements that could prevent an organisation from selecting software. Inconsistencies should be identified (also extremes in scoring that may influence it—or a competency or deficiency within a single business function). Criteria such as a business partnership, potential future business, or other intangibles, must also be included.

Notify the Vendor

Once the steering committee has approved the vendor, then the vendor should be notified and a contract drawn up. The diagram (on the next page) was derived from the theory to illustrate steps used during the evaluation and selection process.

PROBLEM STATEMENT

Brown and Wallnau (1996) state that organizations should recognize the importance of technology "refreshment":

- To improve the quality of their products and services
- To be competitive with organizations providing similar products and services
- To remain attractive to investors

Any organization should invest in appropriate software to stay in business. Careful decision-

making on investment into software is therefore essential. Whether the release of an update, or the availability of new software, should force an organization to initiate an evaluation process that provides timely, balanced information on which decisions can be made.

The problem statement was thus stated as:

…the more thorough the evaluation of software, the greater the chances could be for the organization to select the software that will meet their business objective and the requirements of the users.

RESEARCH METHODOLOGY

Objective

The objective of the study is to determine whether Media24 uses the correct software evaluation and selection guidelines when purchasing software from a vendor as prescribed in the theory and whether these guidelines have obtained results. Based on the theory and the problem statement, the research questions were stated as follows:

- Is the software evaluation process of Media24 thorough enough to select software that fulfils their end-user requirements?
- Does the organisation pick software that meets the business objectives easily?

Limitations of the Study

The researchers had decided to exclude legal procedures and only focussed on the project from a technical viewpoint. Also they were not allowed to use the name of any software used by Media24 as it might discredit the vendor or Media24. The research scope also does not include the processes (e.g., contract negotiation) followed after a decision has been made.

Development of Questionnaire

The researchers compiled a questionnaire consisting of two sections (Section A and B). Section A contained seven sub-sections, covering aspects of evaluation and selection processes. Section B contains the user satisfaction survey. The questionnaire was handed to a Media24 IT manager who reviewed it. He also identified people responsible for evaluating and selecting software in Media24 to be respondents while he completed a questionnaire himself.

Section A was sent out to 15 decision-makers involved in acquiring software in Media24. The objective was to measure whether they used the evaluation techniques as prescribed in the theory. Section B was sent to 50 users of the software. The objective was to determine whether they were satisfied with present software. All questions where derived from the theory described earlier. A 5-point Lickert scale was used in most of the questions (it included questions that required respondents to pick more than one answer). The reason is to evaluate areas where answers from the respondent indicates an in depth approach to the evaluation process. At the end of each section, a section was dedicated to find out in more detail what respondents think.

Analysis of Data

The results of both sections were compared using cross tabulation. The data was captured and analyzed on an excel spreadsheet. Twelve "section A" respondents returned the questionnaires and 74% of the respondents who received section B returned the questionnaire.

Section A: Evaluation Process

Nearly all the respondents agree that the steering committee correctly directs the evaluation pro-

Figure 1. Graphical summary of software selection

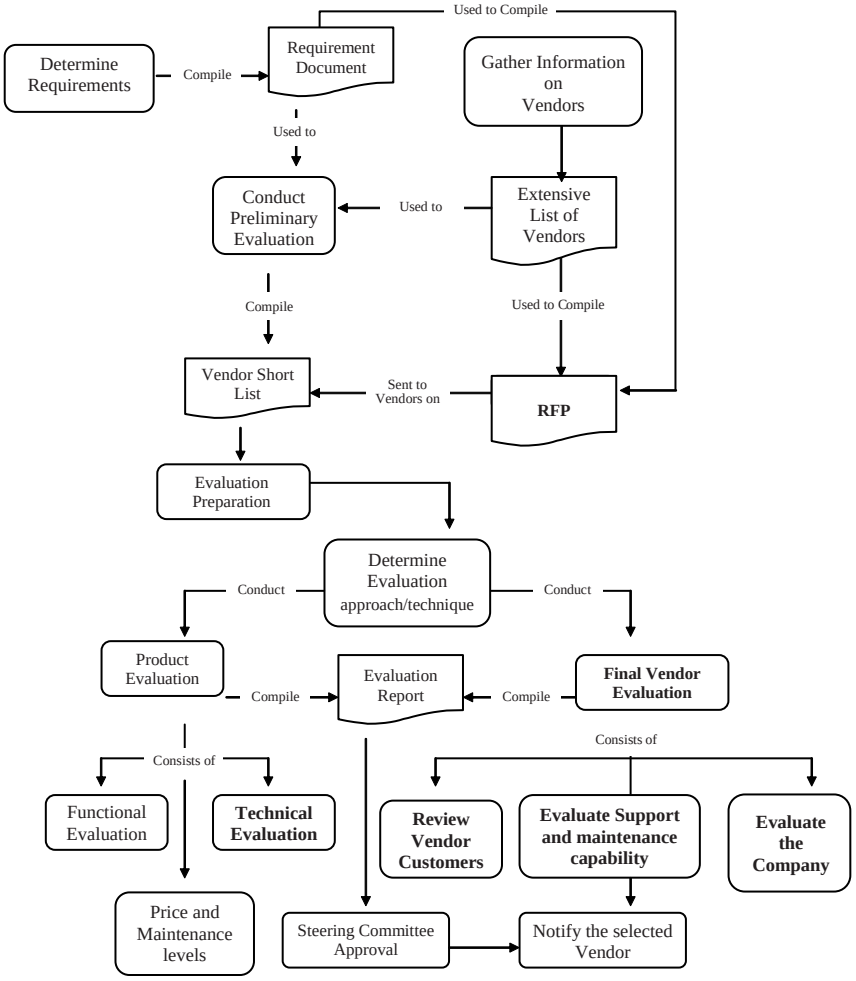


Table 2. Project management structure

	SA	A	NDA	D	SD
There is a Steering Committee in place to oversee and direct the evaluation process	11	7	1	0	0
Sufficient resources are allocated to the evaluation process	4	8	0	0	0
A project team with the required expertise is assigned to conduct the evaluation and selection of the new software system.	3	5	1	3	0
AVERAGE	4	6	1	1	0

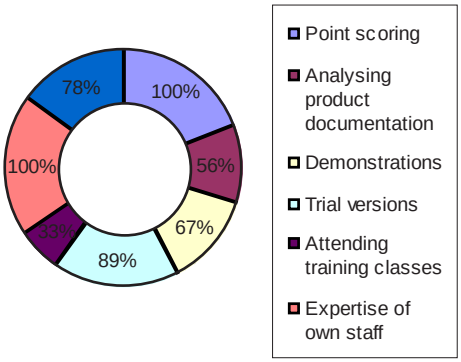
Table 3. Requirements definition

	SA	A	NDA	D	SD
Technical requirements are determined (e.g., the system interface)	5	7	0	0	0
The Functional requirements are determined (e.g., business objectives system has to fulfil)	5	5	2	0	0
Managerial requirements are determined (e.g., reporting capabilities, budget, timing)	4	8	0	0	0
Requirements are properly documented and easy to understand	2	7	1	2	0
AVERAGE	4	7	.5	.5	0

Table 4. Vendor identification and evaluation

	SA	A	NDA	D	SD
Various sources (e.g., Web sites) are investigated in order to identify the software available.	3	7	2	0	0
A preliminary evaluation is conducted to limit software that are going to be extensively evaluated.	3	7	0	1	1
The support and maintenance staff provided by the vendor is evaluated based on…					
…response time	4	7	1	0	0
…quality of support	1	9	2	0	0
…number of people	0	1	1	8	2
…cost	1		1	0	0
The company providing the software system is evaluated based on…					
...long term Financial stability	5	5	0	2	0
...customer References	4	7	1	0	0
...number of clients	4	5	3	4	0
...long and short term strategic planning	0	2	2	8	0
AVERAGE	2	7	1	.7	.3

Figure 2. Techniques used while evaluating software



cess (11). This indicates that people realise that there is a structure in place that could oversee the evaluation of software and is in agreement with a similar comment by Capterra (2002).

All the respondents agree that there are sufficient resources available to ensure that the evaluation process runs smoothly. Nine of the respondents indicated that the project team has enough expertise to conduct an evaluation. Three indicated that they disagree with this. There might be a problem that falls outside the scope of this study and needs to be addressed by management. The organisation therefore needs to assemble a project team that is representative of the people working in the organisation (see also Dean & Vigder, 2002). Nine of the respondents indicated that Media24 used the interview method to determine requirements. The same respondents have also used the present system to ensure that they meet the correct requirements (see Figure 4). Six respondents have indicated that they use questionnaires to collect requirements. It seems that project teams in Media24 prefer to use three methods to determine software requirements.

All respondents have indicated that technical requirements are determined beforehand (systems interface). Ten of the respondents are happy that business objectives have been met while determining functional requirements. All agree that managerial requirements should be met when requirements are defined. Nine of the respondents noted that requirements are properly documented. Media24 needs to address this to ensure that people are in agreement otherwise it can become a problem.

Ten of the respondents agree that many sources should be investigated to identify correct software. The same number agrees that Media24 should ensure that software that meets requirements are evaluated. This is done to limit the number of products to be considered. On the other hand, 11 of the respondents agree that the support/maintenance staff provided by the vendor is evaluated on response time while the quality of support is rated high by 10 of the respondents. It seems to the authors that 'old fashioned' values are still important while looking at new IT investments. This is in agreement with the statement made by Capterra (2002) that end-users value help provided by the supplier.

The number of people working for the vendor is not regarded as important by the respondents;

Table 5. Request for proposal

	SA	A	NDA	D	SD
A Request for proposal is sent to Vendors.	10	1	1	0	0
The RFP includes…					
…the purpose of the RFP	1	9	2	0	0
...all necessary background information of the company issuing the RFP	1	4	6	1	0
...the requirements for the new system	4	7	1	0	0
...the hardware and software environment currently being used	3	4	4	1	0
...instructions on how to reply and a description of how responses will be dealt with	9	2	1	0	0
...a statement of work (SOW) that includes the required work needed for the procurement of the new software solution.	1	9	2	0	0
AVERAGE	4	5	2.6	.4	0

as long as their service is not affected by it (quality is rated higher). Eight stated that cost plays a role while maintenance is evaluated. Most of the respondents noted that the vendor should be evaluated on financial stability and customer references. The number of clients and long-term strategy of the vendors were not used when looking how reputable the vendor is. This is something that Media24 and similar organisations need to investigate.

Table 5 displays the data collected on the RFP. Most of the respondents agree that this is complied with by Media24. Many (10) stated that the evaluation processes are properly documented while thought is given to objectivity (11) and impartiality (9). Most (10) of them agree that repeatability is lacking while reproducibility (7) is important and needs careful attention. Impartiality is an aspect that needs to be addressed as a third of the people noted that it is lacking. The averages for the request for proposal can be improved but because there are individual items that management of Media24 needs to pay attention to as stated. The common method used to evaluate the software is the point scoring technique (Figure 2). Trial versions of software have a better chance to be selected if it was analysed previously or used. The other method that Media24 uses to evaluate software is the expertise of their staff in a particular field.

Table 6 shows the evaluation process. The respondents were positive that the final score used to determine how well the software product meets the requirements is a good method. Nine respondents are in agreement that the documentation to do this evaluation is well laid out beforehand. Management needs to convince the three that does not agree to accept the documentation as presented. Objectivity is complied with but repeatability of the evaluation is not highly thought off (and the same for reproducibility). Nine of the respondents noted that the tests were

Table 6. Evaluation of the product

	SA	A	NDA	D	SD
The evaluation score of the product is determined based on how well it meets the pre-determined requirements.	1	11	0	0	0
The evaluation process and results are properly documented.	1	8	2	1	0
When conducting the evaluation thought, is given to…					
...Objectivity	7	4	1	0	0
...Repeatability	0	0	4	7	1
...Reproducibility	0	6	4	2	0
...Impartiality	7	2	2	1	0
AVERAGE	3	5	2	1.8	.2

impartial. Again, it would be a task for management to convince the remaining three that this is the case before problems are experienced (also supported by Shelly et al., 1998).

Most of the people agree that the final decision to acquire software is based upon the evaluation results. Eleven of the respondents agree that they are happy with the results achieved by Media24. Agreeing meant that they use the methods as prescribed in theory but there are some methods that are not used presently (e.g., benchmarking and black box testing). These should be investigated and used to ensure that the methods presently being used is still considered the best (Beizer, 1995).

Table 7. Results obtained from decision makers

	SA	A	NDA	D	SD	TOTAL
The final decision is not made solely based on the evaluation result, but also includes other intangible aspects (e.g., potential future business)	1	9	2	0	0	12
I am satisfied with the evaluation results obtained by our company.	4	7	0	1	0	12
AVERAGE	2.5	8	1	.5	0	12

User Satisfaction and Overall Effectiveness of Evaluation Process Used

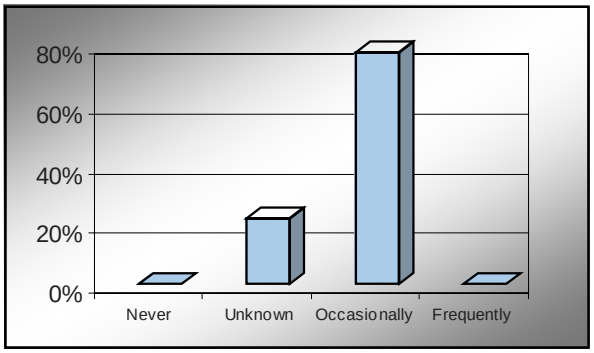
Nearly all the respondents (11) at Media24 are satisfied with the software that was purchased by Media24. Eight of the respondents have agreed that it falls within the parameters set by the organisation. The respondents that do not agree (4) may be users that were not part of the project team. They should be convinced that the software is of benefit to Media24. Eleven of the respondents agreed that the software meet their requirements. This is supported in their article by Michel and Fitzgerald (1997) who stated that normally most of the users who were part of the process are happy with the software.

This could mean that the individual user requirements agree with Media24's requirements. However, seven of the respondents agree that there is room for improvement This should be investigated in another study as this falls outside the scope

Table 8. User satisfaction

	SA	A	NDA	D	SD
I am satisfied with the software I am using.	2	9	0	1	0
The commercial software system fulfills the business objective it was assigned to.	1	7	3	1	0
The commercial software system adheres to my requirements.	2	8	0	1	0
There is no room for improvement for the commercial software system I am using.	0	4	2	5	1
AVERAGE	1.5	7	1.25	2	.25

Figure 3. Number of complaints from users



of this study. This does not agree with previous statements made by the respondents. Maybe the present evaluation procedure should be extended to inquire about reasons why the respondents argue that there is room for improvement. This should be included as part of the evaluation.

The respondents (9) complain about the software occasionally to ensure that management pay attention. Seven of the users argue that there is room for improvement. This would require another survey to find out why users complain about the software and if there is room for improvement.

Section B: Assessment

The main objective of section B was to assess whether the users where satisfied with the software acquired by Media24. There are some complaints lodged by the respondents (e.g., the systems response time and redundant processes and procedures included in the current system and that the software does not integrate with other software). The figure on the next page illustrates the percentage respondents who agreed that they use the correct evaluation techniques. The results are summarized according to key areas.

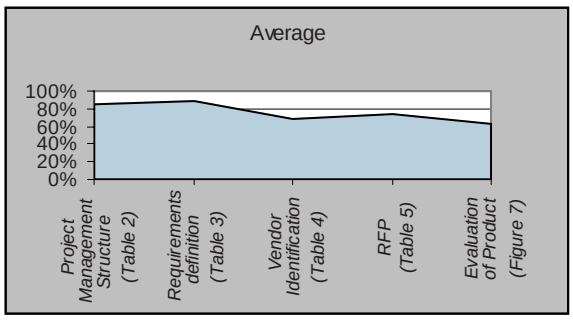
An area of concern should be the evaluation of the product for management (only 63% average with some areas that were identified that needs attention). Eighty-six percent of users agreed that they where satisfied with the software obtained. Sixty-five percent of users verified that the software systems meet the business objective and 84% noted that the software they are using meets their requirements (also supported by Voas et al., 1997).

DISCUSSION AND CONCLUSION

From the findings, the researchers conclude that Media24 has identified and used the better suited evaluation techniques as described in the theory. End-users were generally satisfied with the software and agreed that the software meet their requirements and business objectives. The above-mentioned statement answers the first research question. It is evident that software that help Media24 achieve its goal has a better chance of being selected (answer to the second research question).

The researchers note that there were some complaints mentioned by end-users. They feel that this "unhappiness" could be because not all employees were actively involved in determining the requirements or because the requirements were

Figure 4. Key area's average



not communicated to all users. A manager during an interview noted that the software he was using does not integrate with one of the sub-systems. This eventually leads to more work, as manual reconciliation has to be done between the sub-system and the software. This could be because the requirements or the software weren't described properly in the request for proposal. Most of the statements indicated that e-commerce organisations should be careful how they select software (Capterra, 2002; Hausen & Welzel, 1993; Lars & Methven, 2002). This was also supported by the research findings of this study.

FUTURE RESEARCH

Further research might be needed to refine or redesign the evaluation approach used by Media24. The reason is that the current software will be outdated soon and with the rise of new technology, the evaluation and selection process might have to be adjusted. There were also some issues that were not picked up before the installation of the product (e.g., integration with sub systems). This indicates why the evaluation and selection process might have to be revised. Other studies could help place emphasis on the use of specific evaluation models. In order to speed up the process and to gather more data more respondents will have to be included in the sample before the next survey is conducted.

REFERENCES

Base Consulting Group: Strategic Technology White Paper Series. (2000). *Software selection: A business-based methodology.* Retrieved August 2002, from http://www.baseconsulting.com/ assets/PDFs/BusinessBasedMethodology.pdf

Beizer, B. (1995). *Black box testing: Techniques for functional testing of software and systems.* New York: John Wiley & Sons Inc.

Buyerzone.com. (n.d.). Retrieved August 2002, from http://www.buyerzone.com/ software,internet_software/printable_bg.html

Brown, A.W., & Wallnau, K. C. (1996). *A framework for systematic evaluation of software technologies.* Software Engineering Institute Carnegie Mellon University. Pittsburgh. Scientific Literature Digital Library. Retrieved August 2002, from http://citeseer.nj.nec.com/cache/papers/cs/23040/ http:zSzzSzwebfuse.cqu.edu.auzSzInformationzSzResourceszSzReadingszSzpaperszSzsoftware.evaluation.pdf/brown96framework.pdf

Capterra detail-level software selection methodology. (2002). Retrieved August 2002, from http://www.capterra.com/detailed_software_selection_methodology.pdf

Dean, J. C., & Vigder, M. R. (2002). *COTS software evaluation techniques.* National Research Council Canada: Software Engineering Group. Retrieved August 2002, from http://seg.iit.nrc.ca/papers/NRC43625.pdf

Hausen, H. L., & Welzel, D. (1993). *Guides to software evaluation scientific literature digital library.* Retrieved August 2002, from http://citeseer.nj.nec.com/cache/papers/cs/12053/ftp:zSzzSzftp.gmd.dezSzGMDzSzSW-QualityzSzEval-Guide746.pdf/hausen93guides.pdf

Lars, M., & Matthew, G. (2002). *Ten points for improving the testing process: White paper.* Retrieved August 2002, from http://www.tautester.com/download/ten_points.pdf

Levinson, M. (2001, July). Vendor management: Do diligence. *CIO Magazine.* Retrieved August 2002, from http://www.cio.com/archive/070101/vet.html

Michell, V., & Fitzgerald, G. (1997). The IT outsourcing market place: Vendors and their selection. *Journal of Information Technology, 12*(3), 223-237.

Oberndorf, P., Brownsword, L., Morris, E., & Sledge, C. (1997). *Workshop on COTS-based systems.* Retrieved August 2002, from www.sei.cmu.edu/pub/documents/ 97.reports/pdf/97sr019.

Pollard, W. E. (1999, July). Confessions of a software salesman. *CIO Magazine*. Retrieved August 2002, from http://www.cio.com/archive/070199/expert.html

Romney, B. R., & Steinbart, P. J. (2000). *Accounting information systems* (8th ed., pp. 638-641). NJ: Prentice Hall.

Schwalbe, K. (2000). *Information technology project management.* PA: Course Technology. 311, accessed August 2002.

Shelly, G. B., Cashman, T. J., & Rosenblatt, H. J. (1998). *Systems analysis and design* (3rd ed.). Cambridge: Course Technology.

Voas, J., Charron, F., & McGraw, G. (1997). *Predicting how badly "good" software can behave.* Reliable Software Technologies Corporation. Scientific Literature Digital Library. Retrieved August 2002, from http://citeseer.nj.nec.com/cache/papers/cs/743/ftp:zSzzSzrstcorp.comzSzpubzSzpaperszSzieee-gem.pdf/voas97predicting.pdf

Ward, S. (2001, December). Keep it simple when buying enterprise apps. *CIO Magazine*. Retrieved August 2002, from http://www.cio.com/analyst/051101_hurwitz.html

This work was previously published in Managing Information Communication Technology Investments in Successful Enterprises, edited by S. Lubbe, pp. 190-208, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.22
Planning and Managing the Human Factors for the Adoption and Diffusion of Object-Oriented Software Development Processes

Magdy K. Serour
University of Technology, Sydney, Australia

ABSTRACT

Although there are a large number of contemporary software development processes/methodologies available to assist and guide software professionals in developing software systems, there is no specific process that can assist organizations in planning and managing their transition to this new work environment. As a result, there are still a large number of information technology (IT) organizations that have not yet implemented any object-oriented (OO) process. For them, the transition to a new work environment and the adoption and utilization of a software process implies a number of problems, commonly including necessary human and organizational resistance to the ensuing cultural change. This chapter provides IT organizations and professionals with insights into the most important key success factors that may promote the entire process of organizational change. We investigate the effect of various human factors on the adoption and diffusion of an object-oriented software development process. Some of the human factors include motivation, leadership, resistance to culture change, and willingness and readiness to change. In addition, this chapter explores the significant role of these factors in controlling the entire process of implementing an OO process in practice, emphasizing the significance of planning and managing these "soft" factors to achieve clear advantages and gain enviable results.

INTRODUCTION

This chapter investigates and examines the effect of various human behavioral patterns during the organizational transition to an object-oriented (OO) work environment, and the adoption and diffusion of an OO software development process. Technology is only a tool; what makes the difference is the individual who makes use of the technology, and the culture that motivates people to realize and understand the advantages of adopting such technology (Zakaria & Yusof, 2001).

During any paradigm shift, human tendencies play a critical role that may invariably result in either success or failure. Examples of such human aspects may include cultural change coupled with people's resistance, motivation, education and training, communications, and leadership. Collectively, these factors can form either opposing or supporting forces that may influence and impact on the entire transition process. Therefore, human aspects must be seriously considered, well addressed, planned, and managed for a rewarding result.

Past studies (e.g., Gibson, 1999; Ioannidis & Gopalakrishnan, 1999; Nambisan & Wang, 1999; Auer & Dobler, 2000; Jurison, 2000; Burshy, 2001) of the process of organizational transition have related the transition process to how organizations adopted innovation, ideas, new technologies (e.g., Web services and e-business), or new "ways of doing things" (e.g., the adoption and deployment of an OO process).

What these processes missed in the past was the first (and the most critical) step towards the adoption of a new technology. They all missed the study of moving organizations from their current state or environment to their desired one where they can feel comfortable, familiar, and confident to adopt and diffuse an innovation or new technologies such as OO processes. Getting organizations ready to adopt and diffuse a new technology involves a number of serious managerial decisions that must be made to provide full management support, dedication, and commitment. Organizations must feel comfortable and familiar with the new way of "doing things" before any attempt is made to implement these new ways in practice to avoid or lessen people's natural resistance to change, and also increase their acceptance and readiness.

Hence, the main objective of investigating the impact of human issues is to gain a full understanding of individual behavior during the transition and also to examine different human factors that influence the response of individuals within organizations toward the adoption of an OO software development process.

ORGANIZATIONAL CHANGE AND HUMAN FACTORS

"The greatest difficulty in the world is not for people to accept new ideas, but to make them forget about old ideas." John Maynard Keynes

People Behavior During Organizational Change

During an organizational transition, different people play different roles, such as motivators, adopters, resistors, opposers, and neutral or observers (Bridges, 1995). How they respond to change during transition can, and in most cases does, dominate and determine the success or failure of the entire project. The inextricable reality is that people are different, and so act and react to changes differently; from time-to-time even the same person can behave in a different manner.

Bridges (1995) claims that changes are always accompanied by natural resistance, as changes often drive people out of their comfort zone. Consequently, people can develop a resistance to change and become the main obstacle to the whole organizational change.

Once an organization comes to realize what it needs to achieve and decides how it will accomplish its goals, the main challenge becomes the issue of effective and efficient management of human factors. It is quite surprising to know that 80% of project failures are traced back to mismanagement of human factors (Jacobson, Ericsson, & Jacobson, 1995).

Unfortunately, there are many organizations still struggling to deal with difficulties related to the effective management of human or sociological factors during technology adoption and diffusion. This type of problem is usually caused by management's lack of commitment to the human factors of IT. Szewczak and Khosrow-Pour (1996) relate this problem of mismanagement of human aspects to the fact that, in general, organizations traditionally invest a significant proportion of their resources to obtain the necessary hardware and software technologies, but with insignificant investment in the human aspect of technology. An experienced software development team that has been around for more than a decade, for example, is likely to have superior expertise in (and consequently be comfortable with) traditional software modeling techniques such as Flow Charts, Data Flow, and Entity Relationship Diagrams. Professionals of this type would not be openly receptive to changing their existing work culture and switching to modern OO techniques such as Object Model, Use Case, and interactions Diagrams. This kind of human culture change can form a major challenge during the transition that may increase people's resistance to change.

The Challenges of Human Factors

In general, human aspects are the most difficult challenge to be addressed during any organizational change (Zakaria & Yusof, 2001). The transition process to OO and the adoption of an OO process usually involves a large number of technical as well as non-technical issues. Various authors have referred to these non-technical issues as *soft factors* (Constantine, 1995) and *sociological factors* (DeMarco & Lister, 1987). We call these 'human' Key Success Factors in this discussion, since they deal with the 'human aspect' of change. These human factors necessarily require complete understanding and managing alongside the technological factors. Since OO contains elements relevant to all stages of the development lifecycle (not only coding), and includes models for requirements engineering, project management, team building, and so on, adopting OO process requires a combination of learning about technical issues as well as larger scale sociological issues.

People, procedures, and tools are the three critical aspects that must be well planned and managed during any organizational change (Serour, Henderson-Sellers, Hughes, Winder, & Chow, 2002). Certainly out of these three aspects, people are the most demanding aspect to be changed and managed. Eason (1983) argues that the human factors of technology are far more important than technical factors. The most challenging aspects of the transitioning to object-orientation remain in establishing a new software development environment and in introducing a new work culture to managers, software developers, and customers (Ushakov, 2000).

As a result, it is imprudent for management to expect every team member to agree with all the proposed changes. It may even be unwise to anticipate full commitment and belief in the new organization mission, especially at the early stage of transitioning. Organizations must face reality by realizing that change is often difficult in the best of circumstances and seldom goes exactly to plan. Management must be mindful that only a minority of individuals will wholeheartedly welcome any proposed changes to the way they have done things for so long.

The challenge to software professionals is to change and adapt to a new environment. A pos-

sible scenario may ensue where developers need to adopt new ways of 'thinking' about software, followed by OO modeling/designing, developing, quality assuring, and testing the software. Adopting a new approach of developing software may include technical factors (e.g., CASE tools, Programming languages, and databases) that require a reasonable degree of human culture change in order to utilize them to their maximum potential. Individuals, especially those working on software projects, work under the influence of various personal, motivational, and social factors. These factors, more often than not, remain in the background. Due to their elusive and intangible nature, these human factors are often difficult to discern, analyze, and improve on when change is about to be implemented.

As a result, moving working professionals to new ways of perceiving and undertaking various tasks is very difficult (Fingar, 1996). However, many studies such as Szewczak and Khosrow-Pour (1996) suggest many organizations are still struggling to deal with problems related to the human aspects of technology. Furthermore, they correlated this problem to management's lack of commitment to the human side of IT.

It is vital, then, that organizations pay careful attention in addressing all human or soft factors, and be open and ready to confront and solve any conflicts that may influence the entire transition. Organizations need to appreciate individual behavior when examining the transition and the adoption of new OO process; they need to investigate and explore the different factors that influence the response of individuals within the organization towards the new technology. To accomplish a successful transition, an organization needs to advance people's motivation and enthusiasm, maintain management commitment, and provide efficient and persuasive mentors whose leadership may be seen as the source of expert guidance (Jacobson et al., 1995).

The art of managing cultural interfaces has become an everyday business challenge at every organizational level (O'Hara-Devereaux & Johansen, 1994). The main question here, and management's major challenge, is how an organization makes the transition a driving force for all people involved in being adopters and supporters of the change, instead of opposers or neutral players. Constantine (1996), during the OOPSLA'96 panel discussion pertaining to human factors, contended that it is quite easy to communicate with machines and solve problems, but with people it is difficult. The main reason is because people are very difficult to "generalize."

Human Factors and Risk Issues

Organizations must be aware of the consequences and the possible risks involved in mismanaging the human factors during the introduction of a new work culture such as OO process. The improper planning and managing of human factors can easily lead to undesirable consequences, such as building resistance to change, and adding more confusion, uncertainty, and fear that can considerably diminish the chance of success. Schein (1999) interestingly states that there was no such failure for technology adoption; instead it was a failure to understand the organizational and the individuals' culture.

As a result, organizations that are aiming to adopt an OO process need to address not only the technological factors of the adoption, but also the human factors. As an example, organizations must provide adequate education and training to their people in order to understand and grasp the fundamentals and underpinning concepts of object-orientation and development processes, as it is a critical factor to avoid those risky consequences. Bridges (1995) argues that the most prevailing cause for the unsuccessful implantation of organizational changes can be attributed

to a lack of planning in managing the impact of change on individuals. He also emphasizes that many organizations that are not properly geared to handle this facet of change can run the major risk of jeopardizing their own existence.

Therefore, management may be very well advised to create a plan to manage and mitigate these potential risk factors by answering the following questions:

- How are people likely to respond to the change?
- What is required to convince people that the change is worth the effort?
- What actions are necessary to earn people's support and commitment to the change?

In addition, management needs to identify the ultimate and most effective means in achieving the following objectives:

- Selling the change to all people involved, including customers.
- Motivating people to make the transition.
- Reducing people's resistance to change.
- Eliminating people's fear and uncertainty.
- Minimizing the change's disruption to people.
- Mitigating the increasing tensions between people during the change.
- Encouraging people to be enthusiastic for the change, as opposed to being apathetic and an obstacle.

By answering the above questions and achieving the related objectives, organizations will be able to use persuasive approaches to human change, ensuring that everyone is comfortable and willing to accept, and make use of, the new OO process with all its associated changes.

THE HUMAN FACTORS

Human Culture and Necessary Culture Change

"In a time of rapid change, standing still is the most dangerous course of action." Brian Tracy

The Oxford English Dictionary (9th edition) broadly defines human culture as the arts and other manifestations of human intellectual achievement regarded collectively as the improvement by mental or physical training. In particular, Palvia, Palvia, and Roche (1996) define the culture of IT professionals as the set of values and practices shared by these members of an organization involved in information technology activities, including managers, developers, and customers/end users.

Personal culture is usually characterized and distinguished by the individual's values, such as behavior, attitude, experience, and beliefs. There are people who work well under pressure, whereas others work well only when properly supervised and directed, and then there are the 'cowboys' (Constantine, 1993) who prefer to work on their own. Cooper (1994) asserts that an IT person's culture may resist the introduction of a new process which realigns status, power, and working habits, especially when they violate some of the group's shared values. Current personal culture can be incompatible with certain new processes to the degree that risky consequences may be incurred, including resistance to change, a negative attitude and behavior, implementation failure, or the achievement of totally unsatisfactory results. Human culture change—that is, the physiological change that people undergo to alter the way they carry out their work on a daily basis—is one of the hardest and most longstanding parts of the adoption of an OO software development process (Fayad & Laitinen, 1998).

Natural Resistance to Change

Coupled with the introduction of a new work culture, people tend to naturally build resistance to any challenge of changing their culture and/or learning new things. The unfamiliarity with the new changes can lead to a discomfort that naturally increases people's resistance. In general, change is often seen as a personal threat by those involved in transition (Huse, 1975).

Resistance to change can also come from management, project leaders, and customers/end users for similar reasons, including fear of change and uncertainty of their capability of carrying out these changes (Fayad & Laitinen, 1998). Furthermore, adopting an OO process also requires people to advance their skills and knowledge, and/or gain new ones, as well as learning new tools and techniques. All these changes can lead to a threat that, if people are not capable of changing their culture, they will be out of the workforce, and be replaced by others who possess the required OO knowledge and skills and are capable of utilizing the new process. Resistance may also happen during the course of adoption when people are faced with serious impediments. For example, people may reach a stage when they feel that they cannot use their old ways (ad hoc) and at the same time they are not comfortable with the new ways (OO process). They then try to escape or oppose the changes. This leads to an increased level of resistance.

During organizational change, managing people's resistance becomes a critical issue that must be seriously considered so as to accomplish satisfactory results. For that reason, organizations must be able to effectively manage people's resistance to leading and directing the change process. To do so, management must first understand what resistance really means. What do people really resist? Do they resist the new environment, new technology, or the changes they have to undertake? And finally, why do people really resist? Do they resist for psychological reasons, technological issues, personal concerns, or a combination of all?

What Resistance Really Means?

Naturally, people want to improve and find better ways of doing things. A part of making improvements is causing changes, and changes are always faced with different types of resistance. People's resistance can be a result of different human reactions that sometimes form obstacles, impediments, hindrances, and difficulties to change. One of the risky issues regarding people's resistance is that, sometimes, managers see resistance as a sign of laziness, stupidity, or just unwillingness and opposition to change (Fayad & Laitinen, 1998).

In actual fact, resistance can be a sign of people's disinterest or they could be busy with more pressing issues. Resistance could also be a signal of conflict of interest or contradictory point of views. Moreover, resistance could be a silent request for assistance, more information, or an assertion of different priorities. Resistance can be viewed as an opportunity to gather information and learn better about current and desired state (Bamberger, 2002).

What People Really Resist?

Naturally, people do not like to lose. They do not like their own things to be taken away from them, and that is exactly what people resist. People associate change with the loss of their existing comforts. People do not resist the change itself, so much as they resist the uncertainties, fear, and discomforts associated with it. People, especially those who are confident and comfortable with their existing culture, resist the idea of changing their ways of doing things, and facing the risk of becoming unfamiliar and uncomfortable with the new ways. In addition, every change involves a degree of risk, and people are naturally reluctant to take risks and face the unknown.

Why People Really Resist?

To manage the resistance to change, it is important first to understand the various reasons behind it. Resistance could happen at an early stage of the introduction of the new change and/or at a later stage, during the change process. People resist change when they are unaware of the need to change and, accordingly, are uncertain of the final result. On the other hand, Bamberger (2002) notes that change often meets with resistance because it is seen as a personal threat that leads to fear of failure and rejection. Fayad and Laitinen (1998) relate resistance to the lack of a clear view of the current state and objective goals. They further note that resistance to change often exists as a result of structural conflicts within the organization. Lack of management's commitment and inconsistent actions with the new change can also elicit resistance.

Bridges (1995) declares that when changes take place, people get angry, sad, frightened, depressed, and confused. These emotional states can be mistaken for bad morale, but they rarely are. Rather they are more likely to be a sign of grieving, the natural sequence of emotions people go through when they lose something that matters to them. People resist the loss of competence that they once had and which was associated with their old familiar tasks. Transition is tiring; during the change, people build resistance when they feel unfamiliar and uncomfortable with the new ways. Resistance happens when people feel that they cannot use their existing old ways, and at the same time they are not comfortable and familiar with the new ways. From a different perspective, people build resistance when they feel that the new ways they have to follow can negatively affect their productivity. For example, an inappropriate new process or technique can discourage people to change, as it can lead to a drop in people's productivity.

Accordingly, with the introduction of a new OO process, people's resistance to transition and culture change could be a result of one or more of the following human behavior factors:

- They see the transition with its necessary changes as a threat to their jobs.
- They do not have inadequate knowledge and experience related to the new OO process.
- They are afraid of learning new ways of carrying out their jobs.
- They doubt the benefits of adopting a new OO process.
- They are afraid of failure or that they will not be able to understand the new process.
- It is exhausting to learn new things, and some think they are too old to learn a new process.
- Some prefer to do what they know best, even when they acknowledge there may possibly be a better way *(The devil you know!).*
- They are overloaded with their current projects with no time to learn new things.
- They are not in favor of the new process with its associated modeling language and/or CASE tools.

Even during the transition process, and when changes take place, people can develop more resistance and be less motivated for different reasons including:

- Anxiety rises and motivation falls.
- They are afraid of failing.
- They are unsure of the new way.
- They are afraid that they may be blamed if something goes wrong.
- They try to avoid learning new things.
- They become self-protective.
- They respond slowly and want to go back to the "old way."
- They doubt the benefits of the new way.
- They feel panic and confused.

Unsurprisingly, even customers may develop some resistance and be reluctant to accept the

changes as a result of the risk factors involved in introducing a new process that would affect their products. Therefore, customers must be involved in the change process owing to their effective role in supporting the organization's adoption of a new OO process. The customer role needs to be changed from being just a customer requesting and running a software application to that of being an effective and supportive partner in the whole process of producing their products.

Defeating People's Resistance to Change

As discussed above, many managers see resistance as a sign of laziness, stupidity, or just plain perversity on the part of employees (Fayad & Laitinen, 1998). Lawrence (1969) suggests that resistance to change should not be treated as a problem, but rather as an expected symptom and an opportunity, or a request, for learning and better knowing the unknown. Also, Senge (1990) stated that resistance to change generally has a real basis that must be understood in order for it to be dealt with. In order to manage people's resistance to change, organizations need to understand better the very human reactions they face when they are asked to do something differently or change their work habits. Management must plan and practice some strategies to manage and deal with people's resistance.

To do so, management must begin to understand what "resistance" really is? What are the major reasons for people to build resistance to change? Then they must establish a suitable work environment for people to undergo the required changes. Management cannot change people, they have to change themselves and they only change when they have the appropriate and supportive environment (Boyett & Boyett, 2000). Bridges (1995) reported that most managers and leaders put only 10% of their energy into selling the problem, but 90% into selling the solution to the problem. Management must put more energy into "selling" the problem that is the reason for the change, because people are not interested in looking for a solution to a problem they do not know they have. Management must realize that culture change is very tiring. People will feel tired, overwhelmed, down, and depressed. Management must look at those symptoms as a natural human reaction, and work hard to rebuild its people self-confidence and give them the feeling of competence in mastering the new way. Management must realize their people's capabilities and give them what they can do best without any fear of failure. Anxiety is natural, and the best way to defeat it is to educate and train people on their new environment. Adequate education and training regarding the newly adopted process can easily eliminate the fear of the unknown, the uncertainties about the final result, and thus lead to elimination of people's resistance.

Defeating Resistance with Participation

One of the most effective ingredients to defeat people's resistance to change is by encouraging them at an early stage to participate in planning for the change; Huse (1975) confirms this fact by assuring that people—during transition—see change as a threat unless they have participated in its planning. Lawrence (1969) has demonstrated through one of his case studies, where an identical change was introduced to several factory groups, that the first group, who was not offered any explanation, resisted all management's efforts, whereas the second group, who was involved in the change planning, carried out its transition with minimal resistance, and an initial small productivity drop was rapidly recovered.

Involvement of people in planning their change allows them to understand why they need to go through it and what they should expect. Humphrey (1995) affirms that people's resistance can be gradually changed to acceptance, once people are convinced of the necessity of the changes, and also they can see the value of undergoing

the change. So, management must show and convince people that the change they need to go through is not a threat, but rather an opportunity for improvement.

Defeating Resistance with Small Wins

Humphrey (1995) asserts that people's resistance to change is proportional to its magnitude. Therefore, resistance can be reduced by planning a number of small changes instead of a hefty change. Transitioning an IT organization to OO environment and adopting an OO process involves outsized changes. Psychological, organizational, and technological changes that can be planned in an incremental manner are "*small wins.*" By introducing the new changes in small increments, each increment will be easy to sell and implement. People will feel confident and positive every time they successfully achieve one increment, and become even more enthusiastic and motivated to implement the next increment. This technique can lead to a smooth transition by enhancing people's willingness to participate and reducing their total resistance. For example, the process of adopting a new OO method could well be started by addressing the Requirements Engineering (RE) activity as a major focus to engender everyone's involvement. An initial RE approach, using a well-defined technique such as the use case technique, can be introduced and utilized without the burden of all other activities and techniques. Once people feel confident in carrying out the RE activity, another major activity such as user interface design can be introduced in the same manner and so on for all other activities.

Education and Training (Knowledge and Skills)

Younessi and Marut (2002) define education as an opportunity to learn and ideally master a number of theories—principles that enable the recipient to assess, analyze, and act appropriately to a broad range of relevant situations. The impact of education is usually more abstract and wide in scope. They also defined training as the provision of an opportunity to learn and ideally practice some skills in a controlled environment to carry out a particular task(s). The impact of training is usually more focused, direct, and narrow in scope. In other words, education provides people with the answer to "*know what*," whereas training provides them with the answer to "*know how.*" People can gain knowledge and skills either through education and training courses provided by their organization and/or through real-life work experience.

In general, people need education and training in their discipline to enable and empower them to assess and perform their duties in a professional and satisfactory manner. In the context of the adoption of a new OO process, technical people need to learn about the new process and associated technology to feel comfortable when using it (Zakaria & Yusof, 2001). The individual's knowledge and experience related to the new process play an effective role in making the decision for the adoption. People with adequate and appropriate knowledge and experience of their new OO process can be more self-motivated and enthusiastic to make a transition than others. On the other hand, during an organizational change, lack of knowledge and/or experience could elicit people's resistance to change and increase their feeling of incompetency and frustration.

Necessity of Education and Training for Transition

Younessi and Marut (2002) have emphasized the vital role of education and training during the adoption of a new OO process by suggesting that education and training is an obligatory component for the successful introduction of software processes into an organization. Furthermore, they considered education and training as a Critical Success Factor for adopting these processes. Perkins and Rao (1990) stated that

the more knowledge and experience people have, the more they are able to contribute to decisions regarding the adoption of OO processes. They also emphasized the impact of training related to the new technology that increases their ability to adopt, diffuse, and master processes, techniques, and tools. Conner (1992) demonstrates the imperative role of education and training in adoption by saying that people change only when they have the capacity to do so.

People's experience towards OO processes should have a positive impact on the adoption (process) of such technology. In the context of adopting a new OO process, Sultan and Chan (2000) stated that the greater the knowledge and skills of individuals, the more likely they are to adopt it.

Management must recognize the critical nature of proper education and training, since experience has shown that between 25 and 40% of the total cost of an extensive project will be spent on education and training (Mize, 1987). More experience enables people to contribute more towards the transition and the adoption of new OO processes. People's appropriate knowledge, experience, and education may have a positive impact on their transition. The greater the work experience of individuals with the organization, the more likely they are to transition and adopt the new process. Highly skilled staff can manage themselves more easily, particularly during a paradigm shift. Therefore, an individual's satisfactory level of knowledge and education towards the new technology and processes forms another imperative management challenge.

Motivation

"Motivation is the release of power within a person to accomplish some desired results. It is a major key for achievement and success." Dr. Len Restall

Motivation has a remarkable power and is a major influence on people's behavior towards any achievement (Humphrey, 1997). In order to investigate the importance of motivation during the organizational adoption of OO processes, we need to understand what motivation is, its role during the adoption, and what motivates people.

What is Motivation?

Bolton (2002) defines motivation as a sociological concept used to describe individual factors that produce and maintain certain sorts of human behavior towards a goal. Hence, motivation, as a goal-directed behavior, is a driving force that stimulates people to achieve a set of planned goals.

Motivation is often driven from a desire or an inspiration to accomplish a defined objective, combined with the ability to work towards that objective. In other words, Motivation is the ability of taking good ideas or serious changes, and coupling them with appropriate knowledge and skills to achieve desired objectives.

Consequently, people who are aiming to achieve a specific goal must be both motivated and have the power and ability to work towards that goal. People who are motivated towards an accomplishment must be also capable and empowered to do so (carry out the work).

In the context of this chapter, motivating people during the adoption of a new OO software process could mean those factors which cause individuals within the organization to do more than they otherwise would. For example, to transit a development team to a totally new OO environment and/or adopting a new process, people need to work more than usual to change their existing work culture and adopt a new way. Then, motivation becomes a measure of an individual's level of readiness and willingness to participate effectively towards a successful transition.

Role of Motivation During Adoption

Pfeffer (1982) states that when people understand and accept a goal, and believe they can meet it, they will generally work very hard to do so. Additionally and from a different perspective, Maslow (1970), when he described his "*Hierarchy of Needs,*" stated that people are capable of achieving their goals if they believe that they can achieve them.

Motivation provides people with the understanding of the reasons for the change that increases their acceptability, which in turn improves their ability to accomplish a successful mission. Moreover, motivated people more often than not are capable of defining and achieving their own goals.

During an organizational change, people need to have compelling and persuasive reason(s) why they have to go through changes. Once they are convinced and believe in their mission, they will feel more competent to carry out their necessary changes successfully. This becomes a positive motivation that makes people desire to accomplish their goals that result in their very best performance (Humphrey, 1997).

On the other hand, lack of motivation during adoption can lead to negative consequences that may contribute to undesirable results. Those consequences may include fear, confusion, frustration, and uncertainty. Humphrey (1997) confirms the vital role of motivation by saying, "Without motivated and capable employees, no technical organization can prosper."

As a result, management must improve and maintain people's motivation to help them to be more effective and efficient in moving to their desired work environment with the adoption and utilization of a formal OO process. Motivation addresses the degree to which people want to, and are willing to, complete the work necessary to change their existing work culture. Furthermore, during transitioning to a new work environment, management must maintain people's motivation if they try to give up easily, or too soon, so as to encourage them to keep trying as long as they believe in their goals.

What Motivates People?

Bolton (2002) suggests that a good first step towards understanding what motivates people is to know what people want from their jobs. The answer could be gaining financial advantages, acquiring more skills and knowledge, or working with the latest technologies. In reality, it is very difficult to predict and judge on people's desire because it depends on the individual's values and beliefs.

What motivates people to change their work culture and adopt a new process differs, all depending on their needs and perception of the new process. Hence, not all people can be motivated by the same things and to the same degree. Therefore, management—especially during transition—needs to understand why their people do their work, and consequently elicit what motivates them and how to maintain that motivation through the entire process. Understanding people's aspiration can help management not only motivate people, but also support their motivation by maintaining and increasing the reasons for motivation.

Motivation is often seen as the driving force that moves people to perform some actions. Then, in the context of the adoption and diffusion of an OO process, people need a comfortable, familiar, valuable, and convincing driving force or "*motivation factor*" to move them to perform more effectively than they usually do.

People feel comfortable when they confidently know how to do their jobs. Enhancing people's skills and knowledge to the required level for transition makes them feel comfortable with the new process. Formal and professional education and training pertaining to OO processes are considered to be effective and efficient techniques

to achieve people's self-confidence. In order to motivate people, training must include clarification of language and jargon or commands used to avoid further frustration and anxiety, as new technologies usually have a mystifying and alienating potential. The more comfortable people feel with the new technology (here OO process), the more willing they are to experiment with it (Zakaria & Yusof, 2001).

Motivation factors must also add extra values for people such as financial reward, or learning a new language, process, or tool. For example, someone likely to retire in a year is unlikely to learn a new way of doing things. On the other hand, an enthusiastic newcomer to the project is likely to put in the extra effort needed to pick up new methods of thinking and modeling.

People who are in the move (e.g., changing profession, changing company, or reaching retirement) will not be interested in changing their culture, and they become difficult to motivate to change their current culture because they will not gain any benefits. To motivate people to change their culture, they need assurance of benefiting from the change.

The driving force must also be convincing, it must be relevant to what people usually do, and it must be in a way that people can believe and make use of. Most people must be convinced of the need to change before they will willingly comply (Humphrey, 1995).

Leadership

"Leadership is the ability to provide those functions required for successful group action." Weldon Moffitt

Leadership is well thought-out by many researchers and authors as a fundamental activity of project management, and they simply describe it as motivation plus organization (Stogdill, 1974; Thite, 2001; Phillips, 2002; Castle, Luong, & Harris, 2002).

In general, leadership plays a significant role in how people effectively perform their jobs. Rogers (1995) defines leadership as the degree to which an individual is able to guide, direct, and influence other individuals' attitudes informally in a desired way. Stogdill (1974) describes leadership as a process of influencing group activities toward goal setting and goal achievement.

Leadership is a practice that needs special skills and talent to motivate people to get things done in a favorable and constructive way. Bridges (1995) affirms that leading a team of professionals efficiently is an invaluable resource to any leader who is eager and willing to understand how to inspire team members. Also, it is a vital ability to augment teams' cooperation and individuals' commitment and productive participation. Above all, an effective leadership is a Critical Success Factor on how team members become adaptable to changing circumstances (Thite, 2000).

Leadership is basically situational. Fiedler (1967) suggests that there is no one style of leadership that suits every project situation. Rather, it should be contingent upon the nature of the organization, the team, and the project situation. Leadership style should be flexible and agile enough to be adapted to suit the project at hand in the most appropriate manner. Phillips (2002) claims that the "appropriate" leadership style is the style with the highest probability of success. Additionally, a good leader should effectively mix and cooperate with team members.

Leadership Style for Adoption

"A leader is best when people barely know that he/she exists." Whitter Bynner

During an organizational change situation, leadership style must be adapted to promote and support the organizational change. Social factors such as the style of leadership can influence the individual's ability to transit to a new work climate. Thite (2000) emphasizes the importance of

the appropriate leadership style during a change situation by considering it as a Critical Success Factor.

Changing people's work culture and introducing them to a new way of doing their job is usually accompanied by increasing anxiety, ambiguity, and insecurity. During such time, a supportive leadership style can effectively contend with and overcome these problems.

Due to the fact that an effective leadership style should depend on the follower and the project situation, leaders must use the most appropriate style of leadership that best suits the project situation that yields the best chance of success (Phillips, 2002). Furthermore, Phillips states that managers must be able not only to determine the most appropriate leadership style, but also to apply that style correctly.

Stodgill (1974) supports that argument by saying, "The most effective leaders exhibit a degree of versatility and flexibility that enables them to adapt their behavior to the changing and contradictory demands made on them." Moreover, Hersey, Blanchard, and Johnson (1996) proclaim that successful and effective leaders are able to adapt their leadership style to best fit the requirements of the situation. Therefore, leaders within an organization about to introduce a major change to their work environment—such as the adoption of a new OO process—must have adequate resources (mainly time), a strong interest in the mission, and a greater exposure to the new process they are about to adopt. Sultan and Chan (2000) firmly assert that the greater the opinion leadership among group members, the greater the chance of a successful transition and adoption.

Leadership always means responsibility. Consequently, an effective leadership style for a transitioning organization must be driven from top management with full support and commitment, and it must be also coupled with a rational degree of authority. Co, Patuwo, and Hu (1998) support that by saying: "Ideally, the team leadership should come from top management: one who is a 'doer', and who commands respect throughout the entire company."

During the organizational transition to a new work environment, leadership should foster strong alignment with the organization's vision. Team members that depend primarily on strong alignment with their organization's vision are at their best with well-understood practices applied to well-understood problems. Furthermore, successful leaders must be able to create a new work environment in which team members feel comfortable and familiar, and also enjoy what they are doing.

Lockwood (1991) and Constantine (1994) affirm that the working culture shapes the style in which software development is carried out. Different styles of organization during different situations require somewhat different forms of leadership style and management, and each will tend to have somewhat different software practices.

Leaders who are leading their teams through a serious change—such as adopting a new OO process—must be able to:

- Define a clear vision and mission.
- Provide convincing and compelling reasons for the change.
- Focus on and be concerned with the *why to* aspect of the transition, rather than the *how to*.
- Manage and motivate team members.
- Maintain an authority level to make appropriate decisions in the face of uncertainty.
- Provide full support and commitment to their followers.
- Provide adequate and appropriate resources including time, education, and training.
- Establish clear communication channels with team members.
- Monitor and review tasks allocated to team members.
- Recognize and appreciate achievements, and reward people for doing good work.

Perception of OO Processes

While IT organizations perceive new OO processes as an essential means to advance their existing software development culture and effectively compete in the marketplace, individuals could have a different perception. Since people are different, their needs and expectations are also different. The more positively people perceive their new process with regard to its characteristics—advantages, observability, compatibility, complexity, and trialability—the more likely it is that the process will be adopted and utilized.

Management of Expectations

From past projects, it has been proven that people with life experience are able to deal with conflicts of interest that may arise due to their ability to make any decision regarding the transition (Hill, Smith, & Mann, 1987). To emphasize the vital role of education on conflict resolution, Barclay (1991) stated that any organization's members who have adequate education and significant real-life experience are well prepared to deal with interdepartmental conflict. It has to be made very clear from the beginning, to both managers and developers, that following a new OO process is not a magic wand or a silver bullet to make the entire organization's dreams come true, but rather is simply today's best approach option for software development. Nonetheless, it is also important that managers appreciate not only the benefits of adopting a new process, but also become aware of all the pitfalls and consequences of changing people's existing work culture.

Different stakeholders such as managers, developers, and customers may look at, assess, and evaluate OO processes, as a new technology to be adopted and utilized, from a different perspective. Senior managers are always looking for the dollar value return for their spending and investments. They evaluate the technology based on how much it will reduce the cost of software production with improved quality. Project managers and team leaders may assess the new OO processes from a different perspective. They emphasize project management aspects, gaining more control and meeting customer's expectations with on-time delivery based on building the software system from the pre-tested and proven software components. Software developers may value the proposed process for adoption as the new fad and trend to developing software. They view the transition process as a good opportunity to acquire new skills, and learn new tools and techniques. Unquestionably, this will insert desirable additions in their resume, add value to their worth in the market, and make them in more demand.

From the above discussion, unrealistic expectation from adopting a new OO process and also any conflict of people's interest can have a negative impact on the entire transition process. People will be reluctant to work effectively together as a team if they do not share the same interest and/or have different expectations.

In order to overcome the barrier of the conflict of interest, management and champion teams must include a specific plan to resolve that conflict between people, and reach mutual perceptions, consensus, and understanding.

Proper education, mentoring, and open discussions are examples of good techniques that can be used to achieve good understanding and realistic expectations of the new process. Stakeholders must come to some sort of consensus as to what they can expect from adopting a new process and also should be fully aware of the new technology pitfalls as much as its benefits. Stakeholders' consensus and shared understanding, as well as realistic expectations of the new process, can lead to a good working environment that can positively impact on the process of transition and thus the quality of the software produced.

Sharing the Vision with the Organization

People perform better if they feel that they share the value of the change with their firm. Managers need to provide the right organizational climate to ensure that their employees can see that, by working towards the organizational goals, they are also achieving some of their own goals. These goals could be financial rewards, or personal rewards such as the respect of their colleagues, or job satisfaction, or a combination of any number of things that the employee considers to be important.

During organizational change, such as the adoption and diffusion of an OO process, to gain people's acceptance, management should get people involved during the initial stage, listen to what they have to say, respect their view, and involve them in discussions and debates regarding the change. Follow up with them on any further progress. Give them the feeling that they are the owners and supporters of the change. Give them the impression that we are only interested in the change if it will help them and make their job more efficient and more enjoyable. Never blame the people for any faults or mistakes, but rather blame the product, the techniques, and/or the tools being used. Organizations need to try everything they can during the adoption of a new process to gain their people's willingness and support such as:

- Choose a flexible software development process/methodology to be tailored to best suit the organization's environment.
- Obtain senior and middle management support and commitments.
- Set an appropriate change management plan.
- Establish a transition support team to sell the changes to everyone and to deal with all transition issues.
- Plan for adequate resources.
- Plan for rewarding people behind the transition.
- Carry out the changes gradually and avoid overloading people.
- Plan for the appropriate type of education and training to enhance people's skills that minimize their fear and confusion.
- Introduce new ways in formal and informal ways as required.
- Never blame people for any failure; instead blame the process (*no* criticism).
- Listen to people and encourage their individual contribution.
- Reinforce the new way of following the new adopted process.
- Start with small jobs with high chance of success so they to gain self-confidence.
- Celebrate success with people.

Organizations should set a well-defined set of guiding beliefs that the mission of the transition and adopting a new OO process is worthy and must be shared with people. Individuals must be in harmony and agreement with an organization's value and goals. When individuals share the same values and beliefs as the organization, they form a psychological union as a group. They become more congruent to the organization. Thus, the efforts of adopting a new process are determined and rigorous. The sharing of ideas and commitments through cooperation and harmony within an organization often leads to an effective change and positive participation. Needless to say, whenever individual goals are in alignment, or close to alignment, with the organizational goals, then any initiative undertaken by the organization to adopt and diffuse a new technology—in our case, OO process—will be easily accepted and utilized by individuals. When people share the vision and beliefs with their organization, they feel that they share the ownerships of their new process and the accountability of the whole transition procedure.

This feeling can motivate the individuals and assist management to defeat resistance to change should it arise. A strong organizational culture is usually measured by the way in which the key values are intensely held and widely shared. It has a greater influence upon employees than weak culture.

Communication Channels

When Zakaria and Yusof (2001) analyzed the implementation of technological change from a user-centered approach, they asserted that communication is a critical element in ensuring the success of a new technological change such as the adoption of OO processes.

Communications are defined as a process by which individuals exchange information through a common system of behavior. The process of communication between individuals has an influence on organizational transition, because it has a strong impact in conjunction with the leadership. A strong and effective communication network has a strong positive impact on adoption of a new technology. Proper and open communication channels can, from one side, help management to express their commitment and support to their people. From the other side, these channels can give people opportunities to discuss and exchange their concerns regarding their transition in order to adopt the new process. This can strongly increase people's cooperation, which enhances their ability to carry out the required changes effectively. From a project management point of view, open communications keep management in continuous contact with their people, and they can closely observe the transition progress to quickly resolve any problems that may occur. Communication can take different forms such as meetings. During an organizational transition to a new work environment and adopting a new process, meetings—driven by management—are seen as ways of undoubting management's commitment and support to change.

CONCLUSION

It can be clearly noticed that by considering and managing the human factors in great detail, the chances of successful adoption and diffusion of a software development process are enhanced significantly. Based on our research projects and several empirical studies, we conclude the following:

- Human factors play a vital role during the organizational transition to adopt and diffuse an OO process, as they can form either a promoting or resisting force.
- Understanding of the new desired OO process work environment can assist management in putting together a plan to manage the necessary cultural change.
- During an organizational change, managing people's resistance becomes a critical issue that must be seriously considered so as to accomplish satisfactory results.
- Organizations must be able to effectively manage people's resistance to leading and directing the change process.
- Involving individuals in planning and making decisions can positively eliminate their resistance to change and enhance their ability to carry out a successful transition.
- The individual's knowledge and experience related to the new process play an effective role in making the decision for the transition.
- More knowledge and experience enable people to contribute more towards the transition and the adoption of new technologies, including OO processes.
- The greater the work experience of individuals with the organization, the more likely they are to transition and adopt the new process.
- Gaining adequate knowledge and proper training on OO processes and associated

tools can significantly contribute to enhancing people's ability to positively involve themselves in the transition process.

- The more knowledge and experience the organization's individuals have regarding OO processes, the more likely they are to change and adopt the new work culture and thus enhance the chances of a successful transition to a totally new OO software development environment.
- When people understand and accept a goal and believe they can meet it, they will generally work very hard to do so.
- Motivation provides people with the understanding of the reasons for the change that increases their acceptability, which in turn improves their ability to accomplish a successful transition.
- Maintaining and enhancing people's motivation during transition can be a driving force for people to wholeheartedly and positively participate.
- Successful and effective leaders are able to adapt their leadership style to best fit the requirements of the situation.
- The greater the opinion leadership among group members, the greater the chance of a successful transition and adoption.
- The more positively people perceive the new process with regard to its characteristics—advantages, observability, compatibility, complexity, and trialability—the more likely that the process will be adopted and effectively used.
- Management of expectations of the new process and conflict resolution can lead to a good working environment that can positively impact the change process.
- When individuals share the same values and beliefs with the organization, they are more likely to effectively participate towards the change.
- When people share the vision and beliefs with their organization, they feel that they share the ownerships and the accountability of the whole change practice and thus their new OO process.
- The sharing of ideas and commitments through cooperation and harmony within an organization often leads to an effective change and positive participation.
- The more individuals find values in the organizational change, the more likely they are to contribute to the transition success.
- The more congruent and rewarding the individuals perceive the organization's values, the more positive is the influence on the transition and the adoption.

REFERENCES

Auer, D. & Dobler, H. (2000). A model for migration to object-oriented software development with special emphasis on improvement of acceptance. *Proceedings of TOOLS Sydney 2000 Conference* (pp. 132-143). Los Alamitos, CA: IEEE Computer Society Press.

Bamberger, J. (2002). Managing resistance—techniques for managing change and improvement. *Asia Pacific Software Engineering Process Group (SEPG) Conference Handbook and CD-ROM* (p. 30), Hong Kong.

Barclay, D.W. (1991). Interdepartmental conflict in organizational buying: The impact of the organizational context. *Journal of Marketing Research, 18*(28), 145-159.

Bolton, L. (2002). Information technology and management. Retrieved December 15, 2002, from *http://opax.swin.edu.au/~388226/howto/it2/manage1.htm*

Boyett, J.H., & Boyett, T. (2000). The skills of excellence: The new knowledge requirements for the twenty-first century workplace. Retrieved June 3, 2002, from *http://www.jboyett.com/skillsof.htm*

Bridges, W. (1995). *Managing transitions, making the most of change*. Nicholas Brealey.

Burshy, D. (2001). Technology adoption—many roadblocks slow it down. *Electronic Design, 49*(9), 20-21.

Castle, R.D., Luong, H.S., & Harris, H. (2002). A holistic approach to organisational learning for leadership development. *Proceedings of the IEEE International Engineering Management Conference*, Cambridge, UK.

Co, H.C., Patuwo, B.E., & Hu, M.Y. (1998). The human factor in advanced manufacturing technology adoption: An empirical analysis. *International Journal of Operations & Production Management, 18*(1), 87-106.

Conner, D.R. (1992). *Managing at the speed of change*. New York: Villard Books.

Constantine, L.L. (1993). Coding cowboys and software sages. *American Programmer, 6*(7), 11-17.

Constantine, L.L. (1994). Leading your team wherever they go. *Constantine: Team Leadership, Software Development, 2*(12), 1-6.

Constantine, L.L. (1995). *Constantine on Peopleware*. Englewood Cliffs, NJ: Prentice Hall.

Constantine, L.L. (1996, October). Panel on soft issues and other hard problems in software development (Ward Cunningham, Luke Hohmann, Norman Kerth). *Proceedings of OOPSLA'96* (pp. 6-10), San Jose, CA.

Cooper, R.B. (1994). The internal impact of culture on IT implementation. *Information and Management, 27*(1), 17-31.

DeMarco, T., & Lister, T. (1987). *Peopleware: Productive projects and teams.* Dorset House.

Eason, K.D. (1983). *The process of introducing information technology: behavior and information technology* (pp. 197-213). New York: Prentice-Hall.

Fayad, M.E., & Laitinen, M. (1998). *Transition to object-oriented software development.* New York: John Wiley & Sons.

Fiedler, F.E. (1967). *A theory of leadership effectiveness.* New York: McGraw-Hill.

Fingar, P. (1996). *The blueprint for business objects.* New York: SIGS Books.

Gibson, S. (1999). Videoconferencing still shy of adoption. *PC Week, 16*(13), 130-131.

Hersey, K.H., Blanchard, & Johnson, D.E. (1996). Management of organizational behavior. *Utilizing human resources* (8th ed.). Upper Saddle River, NJ: Prentice-Hall.

Hill, T., Smith, N.D., & Mann, M.F. (1987). Role of efficacy expectations in predicting the decision to use advanced technologies: The case of computers. *Journal of Applied Psychology, 72*(2), 307-313.

Humphrey, W.S. (1995). *A discipline for software engineering.* Reading, MA: Addison-Wesley.

Humphrey, W.S. (1997). *Managing technical people-innovation, teamwork, and the software process* (6th ed.). Reading, MA: Addison-Wesley-Longman.

Huse, E.F. (1975). *Organization development and change.* St. Paul, MN: West.

Ioannidis, A., & Gopalakrishnan, S. (1999). Determinants of global information management: An extension of existing models to firm in a developing country. *Journal of Global Information Management, 7*(3), 30-49.

Jacobson, I., Ericsson, M., & Jacobson, A. (1995). *The object advantage, business process reengineering with object technology.* Wokingham, UK: ACM Press.

Jurison, J. (2000). Perceived value and technology adoption across four end user groups. *Journal of End User Computing, 12*(4), 21-33.

Lawrence, P.R. (1969). How to deal with resistance to change. *Harvard Business Review,* 4-6.

Lockwood, L.A.D. (1991). Strategies for managing application development. *Fox Software Developers Conference Proceedings.* Toledo, OH: Fox Software.

Maslow, A. (1970). *Motivation and personality* (2nd ed.). Harper & Row.

Mize, J.H. (1987). *Success factors for advanced manufacturing systems.* Dearborn, MI: Society of Manufacturing Engineering.

Nambisan, S., & Wang, Y. (1999). Roadblocks to Web technology adoption? *Communications of the ACM, 42*(1), 98-101.

O'Hara-Devereaux, M., & Johansen, R. (1994). *GlobalWork* (p. 35).

Palvia, P.C., Palvia, S.C., & Roche, E.M. (1996). *Global information technology and systems management: Key issues and trends.* Nashua, NH: Ivy league Publishing.

Perkins, W.S., & Rao, R.C. (1990). The role of experience in information use and decision-making by marketing managers. *Journal of Marketing Research, 18*(27), 1-10.

Pfeffer, J. (1982). *Organizations and organization theory.* Marshfield, MA: Pitman.

Phillips, D.A. (2002). How effective is your leadership style? *IEEE Antenna's and Propagation Magazine, 44*(2), 124-125.

Schein, E.H. (1999). *Process consultation revisited: Building the helping relationship.* Reading, MA: Addison-Wesley.

Senge, P.M. (1990). *The fifth discipline: The art & practice of the learning organization.* New York: Doubleday/Currency.

Serour, M.K., Henderson-Sellers, B., Hughes, J., Winder, D., & Chow, L. (2002). Organizational transition to object technology: Theory and practice. *Proceedings of Object-Oriented Information Systems: 8th International Conference* (pp. 229-241), Montpellier, France. Berlin, Heidelberg: Springer-Verlag.

Stogdill, R.M. (1974). Historical friends in leadership theory and research. *Journal of Contemporary Business,* pp.7-7

Szewczak, E., & Khosrow-Pour, M. (1996). *The human side of information technology management* (p. 1). Hershey, PA: Idea Group Publishing.

Thite, M.R. (2000). Leadership styles in information technology projects. *The International Journal of Project Management, 18*(4), 235-241.

Thite, M.R. (2001). Help us but help yourself: The paradox of contemporary career management. *Career Development International, 6*(6), 312-317.

Ushakov, I.B. (2000). Introducing an OO technology in non-OO standard environment. *Proceedings of the 4th IEEE International Symposium and Forum on Software Engineering Standards* (pp. 1-5), Curitiba, Brazil.

Younessi, H., & Marut, W. (2002). *The impact of training and education in the success of introducing object-oriented into an organization.* Unpublished Paper.

Zakaria, N., & Yusof, S. (2001). The role of human and organizational culture in the context of technological change. *IEEE Software,* 83-87.

This work was previously published in Management of the Object-Oriented Development Process, edited by B. Roussev & L. Liu, pp. 247-277, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 5.23
Developing Software in a Bicultural Context:
The Role of a SoDIS®[1] Inspection

Don Gotterbarn
East Tennessee State University, USA

Tony Clear
Auckland University of Technology, New Zealand

Wayne Gray
The University of Auckland, New Zealand

Bryan Houliston
Auckland University of Technology, New Zealand

ABSTRACT

This article introduces the SoDIS process to identify ethical and social risks from software development in the context of designing software for the New Zealand Maori culture. In reviewing the SoDIS analysis for this project, the tensions between two cultures are explored with emphasis on the (in)compatibility between a Maori worldview and the values embedded in the SoDIS process. The article concludes with some reflections upon the key principles informing the professional development of software and ways in which cultural values are embedded in supposedly neutral technologies, and reviews the lessons learned about avoiding colonization while working on a bicultural project.

INTRODUCTION

Rogerson and Gotterbarn (1998) developed an early warning system that can be used by software developers to identify and address potential ethical, professional, and social risks in software development. The method, a software development impact statement (SoDIS) inspection (Gotterbarn, Clear, & Kwan, 2004), uses

ethical relations to connect broad-based project stakeholders to the project tasks and deliverables. The SoDIS methodology based upon standard Western ethical values has been proven effective in a variety of environments. It has been applied to the development of a data warehousing project in Boston, the development of software for the analysis of psychometric test data for youth, Web sites in New Zealand, and electronic voting in the UK. In each of these cases, the SoDIS inspection process identified significant risks in the development process and in the final product (Gotterbarn & Rogerson, 2005). This identification of the risks gave management the opportunity to develop successful risk mitigation strategies.

This article introduces the Software Development Impact Statement process and discusses its application to a bicultural project. In reviewing the SoDIS analysis for this project, the tensions between the two cultures are explored with emphasis on the compatibility between a Maori worldview and the values embedded in the SoDIS process. The article concludes with some reflections upon the key principles informing the professional development of software and reviews the lessons learned about working on a bicultural project.

The article recounts the application of the SoDIS process to an ethically sensitive project involving software development for a Maori Tribal Authority. Maori are the indigenous people of New Zealand, a bicultural society in which the other culture could broadly be termed *Western* (comprising subsequent New Zealand settlers from a predominantly European immigrant community). Clear and Gotterbarn used this opportunity to address action research questions such as the following:

1. Would the ethical connectives between task and stakeholder that had been derived from software codes of ethics and codes of practice be adequate to identify ethical risks in a manner that was sensitive to the indigenous culture in this bicultural context?
2. Would the use of the SoDIS process colonize the software system for Maori stakeholders with Western cultural values?

BACKGROUND AND TERMS RELATED TO THE PROJECT

Historical Context

The history of New Zealand, as with most former colonies, reflects a complex series of struggles between the colonizing settlers and the indigenous peoples. A unique feature of New Zealand history is the signing of the Treaty of Waitangi in 1840 between the British Crown and a large number of the indigenous Maori tribes. This controversial treaty, in which the Maori people engaged as equal parties in a partnership whereby they ceded a degree of sovereignty to the crown in exchange for certain rights, since has been regarded as "the legitimate source of constitutional government in New Zealand" (Walker, 1990, p. 98). The English and Maori language versions of the treaty differ in substantial ways and are each open to quite different interpretations. Subsequent debate has revolved around these differing interpretations of the treaty, and consequent actions have been taken by the Crown and its agents. Walker (1990, p. 98) observes that "acquisition, control and, ultimately, expropriation of land were the key factors in the consolidation of sovereignty" by the Crown. This was in spite of the treaty's guarantees to the Maori signatories of "full exclusive and undisturbed possession of their Lands and Estates, Forests, Fisheries, and other properties which they may collectively or individually possess, so long as it is their wish and desire to retain the same in their possession" (Walker, 1990, p.

92). He argues that the "outcome of colonization by the end of the century was impoverishment of the Maori, marginalisation of leaders and chiefly authority and a structural relationship of Pakeha [European] dominance and Maori subjection" (Walker, 1990, p. 10).

Relatively recently, a process of reconciliation has been engaged in by the Crown, one element of which has involved the devolution of responsibilities from the former Government Department of Maori Affairs to Maori tribal authorities known as Runanga. The intention was to move from a paternalistic governmental model of service provision to Maori toward a self-determined model managed by the Maoris themselves. An earlier key step in the reconciliation process had been the establishment in 1975 of a specialized tribunal (the Waitangi Tribunal) to hear and to adjudicate upon Crown breaches of the treaty brought by Maori claimants. A number of settlements have resulted from this process, in which significant financial compensations have been made to claimant tribes.

One of the most significant nationwide settlements resulting from the Maori Fisheries Act 1989 and the subsequent Deed of Settlement 1992 has resulted in the establishment of the Maori Fisheries Commission, a Crown body with more than NZD$700 million worth of fisheries assets to be redistributed to several claimant tribes. This redistribution will be effected by the establishment of a "new trust, Te Ohu Kai Moana … with iwi (the Maori tribes) as beneficiaries" (Maori Fisheries Bill, 2003, p. 1). The trust will undertake a process of allocation of groups of shares to tribal authorities, who then will apportion them accordingly to bona fide members of that tribe. "For individual Maori to become shareholders in this asset they must be registered with their iwi" (Tuhono, 2004). Thus, in order to demonstrate membership of a tribe, a process of tribal registration similar to voter registration will be required.

Project Context

In mid-2002, a number of students at Auckland University of Technology (AUT) began work on a project to extend the existing IT systems of a Maori tribal authority, Te Runanga a Iwi o Ngapuhi (TRAION), the statutory body representing the Ngapuhi tribe, or *iwi* (Clear et al., 2004). Proposed changes in a broadly conceived project included the following:

- Online registration of tribal members
- Linking members to several groupings of significance to Maori
  - Extended family (*whanau*)
  - Subtribe (*hapu*)
  - *Marae* (a meeting-house complex used for several cultural purposes and serving the Maori community centered in that location)
- Creating a database of genealogical (*whakapapa*) information
- Creating a database of interests in communally owned tribal land

The TRAION project, then, was entering into sensitive areas. For Maori, "Identity and worth were found in family and tribal connectedness [and] …identity was linked to both ancestry and place" (King, 2003, p. 77). As a consequence, Maori people have known sensitivities about research related to *whakapapa* (genealogical and land-based information), which is considered a *taonga* (treasured possession) particular to the groupings (*whanau*, *hapu*, and *iwi*) who have interests in this information.

To better articulate the risks and to investigate the issues inherent in computerizing such sensitive information, a Software Development Impact Statement (SoDIS) analysis was undertaken.

THE SODIS PROCESS

There is significant evidence that many software projects fail because, during development, they do not adequately identify and address significant qualitative risks, including social, professional, and ethical risks. Rogerson and Gotterbarn (1998) developed a process, Software Development Impact Statements (SoDIS), to address this problem. The SoDIS process was prototyped in software (SDRF, 2005). These unidentified risks and associated software failures can range from trivial annoyances to cumbersome and dangerous situations. Developers frequently are surprised by the impacts of the software they develop and their own failure to pay attention to a wide range of risks. Sometimes, the missed risks can have tragic consequences.

A major cause of failed software projects is the narrow focus of risk analysis on only the concerns of the developer and the customer. Risk analysis sometimes is extended also to include those with a financial stake in the software. But this limited scope of analysis excludes those who do not have a financial stake in the development of the software. Because they have no financial stake in the development of the software, the needs of the pacemaker recipient, or the vehicle passenger whose life is affected by the successful application of anti-lock braking software, often are overlooked. The SoDIS methodology is designed to include in the risk analysis those people who are impacted by the development of the software. Broadening the base of stakeholders requires a broadening of the types of risks considered.

The decision support technique, Software Development Impact Statements, is a modification of an environmental impact statement. Software impact statements, like environmental impact statements, are used to identify potential negative impacts of the planned software and to specify actions to mediate those impacts. SoDIS is intended to reflect both the software development process and the more general ethical obligations to various stakeholders. In particular, it was designed to access the possibility of ethical, professional, and social risks — risks that occur where the software interacts with people or modifies their social environments.

The SoDIS process is an ethical decision support process insofar as it focuses on the developer's responsibility to consider the diversity of stakeholders and the special nature of the project tasks. The SoDIS analysis precedes the software development. A SoDIS analysis identifies concerns about the potential impact of particular planned tasks on individual stakeholders. It highlights changes in some planned tasks and additional possible tasks that are needed to prevent any anticipated problems. The analysis is done first during the planning stage for software development.

On a very high level, the SoDIS process can be characterized as examining the way every task or function of a proposed system impacts a stakeholder. Unacceptable potential impacts are identified, and the developer has the opportunity to address the risk before the product is developed. The process generates a series of questions of the form, "Might task *n* have a specified ethical impact on the particular stakeholder?" For example, a retirement payment system may require all data on a single screen. "Failing to take into consideration the needs of" is an ethical impact issue. Thus, a SoDIS question might be, "Might putting all data on a single screen fail to take into consideration the needs of the senior citizen (visually impaired) stakeholder?"

The SoDIS process can be reduced to four stages: (1) the identification of the immediate and extended stakeholders in a project; (2) the analysis of the tasks in a project; (3) for every task for each stakeholder, the identification of potential ethical issues raised by the completion of that task; and (4) the recording of the details and solutions of those ethical issues and a decision whether the current task needs to be modified or a new task created in order to address the identified concern.

The process of developing a SoDIS encourages the developer to think of people, groups, or organizations related to the project (stakeholders in the project) and how they are related to each of the individual tasks that collectively constitute the project. This encouragement is done by generating a series of questions for the analyst.

The generation of these questions requires the list of tasks, the potential issues connecting (ethical relata and impact issues) task and stakeholder, and the list of stakeholders. The task list can be at a variety of levels of abstraction and is provided by the requirements, function lists, or a project task list.

The SoDIS process identifies overlooked risks through the inclusion in the analysis of a broader range of stakeholders. Stakeholders include individuals or groups who may be directly or indirectly affected by the project and thus have a stake in the development activities.

Stakeholder Identification Techniques

The stakeholders can be identified in several ways. A preliminary identification of software project stakeholders is accomplished by examining the system plan and goals to see who is affected and how they may be affected. When determining stakeholders, an analyst should ask whose behavior, daily routine, or work process will be affected by the development and delivery of this project; whose circumstances, job, livelihood, or community will be affected by the development and delivery of this project; and whose experiences will be affected by the development and delivery of this product. All those pointed to by these questions are stakeholders in the project.

The identification of stakeholders also must strike a balance between a large list of stakeholders who are ethically remote from the project and a list

Figure 1. SoDIS® process (Gotterbarn & Rogerson, 2005)

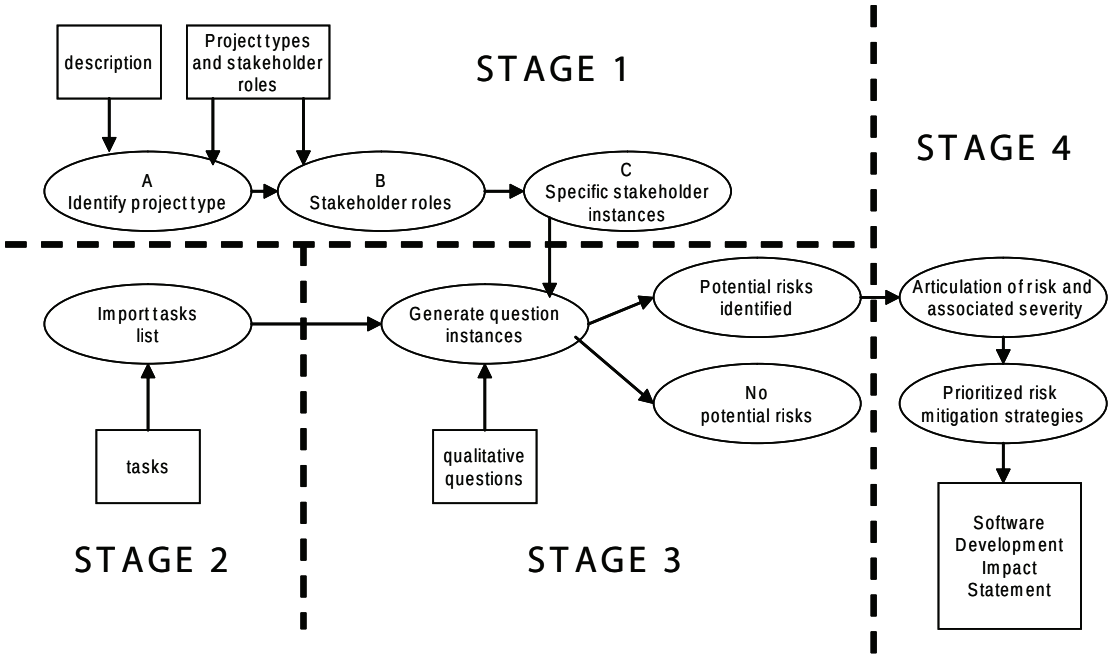


Table 1. Gert's moral rules

Don't kill	Don't cause pain	Don't disable	Don't deceive
Don't deprive of freedom	Don't cheat	Don't deprive of pleasure	Keep your promises
Do your duty	Obey the law		

of stakeholders that only includes a small portion of the ethically relevant stakeholders. Rogerson and Gotterbarn (1998) proposed a method to help achieve a balance between an underestimation and overestimation of the relevant stakeholders based on Gert's (1988) moral rules. Gert gives the 10 basic moral rules shown in Table 1. These rules carry with them a corresponding set of rights, such as the rights to liberty, physical security, personal liberty, free speech, and property.

One approach to stakeholder identification is to use Gert's (1988) rules as stakeholder search criteria, search for stakeholders who would be caused pain by a particular project task, search for stakeholders who would be disabled by a particular project task, and so forth. A preliminary identification of software project stakeholders is accomplished by listing each of these rules and rights and examining the system plan and goals to see who is affected and how they may be affected. For example, according to the rule "don't cause pain," we should ask if the system changes the level of pain felt by anyone.

Stakeholders are also those to whom the developer owes an obligation. The imperatives of the Software Engineering Code of Ethics and Professional Practice and similar codes of ethics and practice define the rights of the developer and other stakeholders. These imperatives can be used to guide the stakeholder search. The process of identifying stakeholders also identifies their rights and the developers' obligations to the stakeholders.

Ethical Relata

These codes also can be used to identify the impact issues connecting tasks and stakeholders. Many of the computing codes have similar imperatives, which embody the software profession's views of responsibility. The SoDIS process uses abstracted imperatives from many codes and organizes them under general moral principles. These imperatives have been organized as a set of 32 impact issues, which form questions when relating tasks and stakeholder. There may be some special circumstances that are not covered by these 32 questions, so the SoDIS analyst can add impact issues. Only codes of western professional computing societies were abstracted (e.g., the British Computer Society, the Association of Computing Machinery, etc.). There is in each of the codes a clear statement that protects individual rights of ownership. The ethical umbrella under which the SoDIS was developed seems to reflect a Western, largely individualistic worldview (see Hofstede [1980] and discussion in the section "The Ethical Problem") of software development and ethical rights and responsibilities.

We now can modify our initial research questions as follows:

1. Would the ethical connectives between task and stakeholder that had been derived from software codes of ethics and codes of practice be adequate to identify ethical risks in a manner that was sensitive to the indigenous culture in this bicultural context? This can be put as the following more general question:
   1*. Does the application of a Western worldview of software development constrain (impact, affect, colonize) the system it develops with more general ethical values (which would include indigenous values or the general value sets implicit in the Western worldview)?
2. Would the use of the SoDIS process colonize the Maori software system with Western cultural values? This is now divided into the following two questions:
   2*. Does the SoDIS process really embody Western values, or does it embody a view that is conditioned by software development rather than a social-ethical culture?
   2**. Does the SoDIS process necessarily force such constraints on a different culture?

Question 2** is different from the question, "Can the process be used in such a way to colonize values into another culture?" A simple example of software colonization is the loss of accountability in vote counting caused by the development of vote counting machines that do not produce paper audit trails for votes cast.

THE PROPOSED PROJECT AS TARGET FOR THE SODIS PROCESS

The Ngapuhi *iwi*, 103,000 according to the 2001 census, have their interests represented and managed by Te Runanga a Iwi o Ngapuhi, the project client. Because the majority of these members live outside the tribal boundaries in the Northland, identifying and maintaining contact with members is a costly and difficult exercise.

The software project, TRAION, was to replace the current paper-based membership system with only 2,000 recorded members. The *iwi* is very large and dispersed, but a complete roster is needed for representative decision making in *iwi* matters, such as the long-term management and reporting of tribal resources and the facilitation of Treaty of Waitangi settlements. Communicating with the diverse membership is facilitated by a Runanga-developed Web site (http://www.ngapuhi. iwi.nz). A major function of the Web site is the distribution of *iwi* membership application forms. "Nevertheless, providing a fully dynamic, robustly architected website had been beyond the resources of the Runanga at the time of commissioning the project" (Clear et al., 2004, p. 8).

The Tribal Membership System

One of the immediate needs of the Runanga was the establishment of an up-to-date membership system verifying that people, indeed, are members of the *iwi*. This list was required for *iwi* management and the potential disbursement of settlement funds, a task made more difficult by the international dispersal of members of the *iwi*.

Implications for the Project: Technical and Social Issues

To meet the membership recording needs of the Runanga, it was decided that the system would be Web-based. The project as originally conceived (i.e., developing a full genealogical/land-based system) could not be completed in the time allocated for a student project. "Developing a land information system revolves around not merely providing the right technology, but the difficulty

of the information gathering process and the accuracy and currency of the results garnered from the process" (Clear et al., 2004, p. 10).

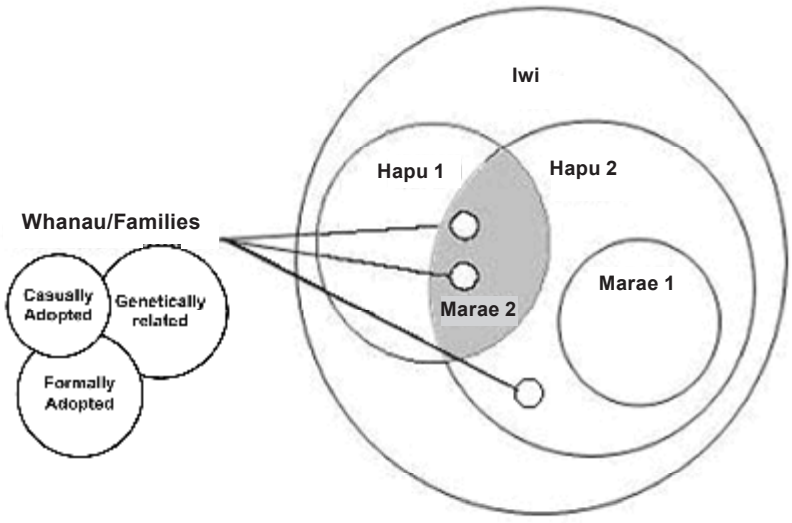
Determining land ownership is very complex for Maori. As in other societies, land is inherited through family linkages, but there is not a complete paper trail of all of these linkages. The most accurate information can be found by investigating the family relationships and background of land ownership candidates. Another problem is that the Maori culture is community-based, so there are often hundreds of owners for one block of Maori land, with each owning a proportion. The records for these dispersed owners are often inaccurate.

The three factors of land, genealogy, and tribal membership are interconnected and provided the original scope of the investigation. A decision was made to focus on the membership system requirements, with the database designed to allow for future implementation of genealogical and land information system requirements.

Membership Information Structure Difficulties

Given the lack of genealogical documentation, tribal debate informs matters of *whakapapa*, and every piece of information is important in genealogical research, regardless of the correctness of the information at the time of examination. Even dubious data must be preserved to allow for later reinvestigation. Not only must inconsistent stories be maintained, but the system also must track the time-based nature of genealogical data. Instead of a simple hierarchical structure, the Maori also have an overlapping network structure in different groups (i.e., family, *hapu*, tribe, etc.). The levels of family relationship (shown in Figure 2) include parent-child, *whanau* (extended families), *marae* clusters within *hapu* (local meeting complexes within the territories of a subtribe), and *hapu* clusters within a larger tribe (*iwi*). The Whanau is not a simple genetically related group but frequently includes members who have been

Figure 2. Iwi relationship structures (adapted from Clear et al., 2004)



adopted formally from other Whanau; it also includes those who have been adopted casually by a Whanau. Thus, someone who is adopted casually into a particular family will be a member of that family's *marae* of their *hapu* in their *iwi*, but that same person is related genetically to a different family that could belong to a completely different *iwi*.

THE SODIS APPLIED TO THIS PROJECT

Smith (1997) identified several Maori concerns about putting cultural information on the Internet, including concerns relating to threats to cultural values, loss of control of information, intellectual and cultural property ownership issues, accuracy and authority of information, the commercialization of information, and access issues. Based on consultation with the Maori about computerization of Land Court records, the Department of Courts (1999, p. 3) noted that Maori had objected to the database being made available over the Internet and believed "that management of the records (paper and electronic), the information therein, and access, has to be consistent with the following principles: the *mana* (authority) of the records/information comes from *iwi*, [and] *whakapapa* is intrinsically *tapu* (sacred)."

The Maori say that cultural information (here, *whakapapa*) has an intrinsic value, while in the West, information is viewed primarily in an instrumental sense; the value of the information is a function of its usefulness. In addition to the cultural concerns, there are significant technical issues with the TRAION project. The potential of the final outcome of this project to support not only the production of a tribal membership register but also the recording of *whakapapa* and consequential land ownership rights gave rise to natural concerns about the design of the system, use of data provided in the course of the project, the responsibilities for management of the resulting information, and access to this sensitive data.

The Ethical Problem

Given that *whakapapa* is sacred, would the analysis of the project tasks manage to identify the development of this database as a potential instance of cultural colonization of the Maori by another culture?

Our investigation requires at least a distinction between the two cultures. Culture is a complex phenomenon and generalizations are all too easy to make. Nonetheless, differences do exist, and some categorization such as the dimensions proposed by Hofstede (1980) can be helpful. In one such example, at a country level, New Zealand has been categorized, along with countries such as Canada, Great Britain, and the United States, as possessing an individualistic culture (Jarvenpaa & Leidner, 1998), whereas Maori would be classified more appropriately as possessing a collectivist culture. The group is a dominant structure in the Maori world, just as the individual is dominant in the Western perspective of the previous country grouping. Avison and Myers (1995, p. 52), arguing that culture is essentially an emergent phenomenon, have refuted simplistic definitions, noting, "The assumption that cultures are separate, distinct entities which identify and distinguish one group from another … is too simplistic." As with all generalizations, there are obvious exceptions with each culture, but the individualism — collectivism dimensions — seem to be based upon strong conceptual foundations. Given this obvious difference in perspective, forcing the values from one culture onto another would be an example of ethical/social colonization.

Therefore, when engaging with Maori as with many other indigenous cultures, we need to acknowledge that the Western perspective is different from the Maori. To develop software that accommodates this difference in perspective

requires that we work in a partnership model and that we have tools and techniques that can surface issues based on these different perceptions. The SoDIS process is one such analytical technique. The SoDIS was designed with a fiduciary model of professionalism in mind (Bayles, 1981). A fiduciary model of professionalism imposes high standards of development, but the resulting software is developed in partnership with those affected by the developed product. But we are wrestling with the extent to which its underpinning logic is essentially Western and fails also to address the different value systems of indigenous peoples who come from a strong tribal or clan-based community culture. The tools used require a focus on what that community values.

THE RESULTS OF THE FIRST APPLICATION

The SoDIS process is part of an inspection process. The first step in the SoDIS inspection process (Gotterbarn, Clear, & Kwan, 2004) is the identification of potential high-level risks — clusters of concern. The awareness of these high-level issues is used as a filter for the SoDIS analysis. In application of the SoDIS process through a SoDIS inspection of the TRAION project, the following four distinct clusters of concern were identified:

- Client-developer communications
- Sensitive data
- User experience
- Academic considerations

Within these clusters, the specific concerns later identified during the detailed SoDIS analysis related to several further subcategories.

Client-developer communication concerns that were identified related to a conflict of interest between the local proxy client, who was a member of the *iwi* and also of the academic staff at the developers site, and the remote client at the Runanga itself, who would operate the final systems once delivered. Questions arose about the ability of the proxy client to adequately represent the requirements to the developers and mechanisms for securely transferring sensitive project related documentation between the development and client sites.

Sensitive data concerns related to reasons for collecting data (consistency with privacy legislation and mechanisms for obtaining consent for provision of data for genealogical research purposes [http://www.caldeson.com/RIMOS/nzguide. html]) and access to data (determining authorization protocols, managing privacy considerations, keeping data over time, managing multiple copies of the database, etc.). Cultural sensitivity and avoiding exploitation through commercialization were concerns, especially over display of cultural artefacts over the Web or use of *whakapapa* information for commercial purposes.

The user experience concerns related to unusual circumstances (e.g., where contested values over tribal groupings were held and where members could be refused enrollment under their chosen groupings, or when dealing with cases of adoption and the desire to represent both natural and adoptive lineages); refusing membership and the resulting impact on tribal members, whether that refusal was by design or through administrative, security, or data integrity occasioned errors; access and performance issues for rural members with slow dial-up connections; visual and language impairments (e.g., the impact of Maori language text without translation for young members who are more frequently non-Maori speakers and for the elderly who have poorer eyesight and for whom the choice of a suitable font will be an important consideration).

Academic considerations were concerns related to developer inexperience with taking a system through all the activities of a full devel-

opment life cycle to production use, and time constraints potentially leading to shortcuts and quality compromises.

This SoDIS-generated list of Clusters of Concern and subcategories did not impose Western social values. The list appears as a relatively culturally neutral collection that has identified issues that would be of concern to any professional developer and likewise to their clients and other stakeholders in the system. As we shall see, there are, however, some issues here that are of specific concern to Maori due to their cultural priorities and values. We return to the research question 1* — Is the SoDIS with its Western cultural base capable of including indigenous values?

SOME SEEMINGLY CULTURAL DIFFERENCES RELATED TO THE DATABASE AND THREATS TO CULTURAL VALUES

In this clash of values between Maori and Western perspectives, however, we have identified some critical issues. Broadly speaking, we can say that Western culture is inherently individualistic (whether through natural persons or notion of the firm as a natural person) and tends to assert property rights through ownership; it also tends to view resources through a utilitarian ethic as opportunities for personal gain through some form of exploitation. Maori culture, by contrast, is group-based, emphasizing the collective good and views resources not simply as opportunities for exploitation but from an innate sense of oneness with the natural world, as entities deserving respect in their own right. The notion of collective responsibility for the guardianship of these natural resources is an accompanying strong cultural imperative.

Solomon (2000, p. 4) phrases the distinction as follows:

Acknowledging the spiritual dimension of their universe and respecting the mauri or central life force of every living thing was fundamentally important to the Maori world view. In other words, the reciprocity of obligations was balanced against the right to use and exploit. This can be contrasted with the notion of intellectual property rights which focuses on the economic right to exploit for profit and financial gain. The needs of the individual, and corporate legal personalities such as multinationals, are preferred to the collective good.

Under this capitalist model, resources are viewed entirely as a means of exploitation for economic gain. There is little or no reciprocity or respect for the integrity of the resources as living and breathing entities with their own mauri or life force.

A New Stakeholder

This raised an interesting question. We hold a model of data as information to be used for our purposes, which underpins the informational perspective of *whakapapa*. Does our model violate the Maori ethic relating to *whakapapa* as a sacred entity warranting respect and for whose respectful guardianship we have a collective responsibility? Thus, the notion of a data collection itself (here, the *whakapapa* is the collection considered) having an inherent integrity in its own right leads us to reconsider the notion of stakeholder. Not only the impacted individuals and groups but even the very information may be a stakeholder, if we adopt a Maori perspective (Floridi, 1998). Thus, including *whakapapa* in our stakeholder list and conducting subsequent analysis to determine whether the integrity of the whole collection is at risk by actions in the course of the project may assist in bridging the gulf between the two cultures. In this way, we may apply a tool based on Western value systems in a manner nonetheless

consistent with the worldview of a very different culture. The key is being able to adopt the Maori perspective when doing the analysis.

The inspection itself had identified as a concern issues to do with data integrity, in general, but had not done so through a perspective that animated in this way the data itself as a collective stakeholder. Thus, the Maori perspective appears capable of being represented through the SoDIS inspection process, but only if the analysts themselves adopt a similar and compatible (in this case, animist) mindset. This is a subtle but significant shift in thinking. Research question 1* is answered affirmatively. The SoDIS process is capable of identifying more general value issues.

2** is not resolved.

ANOTHER SENSE OF COLONIZATION AND THE SODIS WESTERN PRINCIPLES

Myers and Young (1997) have noted how an information system may act as a steering mechanism enlisted to serve the agenda of a power elite, and, in this way, the system itself may act as a colonizing device. While the term *information system* has been used by Myers and Young (1997), it equally might be substituted by a systematic process, a set of processes, or a software tool used in support of a process. One reason for the concern about cultural ethical colonization is the fact that the ethical relata in the SoDIS process were derived from a Western cultural-ethical perspective embodied in codes of professional ethics. A necessary condition in order for the process to avoid colonization is its application from a Maori perspective. But is that a sufficient condition, or does the SoDIS process' seeming foundation in a Western ethic mean that the use of the process in other cultures necessarily colonizes social and ethical values in this way?

Our initial questions were (1) Would the ethical connectives between task and stakeholder that had been derived from software codes of ethics and codes of practice be adequate to identify ethical risks in a manner that was sensitive to the indigenous culture in this bicultural context? and (2) Would the use of the SoDIS process colonize the Maori software application with Western cultural values?

Did the SoDIS Have These Threats — 1* Was it Adequate to the Task? and 2** Did it Necessarily Act to Colonize the Maori Culture?

a. SoDIS and Western Culture or Professionalism. The SoDIS process embodies standards of professionalism, which, at a certain level, are universal but may be sociologically defined as merely a set of norms. However, there does appear to be a common ethic inherent in the international community of practice (Wenger, 1998) constituted by professional software developers. Is this ethic to do their job well, sensitively, and in a manner that does not cause harm, a universal ethic of the profession?

b. The derivation of the connecting issues between tasks and stakeholders in the SoDIS analysis was driven by ethical imperatives common to software codes of conduct and codes ethics. A significant number of ethical imperatives were gathered from the various codes and were assigned as instances to one or another of Gert's (1988) moral rules. Some of the imperatives derived from professional codes did not fit clearly under any particular Gertian moral rule. These imperatives were related specifically to a moral approach to a project and did not have a stakeholder other than the project itself. These included issues like "require the developer to work on projects with infeasible goals." This process identified 32 common issues distributed among five distinct principles depicted in Table 2.

Table 2. Alignment of SoDIS principles with those of Gert

SoDIS Principles	**Gert's Original 10 Principles**
Project issues (no stakeholder focus)	none
Cause harm	Don't kill, Don't cause pain
Unreasonably restrict	Don't disable, Don't deprive of freedom
Involve deception	Don't deceive, Don't cheat
Conflict with your responsibility toward	Don't deprive of pleasure, Keep your promises, Do your duty, Obey the law

As some imperatives seemed to belong equally under two different Gertian rules, it was decided to collapse the rules under more general moral principles, so "Don't kill" and "Don't cause pain" were subsumed under the principle "Cause harm." The negative form of the principles was used so that when a question was answered with a *yes*, it would indicate an ethics violation. For example, affirming that "a particular task will cause harm to a stakeholder" indicates the need for remedial ethical action.

The resulting generic SoDIS principles and the relation to Gert's rules are in Table 2.

Some of the rights and obligations identified when following this method of categorization may be in conflict, and it will become necessary to prioritize how these rights are addressed and which of them can, within the bounds of the five previous principles, be addressed. This prioritization, of course, involves value decisions and a setting of priorities among values identified. As Rikys (1980, p. 26) has noted, these value-determined priority choices may create an opportunity for cultural exclusion, since "if planners are not exposed to or in possession of some values, these values will fail to be reflected in the planning policies and priorities which result from planning." One of the general ways to help prioritize the ethical obligations within a project is to determine what actions are necessary in order to satisfy the perceived obligation and to evaluate those actions in terms of whether they are morally required, morally wrong, or merely morally permissible. This approach to evaluating potential actions is a variation on Green's (1994) decision tree for assessing obligations.

Colonization of Social and Ethical Values

One of the research questions in general and in particular for a multicultural process is related to the derivation of the ethical issues relating the task and the stakeholder. This potential problem may be more likely because change of the moral rules used in the SoDIS may have an impact on the way stakeholders will be identified. It may seem that this change might be a cause of ethical colonization.

Justification of the Process as Non-Colonizing

There are three items that weigh against the charge that the SoDIS engages in ethical colonization. The first is that the underlying premise of the SoDIS is an analysis of an expanded set of stakeholders. The second is that the set of

principles and questions is expandable by the user and, thus, does not mandate the ethics of the analyst. Third, the ethical questions are answered from the analyst perspective and subject to the analyst's interpretation. In this regard, it may be the analyst who is guilty of ethical colonization rather than the SoDIS process. This risk normally is mitigated by conducting the analysis process jointly with stakeholders from the client organization or community. This ensures that participants are well-versed in the cultural context and subject domain.

No Ethical Colonization

First, the process expands rather than narrows who the stakeholders are so that it broadens the issues and is not colonizing. If the stakeholder expansion were consistent with the Western business ethics model that the primary stakeholders in any project are only those who have a financial interest (Mitchell et al., 1997), then enlarging the stakeholder base indeed would be colonizing. As we have seen, however, stakeholders are identified by the following three questions that address affected parties in their cultural circumstance:

- Whose behavior, daily routine, or work process will be affected by the development and delivery of this project?
- Whose circumstances, job, livelihood, or community will be affected by the development and delivery of this project?
- Whose experiences will be affected by the development and delivery of this product?

By stakeholder, we mean individuals or groups who may be affected directly or indirectly by the project and, thus, have a stake in the development activities. Those stakeholders who are affected negatively are particularly important regarding ethical sensitivity, because they are often the ones overlooked.

Negative effects include both overt harm and the denial or reduction of goods. So, obviously, the development of a medical software package that delivered erroneous dosages of medicine that killed patients would have a negative effect. But SoDIS analysts also would include software that limited people's freedom of expression as having a negative effect. Limitations on positive ethical values and rights are negative effects. It can also be argued that the failure to promote positive ethical values is also a negative effect. This is the model that was used in part for the project questions about tasks.

Therefore, we extend the traditional software project stakeholder list from customers and corporations or shareholders to include all those who will be affected by the software and by its production. This includes users of the software, families of the users, social institutions that may be altered radically by the introduction of the software, the natural environment, social communities, software professionals, employees of the development organization, and the development organization itself. If the analysis is done from a Maori perspective, those affected further include *whakapapa* and related social groupings.

Space precludes a fuller discussion here, but this inclusive motivation could be considered to be broadly consistent with the Habermasian (1984) desire to avoid distorted forms of communication, whereby dominant groups use systems as mechanisms to effect a "colonization of the lifeworld" (Myers & Young, 1997, p. 226) of other groups.

Second, the SoDIS is tailorable in two ways: (1) new questions can be asked and existing ones can be ignored and (2) the meaning of terms is conditioned by one's worldview. A basic assumption of the SoDIS process is that the original ethical relata are a starting point for the analysis. The SoDIS analyst should add appropriate ethical relata to the analysis. The ethical relata are about software development impacts.

Third, the questions are answered from the perspective and understanding of the analyst. The analysts answer questions from the perspective of the stakeholder. The effect of taking the stakeholder perspective can be seen by looking at some specific ethical issues and the guidance contained in the SoDIS process. For example, the principle, "Does task *cause harm* to the stakeholder?" includes the alteration of information (interpreted from a Maori perspective to include *whakapapa*) as a potential harm. This approach also facilitates the addition of context-relevant stakeholders. So, in the case of the Ngapuhi, the information is a stakeholder who is exposed to significant risks.

Under the principle *cause harm* are the ethical relata that follow.

Does the task in question:

- Allow unauthorized access or alteration of the data of the stakeholder?
- Violate the privacy and confidentiality of the stakeholder?
- Discriminate against the stakeholder?
- Fail to take into consideration the needs of the stakeholder?
- Involve the design or approval of software that may lower the quality of life of the stakeholder?
- Cause loss of information, loss of property, property damage, or environmental impacts that affect the stakeholder?

It is clear from the earlier discussion that these principles are broad enough when interpreted from a Maori cultural perspective to encompass most of their specific concerns with this project.

Even if the SoDIS process were to attempt to colonize, we have seen Maori resistance to possible colonization in their resistance to the computerization of land court records and other information (see the earlier section "The SoDIS Applied to this Project").

MAORI-SPECIFIC ETHICAL RELATA FROM THE SODIS INSPECTION

Of the issues identified in the SoDIS inspection (see the earlier section "The Results of the First Application"), several had specific significance from a Maori cultural perspective. While some of the concerns raised could be viewed as fairly relevant to any software development context, the Maori dimension often introduced further culture-specific elements.

In the first cluster of *client developer communication*, the key Maori ethical questions related to who could legitimately represent the *iwi* as the project client. Walker (1990) has noted historical situations in which the authority of the chiefs had been subverted and the authority structures of Maori society undermined by the colonizer. Thus, when engaging in research and development with Maori (Bishop, 1996), the processes of initiation and the need to work through due tribal and group decision making and authority structures are critical.

In the second cluster of *sensitive data*, several Maori ethical questions arose. Privacy concerns and mechanisms for obtaining consent for provision of data for genealogical research purposes raised complex questions of who could legitimately view what data. The collective ownership of *whakapapa* at different levels meant that group and individual access rights had to be negotiated. Individual data were personal, but *whanau* data were the property of the family group to decide, and *hapu* and *iwi* had their own interests and group decision-making processes in order to determine these rights. For instance, what rights would system administrators, data entry clerks, and Runanga management have to access or restrict access to this data? These policies and authorization protocols would need to be developed and agreed upon through accepted tribal decision-making processes. Similarly, protocols concerning display of cultural artifacts over the

Web or use of *whakapapa* information for commercial purposes (e.g., to defray expenses of the site or to support storage and research costs) would need to be agreed upon at the tribal level in order to offset concerns over commercialization and inappropriate use of treasured information and sacred objects. Data integrity and the need to preserve the very authenticity of *whakapapa* as a stakeholder in its own right has been noted as a key Maori concern.

In the third cluster, *user experience*, several more Maori ethical questions arose. Again, questions over authority in disputed circumstances would need to be settled (i.e., who could determine official groupings and their standing?). For instance, a particular Northern grouping, Ngati Hine, claims *iwi* status but has been deemed by Te Ohu Kai Moana as a *hapu*. How do such determinations hold standing, who decides official lists of *hapu* and *iwi*, and how are dissenting voices to be registered? Likewise, under what criteria are membership applications to be refused registration and what is the impact for those refused? What authority will systems administrators and Runanga clerks possess, and what controls will be in place to ensure the integrity of data entered and stored? How is the integrity of *whakapapa* to be maintained in each of these circumstances?

Questions related to access and performance arose, with concerns over slow Internet access speeds for rural users. For indigenous peoples who often live in rural areas, access issues present known problems. Burn and Loch (2001) have observed how the digital divide particularly impacts the rural poor and rural and central city minorities. In a relatively recent report (T.P.K., 2001, p. 7), it was found that "65 percent of Maori respondents reported they had never used the internet, and about 8 percent reported they used the internet on a daily basis…only 34% of Maori households possessed a computer."

Visual and language impairments were further concerns, with untranslated Maori language text a potential barrier for younger Maori, who would frequently be less fluent, and font selection an issue for older Maori.

Another *whakapapa*-related concern arose for those who were adopted (whether formally or informally) and how their data would be represented; for instance, would both natural and adoptive lineages be stored? Who would need to give consent for such a practice?

Thus, it can be seen that the SoDIS process has been able to identify concerns of particular relevance to Maori in this bicultural project. The input from Maori project participants to the SoDIS inspection seems to have resulted in an outcome whereby the Maori perspective has been incorporated successfully. This incorporation of the Maori perspective has been realized by working through the questions of the process derived from the general ethical principles identified in Table 2. While relatively generic, the questions have forced consideration of many issues critical to the stakeholders in this project. The detailed operation of the principles can be seen in the appendix to this article, which provides the full wording for each of the SoDIS principles considered relevant to Maori.

POTENTIAL LIMITATIONS OF THE SODIS IN THE BICULTURAL CONTEXT

Joint Governance Models

In her report to the New Zealand government on the issues related to online authentication and e-government, Kamira (2004) proposes joint governance models to operate in such large technology projects with potential negative impacts for Maori. The joint governance model is intended to enable power sharing between Maori and governmental agencies and to ensure that cultural differences are taken into account (i.e., the key importance

of face-to-face relationships in Maori culture stands to be lost in a computerized authentication process wherein access to government services could become more alien and inaccessible to Maori users of services).

Key notions underpinning a governance-*kaitiaki* (guardianship) model to operate at program and project levels are drawn from the Treaty of Waitangi, the founding partnership document between the two peoples, which include:

- Active participation and participation in decision making;
- Active protection of Maori interests, rights and taonga; and
- An ongoing right to development that is not locked into an 1840 "time closet." (Kamira, 2004, p. 35)

Thus, a general model of oversight is proposed in order to operate at a program level when an information-technology-based program of change is to be implemented in a bicultural context. While we believe that this is a sound approach to an engagement model recognizing difference and seeking partnership, we also believe that this is a difficult model for the dominant partner to adopt and to adhere to consistently.

While the SoDIS process specifically does not operate at this governance level, it can act in support of such structures by offering a mechanism for its operationalization within a project. In some cases, where a joint governance model has not been agreed upon, the SoDIS process even may compensate for its absence. For instance, the SoDIS process can act in concert with several of the following key factors indicated by Kamira (2004) for establishment of governance-*kaitiaki*:

- Align with the treaty (by ensuring joint participation in the analysis process);
- Encompass different cultural considerations;
- Protect the sensitivity of information;
- Ensure that ultimate accountability be to the intended beneficiaries of the project; and
- Cultivate trust and strong working relationships.

Professionalism and Culture

Multicultural discussions often take a "silo" approach to the description of diverse cultures, emphasizing what makes them different at the expense of complete accuracy. We have looked at the question of colonization with this approach in mind and have assumed that cultures are like a series of silos, each isolated and distinct from other cultures. Despite this presumption of difference, two such cultures, Maori and Western, seemed to interact in the application of the SoDIS inspection process to the development of a Maori software application.

The SoDIS principles are designed for the engagement of professionals in the development of software across all cultures. The SoDIS is applied within the silo in order to help the system developers to attend to the social and ethical risks within that silo. The SoDIS could be used as a tool in a paternalistic model of professionalism, where the analyst uses it to impose his or her cultural values on another; but the SoDIS was designed with a fiduciary model of professionalism in mind (Bayles, 1981). A fiduciary model of professionalism imposes high standards of development, but the resulting software is developed in partnership with the culture in which it works. Fiduciary means trust, which must be won by both partners in the endeavor.

The SoDIS provides a model for software inspection, and the values (i.e., acknowledging that *whakapapa* is a stakeholder) are derived from the perspective of the analyst(s) working with project stakeholders in the inspection team. In the Maori context, where models of group decision making based upon oral modes of consensus discussion

(*hui*) are common, it could be argued that the SoDIS process needs to incorporate opportunities for such group conversations, which is certainly an area for adapting the SoDIS process and worth exploring further.

CONCLUSION

Although the SoDIS inspection identified some risks unique to Maori and some more general risks, it did not identify all of the risks. SoDIS, in fact, is limited in that it is not designed to identify risks arising from the interaction of multiple tasks. The application of the SoDIS process, however, has provided several interesting lessons. The application of the SoDIS process did not colonize the TRAION software project, so we can say the following:

1. The use of a Western-based development process need not colonize an indigenous software project, even using several concepts of colonize. The positive result is the following:
2. The process was able to identify some Maori unique issues. The features that facilitated this positive result are as follows:
    - 2a. The underlying premise of the SoDIS is an analysis based on an expanded set of stakeholders.
    - 2b. The set of principles and questions is expandable by the user and, thus, does not mandate the ethics of the analyst.
    - 2c. The ethical questions are answered from the analyst's perspective and subject to the analyst's interpretation.

We believe that these lessons, gained in a bicultural context-developing software with NZ Maori, will be applicable to other indigenous peoples, especially those who have a history of colonization, such as Australian aboriginals, Native Americans, Peruvian Indians, the Inuit peoples, and so forth. In developing software for people with such differing worldviews, the careful use of software development impact statement (SoDIS) inspections may ameliorate the worst forms of re-colonization by Western values through software and may enable the development of software in respectful partnerships, applying techniques that enable deep cultural differences to surface and to be addressed specifically rather than being accorded mere lip service.

REFERENCES

ACM. (n.d.). *ACM code of ethics and professional conduct* (section 1.2). Retrieved from http://www.acm.org/constitution/code.html

Avison, D., & Myers, M. (1995). Information systems and anthropology: Aan anthropological perspective on IT and organizational culture. *Information Technology and People, 8*(3), 43-56.

Bayles, M. (1981). *Professional ethics.* Belmont, CA: Wadsworth.

Bishop, R. (1996). *Collaborative research stories; Whakawhanaunatanga*. Palmerston North: Dunmore Press.

Burn, J., & Loch, K. (2001). The societal impact of the World Wide Web — Key challenges for the 21st century. *Information Resources Management Journal, 14*(4), 4-14.

Clear, T., Charkova, R., Lin, A., & Lomax, T. (2004). Nga Iwi o Ngapuhi membership system: Relationship management and relational design. *NZ Journal of Applied Computing and IT, 8*(1), 8-15.

Courts, D. O. (1999). *Maori land court information management team report — Access to and archiving of Maori land court records after imaging.* Wellington: Maori Land Court.

Floridi, L. (1998). *Does information have a moral worth in itself?* Retrieved July 2005, from http://www.wolfson.ox.ac.uk/~floridi/cepe.htm

Gert, B. (1988). *Morality.* Oxford: Oxford University Press.

Gotterbarn, D. (1991). Computer ethics: Responsibility regained, national forum. *The Phi Kappa Phi Journal, 71*(3), 26-32.

Gotterbarn, D., Clear, T., & Kwan, C. (2004). *Managing software requirements risks with software development impact statements.* Paper presented at the 17th Annual NACCQ Conference, Christchurch, New Zealand. Retrieved from http://www.naccq.ac.nz/conference05/proceedings_04/gotterbarn.pdf

Gotterbarn, D., Miller, K., & Rogerson, S. (1999). Software engineering code of ethics. *IEEE-Computer, 30*(10), 84-88.

Gotterbarn, D. & Rogerson, S. (2005). Responsible risk analysis for software development: Creating the software development impact statement. *Communications of the Association for Information Systems, 15*(Article 40).

Green, R. M. (1994). *The ethical manager.* Englewood Cliffs, NJ: Macmillan Publishing.

Habermas, J. (1984). *The theory of communicative action: Reason and the rationalisation of society, Vol. 1.* (T. McCarthy, Trans.). Boston: Beacon Press.

Hofstede, G. (1980). *Culture's consequences: International differences in work related values.* Newbury Park, CA: Sage.

Jarvenpaa, S., & Leidner, D. (1998). Communication and trust in global virtual teams. *Journal of Computer Mediated Communication, 3*(4). Retrieved December 14, 2005, from http://jcmc.indiana.edu/vol3/issue4/jarvenpaa.html

Kamira, R. (2004). *Research of issues for Maori relating to the online authentication project — For state services commission* (Commission Report). Auckland: Paua Interface Ltd. For State Services Commission. Retrieved December 10, 2004, from http://www.e.govt.nz/docs/tikanga-200408/tikanga.pdf

King, M. (2003). *The penguin history of New Zealand.* Auckland: Penguin Books.

Mitchell, R. K., Agle, B. R., & Wood, D. J. (1997). Toward a theory of stakeholder identification and salience: Defining the principle of who or what really counts. *Academy of Management Review, 22*(4), 853-886.

Myers, M., & Young, L. (1997). Hidden agendas, power and managerial assumptions in information systems development. *Information Technology and People, 10*(3), 224-240.

Rikys, P. (1980). The case for Maori representation in regional planning. *Te Maori*, pp. 26-28.

Rogerson, S., & Gotterbarn, D. (1998). The ethics of software project management. In G. Collste (Ed.), *Ethics and information technology.* Delhi: New Academic Publisher.

SDRF. (2005). SoDIS 4.0 prototype location. Software Development Research Foundation. Retrieved from http://www.sdresearch.org

Smith, A. (1997). *Fishing with new nets; Maori Internet information resources and implications of the Internet for indigenous peoples.* Retrieved April 2, 2003, from http://www.isoc.org/isco/whatis/conferences/inet/97/proceedings/E1/E1_1.htm

Solomon, M. (2000). *IPR and IPRO: Intellectual property rights and indigenous peoples rights and obligations.* Paper presented at the Global Biodiversity Forum, Nairobi, Kenya. Retrieved June 26, 2004, from http://www.inmotionmagazine.com/ra01/ms2.html

T. P. K. (2001). *Maori access to information technology.* Wellington: Te Puni Kokiri - Ministry of

Maori Development. Retrieved June 24, 2004, from *http://www.tpk.govt.nz*

Tuhono. (2004). Tuhono helps Maori link to $bn fisheries asset. Retrieved June 8, 2004, from http://www.scoop.co.nz/mason/stories/PO0405/S00122.htm

Walker, R. (1990). *Ka whawhai tonu matou — Struggle without end*. Auckland: Penguin Books.

Wenger, E. (1998). *Communities of practice* (1st ed.). Cambridge: Cambridge University Press.

ENDNOTE

[1] SoDIS is a registered trademark of the Software Development Research Foundation (SDRF).

APPENDIX. TEXT OF SODIS PRINCIPLE GUIDANCE WITH PARTICULAR RELEVANCE TO MAORI

Principles with Particular Maori Relevance

CAUSE HARM TO

As a software project is undertaken, a central principle requires that the project and its product cause no harm, either direct or indirect. *Harm* means injury or negative consequences, such as undesirable loss of information, loss of property, property damage, or unwanted environmental impacts. This principle prohibits use of computing technology in ways that result in harm to users, the general public, employees, or employers. Harmful actions include intentional destruction or modification of files and programs leading to serious loss of resources or unnecessary expenditure of human resources, such as the time and effort required to purge systems of computer viruses. As the Association of Computing Machinery (ACM) puts it, "Well-intended actions, including those that accomplish assigned duties, may lead to harm unexpectedly. In such an event the responsible person or persons are obligated to undo or mitigate the negative consequences as much as possible. One way to avoid unintentional harm is to carefully consider potential impacts on all those affected by decisions made during design and implementation" (ACM).

CAUSE Loss of Information, Loss of Property, Property Damage, or Environmental Impacts

Software can harm property and the environment directly and indirectly. Examples might include software that regulates chemicals or machine control software and financial systems. The task should be evaluated in light of the current stakeholder's interests.

INVOLVE the Design or Approval of Software that Will Not Lower the Quality of Life

Software first should not harm and, if possible, should improve the quality of life. Each task should be evaluated to determine the effect it will have on the quality of life of each relevant stakeholder. Examples of diminished quality of life could include applications that cause repetitive strain injuries, the inability of customers to return unwanted or damaged merchandise, or software-controlled machinery that is excessively noisy.

FAIL to Take into Consideration the Needs of

Software tasks, projects, and products often are completed without considering the needs of certain stakeholders. The omission of these considerations can lead to mediocre software, disgruntled stakeholders, and, in the worst case, unacceptable products.

DISCRIMINATE Against

Codified, systematic discrimination can damage entire demographic groups and, in the worst cases, result in class-action litigation. Health insurance or life insurance eligibility software, for example, inadvertently can discriminate against persons from certain economic backgrounds. Single-language automated teller machines can prevent groups from getting access to their accounts. Each task should be evaluated to determine whether its outcome might discriminate against the current stakeholder.

VIOLATE the Privacy and Confidentiality of

Could the current task compromise the privacy or confidentiality of the current stakeholder? If so, what steps must be taken to ensure that the stakeholder's needs will be considered and resolved?

ALLOW Unauthorized Access or Alteration to the Data

Could the current task compromise the information security of the current stakeholder? If so, what steps must be taken to ensure that the stakeholder's needs will be considered and resolved?

This work was previously published in the International Journal of Technology and Human Interaction, edited by B. Stahl, Volume 2, Issue 2, pp. 1-23, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 5.24
Issues, Limitations, and Opportunities in Cross-Cultural Research on Collaborative Software in Information Systems

Dongsong Zhang
University of Maryland, Baltimore County, USA

Paul Benjamin Lowry
Brigham Young University, USA

ABSTRACT

Globalization has led to the increasing use of organizational teams comprising individuals with diverse cultural backgrounds. Existing research suggests that collaborative software may benefit multicultural teams. However, most prior studies are limited by their focus on U.S. and Western cultures. We explore this issue by comprehensively examining the literature on cultural effects on collaborative software use. This article makes several contributions by providing common nomenclatures and theoretical perspectives that are essential to promoting scientific progress in this area. It focuses mainly on empirical collaborative software studies in which culture is a key conceptual construct. We discuss underlying cultural theories, research methodologies, and findings of major collaborative software studies on the impact of culture. This article provides insights into various issues surrounding this line of research and highlights future research opportunities.

INTRODUCTION

Globalization has affected business by increasing the competitiveness of the marketplace, restructuring organizational boundaries, and creating

new challenges for managers who deal with multinational companies or international alliances. Businesses often use multicultural collaborative groups working in distributed environments to cope with uncertainty, change, ambiguous problem definitions, and rapidly changing information (e.g., Vick, 1998). Hence, a critical need exists for managers "to develop a new repertoire of skills and abilities to manage and/or work with people whose cultures and value systems can be significantly different from those at home" (Tung, 1995, p. 485).

Improving group processes and outcomes has been one of the most highly investigated research issues of the past two decades. The advance of information technologies makes it possible for distributed teams to be supported through collaborative technologies such as group support systems (GSS) and computer-mediated communication (CMC), which are collectively known as collaborative software (CSW). CSW refers to computer systems that combine communication and decision-support technologies to facilitate the formulation and execution of various group activities.

Culture is central to how people experience their world, make sense of concepts, express themselves, and make decisions. Information technology is not "culturally neutral and may come to symbolize a host of different values driven by underlying assumptions and their meaning, use, and consequences" (Leidner & Kayworth, 2006, p. 359). First, beliefs and values shared by the members of a group affect group behavior in a variety of ways that can either accelerate or retard the implementation of technological changes (Veiga, Floyd, & Dechant, 2001). For example, many management and organization practices developed in Western countries are viewed with suspicion and often fail when introduced into other cultures (Kim, Park, & Suzuki, 1990). Likewise, when group tasks involve cross-cultural teams, cultural conflicts can arise. Since CSW design is typically based on Western cultural values, adopting CSW successfully in another culture may require both technical and social modifications (Watson, Ho, & Raman, 1994).

Second, the manner in which CSW may change group behavior is likely dependent on culture (Tan, Watson, Wei, Raman, & Kerola, 1993). Samarah, Paul, Mykytyn, and Seetharaman (2003) find that cultural diversity has a significant, positive moderating effect on group agreement and perceived decision quality when using CSW. To better understand how CSW can be successfully applied to a variety of cultures, researchers need to systematically compare the effects of CSW across different cultures (Tan, Watson, & Wei, 1995). Yet, only a small number of existing studies have empirically and theoretically examined cultural effects.

To advance this knowledge, we review and critique existing empirical research that specifically addresses cultural effects on CSW-supported group decision making. This article is different from recent literature reviews that focus on relationships between information technology and culture in general (Gallivan & Srite, 2005; Leidner & Kayworth, 2006). We attempt to focus primarily on empirical studies of multicultural, CSW-supported group work, which is critical for testing and validating propositions and hypotheses, and for developing theories about cultural influence in collaborative groups. We hope that our attempts at assimilation and analysis of existing studies will stimulate further research along this line.

The scarcity of literature in this area makes meta-analysis infeasible. For each specific research question/issue, there are usually only two to three studies. Most prior studies had very small sample size (effect sizes are typically small). Thus, our review and discussion are offered from a descriptive and critical perspective that aims to provide a roadmap for researchers. In addition, we focus only on empirical studies that include: (1) participants from different cultures, (2) the use

of certain CSW in face-to-face and/or distributed settings, and (3) culture as a key conceptual construct. Our coverage includes papers published before 2005 in journals and conference proceedings in information systems (IS) and management fields (see Table 1). During the search, we used a variety of searching terms such as cross-culture groups, culture and group collaboration, virtual teams, and virtual community.

In the remainder of the article, we will discuss underlying cultural theories, research methodologies, and findings of the studies we reviewed. We will then provide insights into the limitations of existing studies and highlight some directions for future research.

CULTURE AND COLLABORATIVE SOFTWARE

In general, CSW has been proven useful in alleviating problems associated with intercultural communication primarily by reducing many behaviors that might offend members of other cultures (Gray & Olfman, 1989; Aiken, Martin, Shirani, & Singleton, 1994). However, theories relating to group and organizational behavior and other disciplines are dependent on the culture being studied. Because each culture has unique beliefs, values, and norms, considerable differences may exist in how technologies are used and diffused across cultures (De Vreede, Jones, & Mgaya, 1999).

Table 1. Literature review coverage

Journals	Academy of Management Journal
	Academy of Management Review
	Communications of the ACM
	Decision Sciences
	Decision Support Systems
	Group Decision and Negotiation
	Information and Management
	Information Systems Research
	Journal of Global Information Management
	Journal of Management Information Systems
	Management Science
	MIS Quarterly
	Small Group Research
Conferences	International Conference on Information Systems (ICIS)
	Hawaii International Conference on System Sciences (HICSS)
	Americas Conference on Information Systems (AMCIS)
Digital Libraries	ACM digital library
	IEEE digital library

Since the majority of CSW studies on group performance have been conducted in the United States and other countries with Western cultures, culture is largely neglected or assumed to be irrelevant (Fjermestad & Hiltz, 1999). In an earlier review of 230 CSW studies (Fjermestad & Hiltz, 1999), only nine were found that included culture as either an independent variable or a moderator. Few studies both advance theories involving culture and employ empirical data to test assumptions and hypotheses. Consequently, the benefits of CSW identified mostly from research on Western cultures may not be manifest in different cultures under the same circumstances. For example, Kim et al. (1990) report that some incentives used to motivate North American workers can be counterproductive in collectivistic cultures.

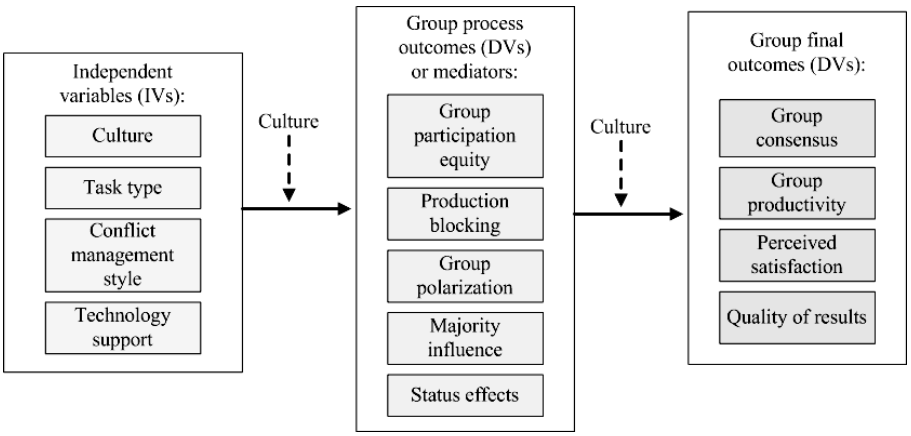
This article discusses CSW studies on cultural influence from several perspectives (see Figure 1). Those studies primarily examine the following independent variables: culture, task type, conflict management style, and technology support. We examine the effects of cultural differences on group processes (including group participation equity, production blocking, group polarization, status effects, and majority influence) and outcomes (including group consensus, perceived satisfaction, group productivity, and quality of results). Culture can also be used as a moderator while studying group process and final group outcome (Samarah et al., 2003). We will discuss the reviewed studies in terms of independent variables (IVs), dependent variables (DVs), and major findings and limitations.

Independent Variables

Culture

Individuals are socially conditioned by their culture (Zakour, 2004). Cultural differences in different nations, religions, ethnic groups, and societies are stable over time and often have existed for centuries. Kluckhohn (1962) states that culture "consists of patterns, explicit and implicit, of and for behavior acquired and transmitted by symbols, constituting the distinctive achievement of human groups, including their embodiments in artifacts" (p. 73). Hofstede (1991) defines culture as the "collective programming of the mind which distinguishes the members of one group or category of people from another" (p. 5). Groeschl

Figure 1. An overview of independent and dependent variables used in CSW studies



and Doherty (2000) assert that "culture consists of several elements of which some are implicit and others are explicit. Most often these elements are explained by terms such as behavior, values, norms, and basic assumptions" (p. 14). Based on previous culture literature, we define culture as a system of implicit and explicit beliefs, values, norms, preferences, and behaviors that are stable over time, are held in common by a group of people, and that distinguish one group from others.

The majority of cultural theories we studied focus on group value orientations such as value dimensions of national culture or the competing values framework at the organizational level

Table 2. Hofstede's cultural dimension model (Hofstede, 1991, p. 28)

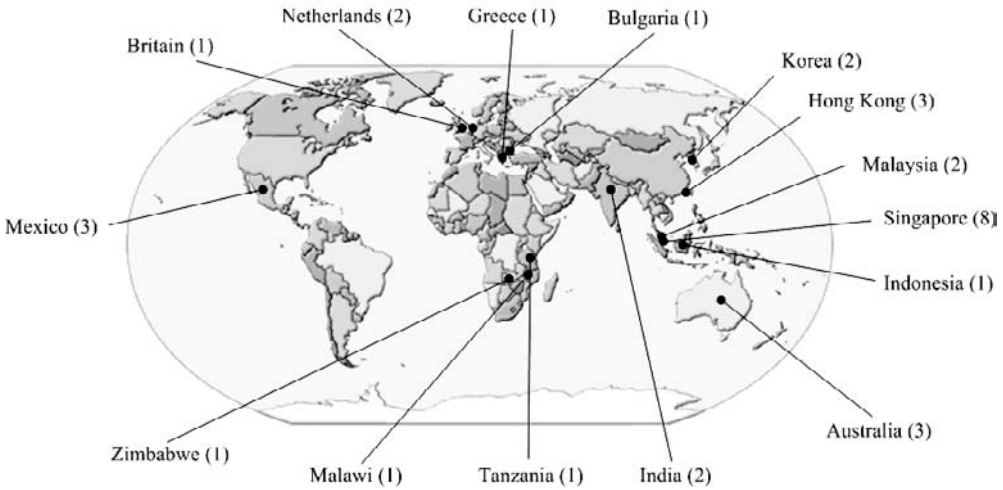
Cultural Dimension	Definition	Examples
Power Distance	***Power distance*** is the extent to which the less powerful members of institutions and organizations within a country expect and accept that power is distributed unequally.	**Low:** U.S. and Canada **High:** Japan and Singapore
Individualism and Collectivism	***Individualism*** describes cultures in which the ties between individuals are loose. ***Collectivism*** describes cultures in which people are integrated into strong, cohesive groups that protect individuals in exchange for unquestioning loyalty.	**Individualistic:** U.S., Australia, and Great Britain **Collectivistic:** Singapore, Hong Kong, and Mexico
Masculinity-Femininity	***Masculinity*** pertains to cultures in which social gender roles are clearly distinct. ***Femininity*** describes cultures in which social gender roles overlap.	**Masculinity:** Japan, Austria, and Italy **Femininity:** Sweden, Norway, and The Netherlands
Uncertainty Avoidance (UAI)	***Uncertainty avoidance*** is the extent to which the members of a culture feel threatened by uncertain or unknown situations.	**Low:** Singapore, Jamaica, and Denmark **High:** Greece, Portugal, and Japan
Confucian Dynamism	***Confucian dynamism*** denotes the time orientation of a culture, defined as a continuum with long-term and short-term orientations as its two poles.	**Long-term:** China and Japan **Short-term:** U.S. and Canada

(Leidner & Kayworth, 2006). Although many culture theories have been developed (Triandis, 1972; Hofstede, 1984; Hall & Hall, 1990), most CSW studies that focus on the impacts of culture rely on Hofstede's (1991) model of national culture, which was developed based on a large body of survey data about the values held by people in more than 50 countries. The participants in Hofstede's study were employees in local subsidiaries of IBM. His model defines five generalizable cultural dimensions based on value orientations that are considered important and are shared across cultures: power distance, individualism and collectivism, masculinity-femininity, uncertainty avoidance (UAI), and Confucian dynamism. Table 2 summarizes these cultural dimensions.

There have been several debates about and criticisms of Hofstede's cultural model. For example, the model is criticized on the grounds that it is rather crude and simplistic, that a survey may not be a suitable way of measuring cultural differences, and that a study of the subsidiaries of one company may not provide information about entire national cultures (Walsham, 2002). Despite these claims, Hofstede's model has been widely validated by theoretical and empirical evidence (Usunier, 1998). Myers and Tan (2002) examined 36 studies on culture in the IS literature. Among them, 24 used Hofstede's dimensions. Although Hofstede's model has various limitations, the general cultural constructs or dimensions appear to be useful for explaining potential differences in culture regarding the use of technology (Vogel, Davison, & Shroff, 2001a). In our literature review, almost every empirical study adopted some of the cultural dimensions of Hofstede's model.

All of the studies we reviewed involved at least one culture other than U.S. culture, as summarized in Appendix A. Figure 2 graphically depicts the non-U.S. countries examined in those studies. As shown, the majority of existing cultural research on CSW involves Asian and European countries, the U.S., and Australia. Only one study involves Africa (De Vreede et al., 1999), and none focuses on Central or South America. The lack of research on Africa, South America, and Central America may be attributable to the relatively rare adoption and use of CSW in those regions due to economic, political, and technological barriers.

Figure 2. Countries examined by existing studies



So far, IS researchers have not developed models or theories for the cultural effects on CSW-supported group work. Some of the empirical studies we surveyed failed entirely to base their hypotheses on any previous cultural model (e.g., Aiken, Hwang, Magalhaes, & Jeanette, 1993; Daily, Whatley, Ash, & Steiner, 1996; Daily & Steiner, 1998).

Although Hofstede's model is the dominant cultural model used in existing IS studies, the individualism-collectivism and power-distance dimensions of his model have received more attention in IS literature than have other dimensions (Leidner & Kayworth, 2006). These two dimensions delineate stark differences between cultures and can be cleanly operationalized for observations between cultures in a controlled setting. Watson et al. (1994) assert that individualistic groups are more likely to embrace the social structures of CSW because those structures conform with their individualistic culture; collectivistic groups are less likely to adopt CSW because the social structures of CSW conflict with their collectivistic nature. Triandis (1989) suggests that researchers develop hypotheses concerning the relationship between culture and social behavior based on the individualism-collectivism dimension. Conversely, the cultural dimensions of masculinity-femininity, UAI, and Confucian dynamism in Hofstede's cultural model have been rarely adopted in cross-cultural CSW research for various reasons. UAI, for example, can only explain Western cultures because a cultural bias toward "truth" is not as respected in Eastern cultures (Hofstede, 1991). Although various cultures examined in a study may differ in several cultural dimensions, the dependent variable(s) may be less influenced by one dimension than another. For example, in a study that examines the effect of majority influence in culturally heterogeneous and/or homogeneous groups, if student subjects with equivalent status are used, then the variance in the individualism-collectivism dimension would have a much larger impact on majority influence than would the difference in power distance. Power distance should, of course, be considered only when there is a hierarchical or non-egalitarian structure among participants. Therefore, while doing cross-culture research, researchers should identify which cultural dimensions would most likely play a role in the treatment of the IT artifact.

Technology Support

Cultural studies on CSW often use technology support as an IV, concentrating on how group processes and outcomes differ with and without CSW support (Ho, Raman, & Watson, 1989; Watson et al., 1994; Daily et al., 1996; El-Shinnawy & Vinze, 1997; Mejias, Vogel, & Shepherd, 1997b; Daily & Steiner, 1998; Tan, Wei, Watson, Clapper, & McLean, 1998a; Quaddus & Tung, 2002; Tung & Quaddus, 2002; Reinig & Mejias, 2003). Those studies have mainly compared three work modes: (1) traditional face-to-face groups without CSW support; (2) face-to-face (FtF), synchronous groups supported by CSW; and (3) distributed, anonymous groups supported by CSW. To our best knowledge, very few CSW studies have examined culture in distributed, asynchronous, and CSW-supported groups, distributed and identified groups, or distributed, culturally heterogeneous groups, mainly because of the great technical and logistic challenges in carrying out empirical studies in those environments.

Different communication media provide different levels of information cues (Daft & Lengel, 1986), social pressure, and immediacy of feedback (i.e., synchronous vs. asynchronous communication). For example, in traditional face-to-face communication, communication partners see and hear each other and have to respond to others immediately, while in a distributed setting, communication partners will have fewer cues but more time for preparing and rehearsing their responses. In addition, direct confrontation between group members which often occurs in

the face-to-face environment is reduced via the use of CSW. Given previous research on the topic, we argue that the cultural influence on group process may vary according to different amounts of social presence afforded by various types of communication media.

Social presence is "the degree to which a medium facilitates awareness of the other person and interpersonal relationships during the interaction" (Fulk, Schmitz, & Steinfield, 1990). According to social presence theory (Short, Williams, & Christie, 1976; Miranda & Saunders, 2003), face-to-face communication typically offers the highest level of social presence, while CSW-supported communication is typically low in social presence. In a study conducted by Zhang, Lowry, Zhou, and Fu (2007), the impact of culture on majority influence was manifested differently in three various communication settings. Because different communication work modes affect the degree to which culture impacts communication, work modes should be examined separately in culture studies.

Several studies use the anonymity feature of CSW as an IV to examine its effect in different cultures (Atkinson & Pervan, 1998; Mejias et al., 1997b). Anonymity works better in an individualistic culture than in a collectivistic culture, because individualistic cultures encourage free exchange of ideas, egalitarian decisions, and creativity (El-Shinnawy & Vinze, 1997; Tung & Quaddus, 2002). Collectivists are accustomed to conforming and restricting their ideas by following the group majority or the group leader's preferences—even when using CSW. Thus, the use of the anonymity feature may induce more conservative decision making and reduce participation equity in a collectivistic culture.

Task Type

Groups primarily exist to complete collaborative tasks, and the choice of group tasks in collaboration research is critical because it may account for 50% of total group performance (Poole, Seibold, & McPhee, 1985).

Most CSW studies use McGrath's task circumplex (1984). This topology proposes four general types of processes: generating alternatives, choosing alternatives, negotiating, and

Table 3. McGrath's task circumplex

Task Category	Task Type	Task Description	Used in Previous Studies
Generate	Type 1: Planning tasks	Generating action-oriented plans (e.g., problem-solving tasks)	None
	Type 2: Creativity tasks	Generating ideas (e.g., brainstorming tasks and idea generation)	(Aiken et al., 1993; Aiken, Kim, Hwang, & Lu, 1995; Atkinson & Pervan, 1998; Daily & Steiner, 1998; Daily et al., 1996; De Vreede et al., 1999; Mejias, Shepherd, Vogel, & Lazaneo, 1997a; Mejias, Vogel, & Shepherd, 1997b; Quaddus & Tung, 2002; Tung & Quaddus, 2002)

continued on following page

Table 3. continued

Task Category	Task Type	Task Description	Used in Previous Studies
Choose	Type 3: Intellective tasks	Solving problems with a correct answer; such tasks have a demonstrable right answer, and the group task is to invent/ select/compute that correct answer	(Tan et al., 1998a; Vogel et al., 2001a, 2001b)
	Type 4: Decision-making tasks or preference tasks	Dealing with tasks for which the preferred or agreed upon answer is the correct one; tasks for which a demonstrably correct answer does not exist and for which the group's task is to select, by some consensus, a preferred alternative (e.g., tasks used in choice shift, and polarization studies; mock juries)	(De Vreede et al., 1999; El-Shinnawy & Vinze, 1997; Griffith, 1998; Ho, Raman, & Watson, 1989; Morales, Moriera, & Vogel, 1995; Quaddus & Tung, 2002; Reinig & Mejias, 2003; Souren, Priya, Imad, & Mykytyn, 2004; Tan et al., 1998a; Tan, Wei, Watson, & Walczuch, 1998b; Tung & Quaddus, 2002; Watson et al., 1994)
Negotiate	Type 5: Cognitive-conflict tasks	Resolving conflicts of viewpoint; tasks where members of the group do not just have different preferences but have systematically different preference structures (e.g., some jury tasks)	None
	Type 6: Mixed-motive tasks (resolving conflicts of interest)	Resolving conflicts of motive-interest (e.g., negotiation and bargaining tasks, mixed-motive dilemma tasks)	None
Execute	Type 7: Contests/battles/ competitive tasks	Resolving conflicts of power; competing for victory (e.g., wars, winner-take-all conflicts, competitive sports)	None
	Type 8: Performances/ psycho-motor tasks	Executing performance tasks; psychomotor tasks performed against objective or absolute standards of excellence (e.g., many physical tasks)	None

executing, which are further divided into eight types of tasks (see Table 3).

As shown in Table 3, creativity and preference tasks are the most popular ones selected, followed by intellectual tasks. The other five task types have never been used in previous research. Among the reviewed studies, four used task type as an IV (Tan et al., 1998a, 1998b; Quaddus & Tung,

2002; Tung & Quaddus, 2002). The preliminary findings suggest that group members with diverse cultures behave differently while performing different tasks.

It is interesting to note that although tasks in *negotiate* and *execute* categories are common in real life, they have never been used in previous studies. We speculate that because most participants in those studies were university students, they lacked sufficient knowledge and experience with such tasks. As a result, those types of tasks may be inappropriate for university students, but should be used in field studies.

Conflict Management Style

Conflict refers to situations in which group members believe their needs cannot be mutually satisfied, reconciled, or integrated. Conflict among group members can emerge for various reasons, including a group's cultural composition (Reinig & Mejias, 2003). Different ways of handling internal group conflict are termed "conflict management styles" (Souren et al., 2004). They include avoidance, accommodation, competition, collaboration, and compromise (Rahim, 1983).

Conflict management styles often vary by culture. Hofstede's dimensions of power distance, individualism-collectivism, and UAI can help explain how conflict is managed differently in different cultures. First, in low-power-distance cultures, people believe in the legitimate use of power and equal rights; when making group decisions, members' views are more likely to be judged according to their merits rather than by the status of their contributors. In high-power-distance cultures, low-status individuals are usually dependent on and respectful of high-status individuals; they are more likely to accept the views of higher-status members. Therefore, we propose that in cultures with higher power distance (e.g., Singapore, Japan, and China), conflicts between low-status and high-status members in a group are less likely to occur, and even if they do occur, they will be more easily resolved than those between members of low-power-distance cultures.

Second, members of individualistic cultures are expected to look after themselves and their immediate families; speaking one's mind is a virtue; self-actualization and individual achievements are valued. Thus, individualists tend to resolve conflict via open and direct communication. In collectivistic groups, relationships prevail over tasks. Such groups tend to employ indirect means for conflict resolution since the maintenance of harmony and the ability to forge consensus are highly regarded. As a result, collectivists are more inclined to reach consensus and resolve conflicts by following majority views than are individualists (Tan et al., 1998a; Zhang et al., 2007).

Third, low-UAI cultures are characterized by people who are comfortable in ambiguous situations, and tolerant of dissent and deviance (Hofstede, 2001; Tan et al., 1995). In high-UAI cultures, people fear ambiguous situations; groups are more likely to suppress deviant ideas and behavior and to resist innovation. As a result, conflicts are more easily resolved in high-UAI groups.

Some studies have shown how culture affects conflict management styles. Samarah et al. (2003) investigated whether the cultural heterogeneity of a group influences the styles of conflict management adopted by group members. Their study used a fuzzy task (Campbell, 1988) to examine the behavior of U.S. homogeneous, Indian homogeneous, and U.S.-Indian culturally heterogeneous groups. Results showed that group heterogeneity, as well as the interaction between collaborative conflict management style and cultural diversity, had positive moderating effects on perceived decision quality and the degree of group management. Another study (Souren et al., 2004) used conflict management style as an IV to examine its effect on group performance using homogeneous and heterogeneous groups consisting of U.S. and Indian members. The study found that the culturally heterogeneous groups had a lower level of collaborative conflict man-

agement style than did homogeneous groups. In addition, the collaborative conflict management style positively influenced user satisfaction with the decision-making process, perceived decision quality, and perceived participation.

Dependent Variables

We discuss DVs in two categories: (1) group *process* variables that assess group process gains or losses, and (2) group *outcome* variables that evaluate the final results of group interaction.

Process: Group Participation Equity

Group participation equity is defined as equal and effective group participation (Steiner, 1972). Barriers to group participation equity often result from cultural and social norms (Steiner, 1972; Hofstede, 1984; Robichaux & Cooper, 1998). Because CSW encourages participation through anonymity and simultaneity, participation equality is often greater in groups supported by CSW than those without support (Reinig & Mejias, 2003).

Mejias et al. (1997b) compared group productivity levels and perceived participation equity in U.S. (individualistic) and Mexican (collectivistic) groups. U.S. groups generated more comments per participant and more unique ideas per group than their Mexican counterparts, even when participants were not anonymous; U.S. groups reported no differences in perceived participation equity between CSW and face-to-face groups, whereas Mexican CSW groups reported higher perceived participation equity than Mexican face-to-face, non-software-supported groups did; Mexican CSW groups also perceived higher levels of participation equity than their U.S. counterparts did. One possible explanation for these results is that CSW might encourage equal participation by reducing the dominance of some members, particularly in collectivistic groups.

Researchers also compare the perceived levels of equal participation between U.S. (individualistic) groups and Asian (collectivistic) groups. In Ho et al. (1989), the use of CSW led to more equal member influence in both U.S. groups and Singaporean groups. The anonymity feature of a group decision support system allowed dominant members in Singaporean groups to openly express negative opinions about other group members' contributions, a behavior that would otherwise be culturally unacceptable. Reinig and Mejias (2003) reported that participants from both the United States and Hong Kong supported by CSW perceived less dominance than did traditional face-to-face participants. Hong Kong participants reported less dominance in both face-to-face and CSW treatments than U.S. participants reported.

These findings conform to the view that the use of CSW, by alleviating status effect and avoiding direct confrontation, should encourage equal participation and reduce individual dominance. In the American culture, openness and directness in communication are often considered a virtue. In contrast, people from Singapore or Hong Kong (who are of high power distance and low individualism) tend to be modest and nonconfrontational with others through direct, open communication. On the one hand, the use of CSW can encourage participation particularly from those low-status group members from cultures with high power distance and low individualism. On the other hand, high-status members in such cultures may be unhappy because they feel they lose the power and influence over other group members because of CSW, which poses challenges to their authority and changes traditional social norms. As a result, those high-status members may be uncomfortable with or even resist the adoption and use of CSW in support of group tasks.

Process: Production Blocking

Production blocking refers to productivity loss in brainstorming groups when group members must take turns to express their ideas. This interferes

with idea generation in two possible ways. First, it disrupts the generation of ideas when delays are relatively long. Second, it reduces the flexibility of idea generation when delays are unpredictable. Because CSW enables parallel input from group members, it can significantly reduce production blocking that is common in face-to-face groups without CSW support.

Three studies have explicitly used production blocking as a DV, either examining whether the use of CSW reduces production blocking or comparing levels of production blocking between Asian and U.S. groups (Aiken et al., 1993, 1995; Reinig & Mejias, 2003). They all found that CSW reduced production blocking in both U.S. and Asian groups. Non-Western participants supported by CSW often experience significantly more production blocking than Western users of CSW did (Chung & Adams, 1997; Reinig & Mejias, 2003). This is likely because collectivism influences CSW users to contribute in a slower, more reserved manner. For example, in the study conducted by Reinig and Mejias (2003), the U.S. and Hong Kong participants were asked to assume the role of members of a "Community Resource Allocation Task Force" and rank nine projects that would be most deserving of money donated by local corporations. The perceived production blocking was measured by three survey questions. GSS participants from both the U.S. and Hong Kong reported less production blocking than those in non-GSS groups reported, indicating that the features of anonymity and parallel communication were successful in reducing process losses. In addition, Hong Kong participants reported more production blocking across both face-to-face and GSS treatments than did U.S. participants, which might be attributable to their cultural difference as collectivists—Hong Kong participants had a higher tendency to be modest and listen to others first, causing delays in expressing their ideas and increasing production blocking.

Process: Group Polarization

Group polarization is the tendency of individuals in a group to engage in more extreme decisions than their original individual inclinations (Moscovici & Zavalloni, 1969). CSW may alter group polarization because it allows people to participate in group discussion with reduced social presence in comparison to face-to-face verbal communication. To date, only one study has examined the impact of technology and culture on group polarization (El-Shinnawy & Vinze, 1997). That study used persuasive arguments theory (PAT) (Pruitt, 1971) to study group behavior in a CMC setting and in a face-to-face, non-CMC setting that included two cultures—the U.S. and Singaporean. The power-distance and individualism-collectivism dimensions of Hofstede's model were used as the theoretical basis. It was found that Singaporean groups polarized in a riskier direction, whereas U.S. groups polarized in a more cautious direction.

Process: Status Effects

Status effects occur when high-status members negatively dominate or marginalize the contributions of low-status members (Berger, Fisek, Norman, & Zelditch, 1977). Cultures that emphasize the status or power differences among members can increase evaluation apprehension and conformance pressure. Cultural norms dictate that critical remarks should be avoided in order to steer clear of conflict (Robichaux & Cooper, 1998). In high-power-distance cultures, status differences among individuals are prominent; individuals with higher status are powerful and exude excessive influence during group communication; people strive to maintain harmony; and relationship concerns tend to prevail over task concerns (Earley, 1994). Therefore, status influence is likely to be strong in high-power-distance cultures. In low-power-distance cultures,

status differences among individuals are less significant and people believe in equal rights, and high-status individuals do not exercise excessive influence during group communication. Hence, status influence is likely weak.

Only one cultural study on CSW has examined status effects (Tan et al., 1998b). That study indicated that the power-distance and individualism-collectivism dimensions of Hofstede's model were relevant to status effects. It examined status influence, sustained influence, and perceived influence. *Status influence* is the extent to which low-status individuals defer to the opinions of high-status individuals during group communication. *Sustained influence* is the amount of status influence remaining after group communication when high-status individuals are no longer present. *Perceived influence* is the amount of status influence that low-status individuals perceive during group communication. The study used Singaporean and U.S. groups and two different tasks (intellective vs. preference tasks). Results showed that the task type and communication medium (face-to-face verbal communication vs. CMC) had significant main effect on both status influence and perceived influence, whereas national culture had nearly significant main effect. National culture, task type, and communication medium had significant effects on sustained influence. Status influence and sustained influence were higher in preference task groups than in intellectual task groups. CMC was able to reduce status effects in both U.S. and Singaporean CSW groups.

Process: Majority Influence

Majority influence is the attempt by a majority of members in a group to impose their common position upon group dissenters during a decision-making process (Levine & Russo, 1987). Individualism-collectivism in Hofstede's model is a pertinent factor to be considered in cultural research on CSW and majority influence. In individualistic cultures, when people disagree with a majority position, they are likely to contradict it. In collectivistic cultures, people are integrated into strongly cohesive groups and base their self-understanding on the reactions of others. When CSW replaces verbal and visual communication, group majorities may exercise less normative influence on minorities (Ridgeway, Berger, & Smith, 1985). Several empirical studies conducted in North America have examined the effects of CSW on majority influence (e.g., Zigurs, Poole, & DeSanctis, 1988; Connolly, Jessup, & Valacich, 1990; Jessup, Connolly, & Galegher, 1990; Clapper, McLean, & Watson, 1991; Gallupe, Bastianutti, & Cooper, 1991). Some argue that minority members are more likely to oppose a majority viewpoint when they use CSW, especially when working anonymously (Dennis, Hilmer, & Taylor, 1998). One possible explanation, according to the Media Richness Theory (MRT) (Daft & Lengel, 1986), is that CSW is a lean medium that results in lower levels of social presence and conformance pressure than experienced in traditional face-to-face communication.

One study investigated majority influence in U.S. groups vs. Singaporean groups using two different tasks in three different communication settings (Tan et al., 1998a). The results showed that the impact of CMC on majority influence was contingent upon national culture. In the U.S. groups, participants challenged the majority position in the CMC settings more often than they did in the face-to-face, unsupported setting. In Singaporean groups, participants using CMC were less willing to challenge the majority position, and majority influence remained unchanged when CMC replaced verbal and visual communication—indicating that CMC may not be able to reduce conformance intention of collectivistic members. The results also showed that the impact of CMC on majority influence was not moderated by the type of group task.

Outcome: Consensus

Consensus refers to the achievement of group solidarity in decision making. Collectivistic cultures are more oriented toward consensus and less tolerant of conflict and discord (Triandis et al., 1994) than individualistic cultures are, which embrace open conflicts.

Research results vary as to the relationship between CSW use and different levels of group consensus in various cultures. In Reinig and Mejias (2003), Hong Kong CSW groups had significantly higher levels of initial consensus than their U.S. counterparts; however, U.S. CSW groups achieved greater change in consensus than Hong Kong groups did. Watson et al. (1994) also reported that Singaporean groups supported by CSW had significantly higher pre-meeting consensus than their U.S. counterparts, but they reported no difference in levels of post-meeting consensus. Mejias et al. (1997b) found that Mexican CSW groups generated higher levels of consensus than U.S. groups did, although no difference existed between Mexican and U.S. CSW groups in the change of consensus levels. Overall, results indicate that collectivistic groups may favor a more defined approach to convergence and agreement in comparison to U.S. groups. Individualists may be more adept at accommodating divergent viewpoints than collectivists are.

Outcome: Group Productivity

CSW can increase group productivity in terms of time spent (Dennis, 1994), idea production (Gallupe, DeSanctis, & Dickson, 1988), and document length (Lowry & Nunamaker, 2003). A few cultural studies on CSW have investigated group productivity in terms of idea generation and reported mixed results. In Atkinson and Pervan (1998), participants from high-power-distance cultures who used the anonymity feature of CSW derived a higher level of productivity than those from low-power-distance cultures (i.e., Malaysia > Indonesia > Singapore > Australia). Conversely, Mejias et al. (1997b) compared group productivity levels between U.S. and Mexican groups and found that culture had a significant impact on group productivity. U.S. groups and Mexican groups supported by CSW (both anonymous and identified) generated more comments per participant than did face-to-face groups without CSW. Regardless of CSW, U.S. groups generated more comments on average per individual than Mexican groups did; U.S. groups also produced more unique ideas per group than Mexican groups did in face-to-face settings without CSW and in identified settings supported by CSW. Another study (Quaddus & Tung, 2002) supported these results, with Australian groups (low power distance) having higher productivity than Singaporean groups (high power distance).

Daily et al. (1996, 1998) compared the productivity of culturally homogeneous and culturally heterogeneous groups. Among groups using CSW, culturally heterogeneous groups produced a significantly higher number of unique ideas than culturally homogeneous groups did. A possible explanation for this finding is that CSW may aid in conflict management and diffuse intergroup conflict in culturally diverse groups, thus increasing productivity. According to Triandis, Hall, and Ewen (1965), when a culturally heterogeneous group employs a process to reduce stress and communication problems, it becomes more creative than a homogeneous group.

Outcome: Quality of Group Outcome

CSW can increase the quality of group outcome by providing group members with equal opportunities to contribute instantaneously and anonymously (George, Easton, Nunamaker, & Northcraft, 1990). Three cultural studies investigated the effects of CSW on the decision quality but reported contrasting findings. Daily et al. (1996) found no significant differences in the quality of solutions produced by culturally

homogenous and culturally heterogeneous groups supported by CSW; Samarah et al. (2003) and Souren et al. (2004), however, reported that the interaction of collaborative conflict management style and cultural diversity of groups supported by CSW had a significant effect on perceived decision quality.

Outcome: Satisfaction

User satisfaction reflects perceived individual goal attainment, as well as perceived future gains (Briggs, De Vreede, & Reinig, 2003). Results of traditional CSW studies on user-perceived satisfaction are mixed, with some showing higher satisfaction (George et al., 1990) and others lower (Gallupe et al., 1988). Because cultures may differ in the nature of individual goals, results of user satisfaction in previous studies are also mixed. Some studies find that Western users of CSW have higher levels of satisfaction than non-Western users (Reinig & Mejias, 2003). Yet more studies report that CSW evokes feelings of comfort and satisfaction among participants from non-Western cultures (Morales et al., 1995; Mejias et al., 1997b; De Vreede et al., 1999). For example, Mejias et al. (1997b) reported that Mexican groups supported by CSW perceived higher levels of satisfaction than their U.S. counterparts did. This difference might be caused by the interactive effect between the culture factor and experimental treatment factors.

The findings discussed above suggest that some national cultures may be susceptible to different types of group dysfunctions than others and that CSW may be used to neutralize those negative influences.

Culture as a Moderator

Although the majority of previous studies considered culture as an independent variable, culture may also be a moderator of other factors (Tan et al., 1998a; Samarah et al., 2003). For example, the effect of communication media may be moderated by culture (Tan et al., 1998a). Thus, if we assume that collectivistic cultures value relationship building and openness, and that distributed communication tends to decrease satisfaction in general groups, then it is possible that the degree of collectivism would moderate satisfaction in distributed groups. Other moderation relationships are possible and remain largely unexplored.

DISCUSSION: LIMITATIONS, EXTENSIONS, AND NEW OPPORTUNITIES

In the previous section, we reviewed the existing literature and analyzed various issues regarding cultural differences and their impact in group settings supported by CSW. This section seeks to highlight major limitations of current research and provides some insights into future research opportunities and methods.

Existing research has barely begun to address fundamental research questions. More empirical research needs to be conducted to fully examine whether and how the effective use of CSW is contingent upon cultural norms. Since culture is a prominent factor in general IT adoption (Hasan & Ditsa, 1999), it also likely affects the adoption and use of CSW. Research suggests that groups are more likely to adopt a technology if their own values match or fit the values embedded within the technology or those associated with its development (Leidner & Kayworth, 2006). One might thus assume that the use of CSW would be more suitable in a collectivistic culture than in an individualistic culture, but this is not necessarily true (Davison, 1996). In collectivistic cultures, the use of CSW that incorporates anonymous communication may have dysfunctional effects (Watson et al., 1994). Further, in collectivistic cultures in which public dissent is discouraged and early consensus is encouraged, members have a social obligation to conform to rules that place

national or group interests higher than individual interests. Although the structure and anonymity of CSW can facilitate expression of conflict in North American groups, they may not help collectivistic groups because CSW forces group members to be direct and open. This feature is undesirable in collectivistic cultures in which people prefer to express disagreement indirectly in order to preserve group harmony. Therefore, the degree of fit between a group's social values/norms and the values embedded in the CSW is an important construct for studying the relationship between cultural values and the adoption of CSW.

To guide future empirical research, we highlight seven major limitations identified in existing research and discuss potential future research opportunities.

Failing to Pass the "Absurdity Test"

Related literature clearly shows that culture matters but does not fully and consistently explain and predict *why* culture matters. Some authors simply note the observed differences among subjects from different countries and label them as "cultural differences," without linking those differences to specific cultural beliefs or values and/or without having any cultural theory as a theoretical foundation (Gallivan & Srite, 2005). Failing to answer "why" may be the greatest limitation of existing research and the greatest opportunity for future research.

The strongest theoretical argument against existing research is that it appears to fail the "absurdity test" for developing a theory. In positivist and postpositivist inquiry, the aim of the empirical researchers is to study effects rather than causes of effects. To develop theories, researchers should check for logical absurdity by examining whether an increase or decrease in a cause logically results in some kind of corresponding increase or decrease in the phenomenon of interest (i.e., effect) (Briggs & Dean, 2005). This simple test checks whether IVs are expressed in a manner that logically gives the possibility to the rise of DVs. For example, "more culture" does not logically lead to "more satisfaction" because culture is a construct too abstract to give logical rise to satisfaction. Conversely, a "higher degree of collectivism" could logically lead to more satisfaction. Hence, researchers can benefit from being clearer in defining the phenomenon of interest they are studying. To do so, researchers should develop generalizable theories and research models to explain and predict the chosen phenomenon and define constructs that operationalize logically to pass the absurdity test.

In order to understand the "why" behind relationships between variables, researchers in the field of computer-supported multicultural collaboration need to more concretely define the phenomenon of interest. There are two major theoretical weaknesses in some existing research. First, rather than focusing on one or two major phenomena of interest, several studies examine a cocktail of outcomes. The problem with the cocktail approach to theoretical outcomes is that it tends to place theoretical explanations and predictions on shaky theoretical grounds by making empirical studies look more like "fishing expeditions" for statistical significance. The cocktail approach leads to at least one violation of the absurdity test as researchers essentially propose something like: "More CSW use in cross-cultural groups leads to more productivity, more outcome satisfaction, more process satisfaction, more efficiency, and more quality of decision." Although this approach allows researchers to operationalize and hypothesize specific measures, it is theoretically flawed because each phenomenon of interest should have a theoretically sound explanation and prediction as to what causes it. Deep theoretical investigations into common phenomena such as productivity, satisfaction, and quality demonstrate that entirely different mechanisms typically exist for explaining and predicting their existence.

Thus, mixing disparate effects often creates theoretical hodgepodge. We need to develop good theories that can succinctly explain and predict the phenomenon of interest affected by culture in collaboration environments.

Second, although CSW is an appropriate operationalization of a particular technology to be used as an IV, a specific technology should not be used as a construct in a theoretical proposition. Accordingly, researchers need to break down specific theoretical constructs of CSW that give rise to the improvement of a group's results; otherwise, the developed theories will be technology dependent and not generalizable. For example, social presence is an underlying construct that has been proposed as a theoretical explanation for superior or inferior technological outcomes. Social presence is constant and potentially generalizable in effects, regardless of specific technologies that will come and go. Given appropriate theoretical explanations and predictions, a study could pass the absurdity test by following a proposition pattern such as: "A higher level of social presence leads to higher satisfaction with group outcome." An IV can then be operationalized for specific CSW, such as various versions of GSS or CMC products. Defining the phenomenon of interest and using a generalizable framework will help build the body of knowledge that supports the "why" behind culture, collaboration, and technology support.

Our theoretical criticisms highlight many promising opportunities in this line of research. First, researchers need to clearly demonstrate that certain constructs, which are known to instantiate differently across cultures, can give rise to differences in certain outcomes that may be important. For example, people are motivated by status attainment in most cultures, but the means of attaining and symbolizing status may differ by culture. Hence, researchers should develop clear taxonomies, conduct studies on particular constructs that are known to manifest themselves differently across cultures, and explain why. For instance, we intuitively know that software developers who are Hispanic Baptists living in Texas are culturally different from those in India who are Hindu. Those differences can create strong team management challenges for a global software engineering firm. Yet it is atheoretical and not generalizable to create a pseudo-theoretical proposition such as: "Baptists behave in a certain way, whereas Hindus behave in another way."

Singular Focus on National Culture

The logic that "Americans will behave in a certain way while citizens of another country X will behave in another way" is the dominant paradigm in prior research. Straub, Loch, Evaristo, Karahanna, and Srite (2002) and Myers and Tan (2002) highlight and criticize the focus of previous research that leans on nationalistic definitions of culture. They assert that with globalization, culture aligns itself less with the definition of a nation-state because many countries are melting pots of various cultures. In addition, culture may not be static. It becomes increasingly difficult for any cultural group to remain isolated and uninfluenced by other cultures. Over time, societies may experience attitude changes towards gender, environment, race, family life, and religion, although these changes would rarely happen as fast as technological changes. Thus, defining a culture by nation may be too simplistic. These critics also argue that in empirical research that involves culture, researchers should identify individualistic or collectivistic individuals rather than preassigning participants to categories based solely on the country in which they live.

Cultures are generally stable in the short term, but cultural change may occur because of immigration, the influence of global media, and other social and technological innovations (Gallivan & Srite, 2005). We agree that researchers may adopt a more dynamic view of culture—one that sees

culture as contested, temporal, and emergent. For example, most prior studies simply followed the findings reported in Hofstede's study to justify their cultural categorizations, without further examination of the cultural differences among participants in the experiments. Watson et al. (1994) and Tan et al. (1998) assumed, based on Hofstede's original study, that the subjects recruited in Singapore were collectivists. We suggest that when a researcher uses national culture as an IV manipulation, it is important to validate the cultural characteristics of recruited participants to ensure that expected cultural differences exist. There should be a manipulation check of participants' cultural characteristics.

Another challenge in IS-culture research is the assumption that all individuals from a specific national culture will behave consistently based on the group's cultural values. This does not take into account the possibility of individual differences that may lead to different behavioral outcomes (Leidner & Kayworth, 2006). This indicates that researchers may need to consider individual differences (e.g., disposition) as a factor when studying the impacts of culture on group processes and outcomes.

Limited Sampling

Because of difficulty in recruiting participants with different cultural backgrounds, many studies have problems of small sample size. It is common in previous studies to have less than six groups in each experimental condition (e.g., Aiken et al., 1993, 1995; Daily et al., 1996; Walther, 1997; Atkinson & Pervan, 1998; Daily & Steiner, 1998; Quaddus & Tung, 2002; Souren et al., 2004). Such small sample size, though understandable given the challenging nature of conducting this type of research, may significantly weaken the validity and generalizability of findings. Small sample size is also the likely cause of some of the mixed findings.

Lack of Research on Group Heterogeneity

With globalization, more tasks are being accomplished by distributed teams consisting of members from varied cultural backgrounds, making it imperative to examine such groups. Thus, one area in need of expanded research is culture's influence on distributed, culturally diverse teams. More research needs to be conducted to examine how diverse team member values complement or contradict each other (Leidner & Kayworth, 2006). Specifically, it would be useful to investigate how group processes and outcomes can be improved in both culturally homogeneous and heterogeneous groups supported by CSW within a broader context of diversity management. Debate exists over whether cultural diversity improves or suppresses group performance (Anderson & Hiltz, 2001). Very few empirical studies have examined how to improve the dynamics associated with a culturally diverse workforce. In such groups, cultural diversity may cause difficulties in reaching an agreement; as a result, group harmony and cohesion might suffer, resulting in psychological pressures such as discomfort and weak feelings of belonging to the group (Samarah et al., 2003). These considerations become more complicated when considering diversity management in a broader sphere that includes nationality, expatriatism, race, ethnicity, gender, and religion.

Several research issues related to culturally heterogeneous groups are worth investigating in future research. One is the potential difference in the level of status effect and majority influence (two of the most common phenomena in group work) on group members from different cultures, which can be reflected by their behavior during a group task. For example, according to the power-distance dimension of Hofstede's model, it can be assumed that in a culture with high power distance, low-status group members may easily follow the opinions of high-status members, even

though they disagree. However, in a culture with low power distance, when there is a disagreement between low- and high-status group members, low-status members are less likely to accept the views of higher-status members if they disagree with them. Will this truly happen? Furthermore, would group members from different cultures behave differently under status effect and majority influence in face-to-face and distributed communication environments? Will the use of CSW reduce (or increase) such influences? Will the individualistic members dominate in a group task? How can managers encourage equal participation in culturally heterogeneous groups? These are all practical and interesting questions to be answered.

Language barriers could significantly prevent team members who speak different native languages from communicating with each other effectively. The inferior capability for communication due to language problems could lead to misunderstandings and less participation. Therefore, selecting appropriate participants for cross-cultural studies that examine culturally heterogeneous groups is crucial. Researchers must ensure that all team members in a culturally heterogeneous group can communicate via a common language (e.g., English) well enough. Some studies, for example, have used foreign students who are currently studying in the U.S. and can communicate in English well enough to form culturally heterogeneous groups with U.S. students (Setlock, Fussell, & Neuwirth, 2004). However, researchers should be cautious and perform manipulation checks on participants' cultural dimensions to make sure that the cultural norms of participants from foreign countries have not dramatically changed.

Too Much Focus on Face-to-Face Groups

Although work mode is one of the major IVs in previous research, participants in most studies worked only in face-to-face mode. Future studies should extend to other work modes such as distributed and asynchronous working environments. According to social presence theory, a communication medium (e.g., FtF communication) that provides more social cues will generate a higher level of social presence, thus leading to stronger social pressure and normative influence on individual group members. Such social pressure may have more impact on group members from cultures characterized by high power distance and low individualism than on those from a culture characterized by low power distance and high individualism (Zhang et al., 2007). By contrast, distributed communication supported by CSW offers fewer social cues. As a result, the corresponding social pressure and normative influence on group members are reduced, leading to potentially different behavior of group members when compared to the FtF work mode. We argue that findings in the FtF environment may not be applicable to a distributed environment. So far, only a few studies examined the effects of culture in distributed groups (Montoya-Weiss, Massey, & Song, 2001; Vogel et al., 2001a, 2001b; Walther, 1997). However, the findings about distributed work in general may provide some theoretical and empirical foundation for theory development and experimental design for research that explores the relationship between culture and distributed work.

FtF and distributed communication may be confounded by task type. Media Richness Theory (Daft & Lengel, 1986) suggests that managers can improve performance by matching media characteristics to the needs of organizational information-processing tasks, which are categorized based on uncertainty and equivocation. When ambiguity and uncertainty in tasks are high (or low), high-richness (or low-richness) media should be used. Thus, an important element of studying culture is examining various work modes with different levels of media richness and the confounding effect of task type. Watson et al.

(1994) suggest that FtF, anonymous meetings are suitable for individualists, while asynchronous and distributed meetings may be more suitable for collectivists. All hypotheses of existing research focus on synchronous communication, which may put time and emotional pressure on participants. It would thus be useful to conduct studies using distributed, asynchronous groups and to examine whether collectivistic group members behave differently when there is less social and time pressure, in contrast to their behavior in FtF, synchronous communication. In such a group, the social and time pressure is probably minimal among all communication media. That being said, studying distributed, asynchronous groups will pose many challenges to the design and execution of the research.

Singular Focus on Small Groups

Almost all studies we reviewed used only small groups (i.e., consisting of three to five members), yet it is well recognized that group size affects group outcomes and the degree of conflict within a group's structure (Steiner, 1972; Valacich, Wheeler, Mennecke, & Wachter, 1995). Small groups are more likely to resolve opinion differences, whereas in larger groups consensus is more difficult to achieve (Hare, 1981). In traditional FtF groups without CSW support, increasing group size can significantly increase process losses (Bouchard & Hare, 1970; Steiner, 1972). CSW has been shown to be effective in support of larger groups (e.g., Gallupe et al., 1992; Dennis, 1994; Valacich et al., 1995). Future research needs to examine cultural effects in groups of different group sizes and to determine whether the group size has a moderating or mediating effect on cultural influence.

Lack of Realism

All of the reviewed previous studies except Morales et al. (1995), De Vreede et al. (1999), and Calhoun, Teng, and Cheon (2002) used students working on hypothetical tasks as opposed to organizational members engaging in real-world tasks. There are a few potential problems in employing lab experiments using student subjects. First, students may not be representative of their culture. Second, some culture values examined, such as power distance, are unlikely to be reflected by participants with equal status. Although the relative homogeneity of student participants prevents a source of uncontrolled variance, their motivation to maximize the quality of group tasks is sometimes questionable. Also, an interaction might exist between types of participants and the effectiveness of technology (Fjermestad & Hiltz, 1999). Likewise, most research is conducted on groups that have no working history. Existing relationships between group members established prior to carrying out group tasks set conditions for a group's interaction (McGrath, 1984; Watson et al., 1994). Future studies should examine the performance of established groups vs. ad hoc groups.

To increase realism, researchers should consider using nonstudent participants performing realistic work tasks. The duration of group tasks may be more varied. Hong, Morris, Chiu, and Benet-Martinez (2000) note that under conditions of heavy time pressure, participants in experimental studies tend to behave in a manner consistent with trait-based cultural norms. Future studies may take a longitudinal focus, with enhanced sensitivity to group characteristics and intergroup dynamics. Since any single group session may not correctly detect long-term effects of CSW, a multilevel perspective in looking at differences between cultures should be adopted. Besides, most prior studies concentrated mainly on one task, limiting the generalizability of research findings. It would be useful to examine the processes and outcomes of cross-cultural groups that deal with different types of tasks.

Finally, we want to point out that compared to general research on CSW, conducting empirical

studies on cultural influence on group processes and outcomes is typically much more challenging and resource consuming, especially when studies involve culturally heterogeneous groups in a distributed setting. One challenge is that researchers may have to deal not only with differences in language (e.g., translating and back-translating related documents used in the experiment) and time zones, but also more complicated logistic issues and experimental design. In addition, the group task(s) and CSW selected for a study have to be equally and unambiguously understood by participants from different cultures. For example, the mock jury task that is commonly used in empirical group studies in the U.S. is not appropriate for participants from China and some other countries where jury systems do not exist.

CONCLUSION

With increasing globalization, CSW is recognized as an important family of technologies supporting collaborative work. This article makes several contributions by providing a common taxonomy of CSW research that examines the impact of culture on group collaboration. Our review and analysis lead to four general conclusions. First, interest in research on cross-cultural collaboration appears to be increasing. However, compared to hundreds of papers published in the field of CSW research, only a limited number of studies have examined the impact of culture. The body of relevant literature is too small to be able to draw any significant and general conclusions. As Ford, Connelly, and Meister (2003) point out, the research has been conducted in a manner that resists building a cumulative tradition. Findings obtained within specific contexts are difficult to generalize. Second, existing studies have investigated only limited types of group outcomes under limited contexts. The effects of cultural differences on many important constructs have yet to be investigated. Third, prior research shows that culture affects group processes and outcomes; however, the findings are still inconsistent. Part of this inconsistency may arise from the research using CSW with different features and design goals, different group tasks, and varying statistical power. Fourth, as an emerging area of research, this line of research lacks comprehensive and empirically validated theories. Researchers should realize that no single solution is universally applicable to all organizational, cultural, or social problems. Many factors can affect the performance of groups supported by CSW. It is imperative that practitioners introducing collaborative technologies into groups with disparate cultures understand cultural differences and their impact. Developing a deeper theoretical understanding of differences in cross-cultural groups will go a long way toward assisting global organizations to manage their groups more effectively. More advanced and validated theories need to be built, and better-designed empirical research needs to be conducted.

REFERENCES

Aiken, M., Hwang, C., Magalhaes, R.D., & Jeanette, M. (1993). A comparison of Malaysian and American groups using a group decision support system. *Journal of Information Science, 19*(6), 489-491.

Aiken, M., Kim, D., Hwang, C., & Lu, L. (1995). A Korean group decision support system. *Information and Management, 28*(5), 303-310.

Aiken, M., Martin, J., Shirani, A., & Singleton, T. (1994). A group decision support system for multicultural and multilingual communication. *Decision Support Systems (DSS), 12*(2), 93-96.

Anderson, W.N., & Hiltz, S.R. (2001, January 3-6). Culturally heterogeneous vs. culturally homogeneous groups in distributed group support systems: Effects on group process and consensus. *Proceedings of the 34th Hawaii International*

Conference on System Sciences (HICSS), Maui, HI.

Atkinson, D., & Pervan, G. (1998, June 4-6). Group support systems and productivity in four national cultures. *Proceedings of the 6th European Conference on Information Systems*, Aix-en-Provence, France.

Baker, S.M., & Petty, R.E. (1994). Majority and minority influence: Source-position imbalance as a determinant of message scrutiny. *Journal of Personality and Social Psychology, 67*(1), 5-19.

Berger, J., Fisek, M.H., Norman, R.Z., & Zelditch, M. (1977). *Status characteristics and social interaction*. Amsterdam: North-Holland.

Bouchard, T., & Hare, M. (1970). Size, performance, and potential in brainstorming groups. *Journal of Applied Psychology, 54*(1), 51-55.

Briggs, R.O., De Vreede, G.J., & Reinig, B.A. (2003, January 6-9). A theory and measurement of meeting satisfaction. *Proceedings of the 36th Hawaii International Conference on System Sciences* (HICSS), Big Island, HI.

Briggs, B., & Dean, D. (2005). Successful research from logical positivist's perspective. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (HICSS), Kona, HI.

Calhoun, K.J., Teng, J.T.C., & Cheon, M.J. (2002). Impact of national culture on information technology usage behavior: An exploratory study of decision making in Korea and the USA. *Behavior & Information Technology, 21*(4), 293-302.

Campbell, D.J. (1988). Task complexity: A review and analysis. *Academy of Management Review, 13*(1), 40-52.

Chung, K., & Adams, C.R. (1997). A study on the characteristics of group decision making behavior: Cultural difference perspective of Korea vs. U.S. *Journal of Global Information Management, 5*(3), 18-29.

Clapper, D.L., McLean, E.R., & Watson, R.T. (1991). An experimental investigation of the effect of a group decision support system on normative influence in small groups. *Proceedings of the 12th Annual International Conference on Information Systems* (ICIS), New York.

Connolly, T., Jessup, L.M., & Valacich, J.S. (1990). Effects of anonymity and evaluative tone on idea generation in computer-mediated groups. *Management Science, 36*(6), 689-703.

Daft, R.L., & Lengel, R.H. (1986). Organizational information requirements, media richness and structural design. *Management Science, 32*(5), 554-571.

Daily, B.F., & Steiner, R.L. (1998). The influence of group decision support systems on contribution and commitment levels in multicultural and culturally homogeneous decision-making groups. *Computers in Human Behavior, 14*(1), 147-162.

Daily, B.F., Whatley, A., Ash, S.R., & Steiner, R.L. (1996). The effects of a group decision support system on culturally diverse and culturally homogeneous group decision making. *Information and Management, 30*(6), 281-289.

Davison, R. (1996). *National cultures, organisational forms and group support systems.* Retrieved September 28, 2004, from *http://www.is.cityu.edu.hk/Research/WorkingPapers/paper/9607.pdf*

De Vreede, G., Jones, N., & Mgaya, B.J. (1999). Exploring the application and acceptance of group support systems in Africa. *Journal of Management Information Systems, 15*(3), 197-234.

Dennis, A.R. (1994). Electronic support for large groups. *Journal of Organizational Computing, 4*(2), 177-197.

Dennis, A.R., Hilmer, K.M., & Taylor, N.J. (1998). Information exchange and use in GSS and verbal group decision making: Effects of minority influence. *Journal of Management Information Systems, 14*(3), 61-88.

Earley, P.C. (1994). Self or group? Cultural effects of training on self-efficacy and performance. *Administrative Science Quarterly, 39*(1), 89-117.

El-Shinnawy, M., & Vinze, A.S. (1997). Technology, culture and persuasiveness: A study of choice-shifts in group settings. *International Journal of Human-Computer Studies, 47*(3), 473-496.

Fjermestad, J., & Hiltz, S.R. (1999). An assessment of group support systems experimental research methodology and results. *Journal of Management Information Systems, 15*(3), 7-149.

Ford, D.P., Connelly, C.E., & Meister, D.B. (2003). Information systems research and Hofstede's culture's consequences: An uneasy and incomplete partnership. *IEEE Transactions on Engineering Management, 50*(1), 8-25.

Fulk, J., Schmitz, J., & Steinfield, C. (1990). A social influence model of technology use. In J. Fulk & C. Steinfield (Eds.), *Organizations and communication technology* (pp. 117-142). Newbury Park, CA: Sage.

Gallivan, M., & Srite, M. (2005). Information technology and culture: Identifying fragmentary and holistic perspectives of culture. *Information and Organization, 15,* 295-338.

Gallupe, R.B., Bastianutti, L.M., & Cooper, W.H. (1991). Unlocking brainstorms. *Journal of Applied Psychology, 76*(1), 137-142.

Gallupe, R.B., Dennis, A.R., Cooper, W.H., Valacich, J.S., Bastianutti, L.M., & Nunamaker, J.F. Jr. (1992). Electronic brainstorming and group size. *Academy of Management Journal, 35*(2), 350-369.

Gallupe, R.B., DeSanctis, G., & Dickson, G.W. (1988). Computer-based support for group problem-finding: An experimental investigation. *MIS Quarterly, 12*(2), 277-298.

George, J.F., Easton, G., Nunamaker, J.F., & Northcraft, G. (1990). A study of collaborative group work with and without computer-based support. *Information Systems Research, 1*(4), 394-415.

Gray, P., & Olfman, L. (1989). The user interface in group decision support systems. *Decision Support Systems, 5*(2), 119-137.

Griffith, T.L. (1998). Cross-cultural and cognitive issues in the implementation of new technology: Focus on group support systems and Bulgaria. *Interfacing with Computers, 9*(4), 431-447.

Groeschl, S., & Doherty, L. (2000). Conceptualizing culture. *Cross Cultural Management—An International Journal, 7*(4), 12-17.

Hall, E.T., & Hall, M.R. (1990). *Understanding cultural differences.* Yarmouth, ME: Intercultural Press.

Hare, A.P. (1981). Group size. *American Behavioral Scientist, 24,* 695-708.

Hasan, H., & Ditsa, G. (1999). The impact of culture on the adoption of IT: An interpretive study. *Journal of Global Information Management, 7*(1), 5-15.

Ho, T., Raman, K., & Watson, R. (1989, December 4-6). Group decision support systems: The cultural factor. *Proceedings of the 10th Annual International Conference on Information Systems,* Boston.

Hofstede, G. (1984). *Culture's consequences: International differences in work related values.* London: Sage.

Hofstede, G. (1991). *Cultures and organizations: Software of the mind.* Berkshire, England: McGraw-Hill.

Hofstede, G. (2001). *Culture's consequences comparing values, behaviors, institutions, and organizations across nations* (2nd ed.). London: Sage.

Hong, Y., Morris, M.W., Chiu, C., & Benet-Martinez, V. (2000). Multicultural minds: A dynamic

constructivist approach to culture and cognition. *American Psychologist, 55*(7), 709-720.

Jessup, L.M., Connolly, T., & Galegher, J. (1990). The effects of anonymity on GDSS group process with an idea-generating task. *MIS Quarterly, 14*(3), 313-321.

Kim, K., Park, H., & Suzuki, N. (1990). Reward allocations in the United States, Japan and Korea: A comparison of individualistic and collectivistic cultures. *Academy of Management Journal, 33*(1), 188-198.

Kim, U., Triandis, H.C., Kagitcibasi, C., Choi, S.C., & Yoon, G. (1994). *Individualism and collectivism: Theory, methods and applications.* Thousand Oaks, CA: Sage.

Kluckhohn, C. (1962). *Culture and behavior.* New York: The Free Press.

Leidner, D.E., & Kayworth, T. (2006). A review of culture in information systems research: Toward a theory of information technology culture conflict. *MIS Quarterly, 30*(2), 357-399.

Levine, J.M., & Russo, E.M. (1987). Majority and minority influence. In C. Hendrick (Ed.), *Group processes* (pp. 13-54). Newbury Park, CA: Sage.

Lowry, P.B., & Nunamaker, J.F. Jr. (2003). Using Internet-based, distributed collaborative writing tools to improve coordination and group awareness in writing teams. *IEEE Transactions on Professional Communication, 46*(4), 277-297.

McGrath, J.E. (1984). *Groups: Interaction and performance*. Englewood Cliffs, NJ: Prentice Hall.

Mejias, R.J., Shepherd, M.M., Vogel, D.R., & Lazaneo, L. (1997a). Consensus and perceived satisfaction levels: A cross-cultural comparison of GSS and non-GSS outcomes within and between the United States and Mexico. *Journal of Management Information Systems, 13*(3), 137-161.

Mejias, R.J., Vogel, D.R., & Shepherd, M.M. (1997b, January 7-10). GSS meeting productivity and participation equity: A U.S. and Mexico cross-cultural field study. *Proceedings of the 30th Hawaii International Conference on System Sciences* (HICSS).

Miranda, S., & Saunders, C. (2003). The social construction of meaning: An alternative perspective on information sharing. *Information Systems Research, 14*(1), 87-106.

Montoya-Weiss, M.M., Massey, A.P., & Song, M. (2001). Getting it together: Temporal coordination and conflict management in global virtual teams. *Academy of Management Journal, 44*(6), 1251-1262.

Morales, B., Moriera, H., & Vogel, D.R. (1995, January 4-7). Group support for regional development in Mexico. *Proceedings of the 28th Hawaii International Conference on System Sciences* (HICSS).

Moscovici, S., & Zavalloni, M. (1969). The group as a polarizer of attitudes. *Journal of Personality and Social Psychology, 12*, 125-135.

Myers, M.D., & Tan, F.B. (2002). Beyond models of national culture in information systems research. *Journal of Global Information Management, 10*(1), 24-32.

Poole, M.S., Seibold, D.R., & McPhee, R.D. (1985). Group decision-making as a structural process. *Quarterly Journal of Speech, 71,* 74-102.

Pruitt, D.G. (1971). Choice shifts in group discussion: An introductory review. *Journal of Personality and Social Psychology, 20,* 339-360.

Quaddus, M.A., & Tung, L.L. (2002). Explaining cultural differences in decision conferencing. *Communications of the ACM, 45*(8), 93-98.

Rahim, M.A. (1983). A measure of styles of handling interpersonal conflict. *Academy of Management Journal, 26*(2), 368-376.

Reinig, B.A., & Mejias, R.J. (2003, January 6-9). An investigation of the influence of national culture and group support systems on group processes and outcomes. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences* (HICSS), Big Island, HI.

Ridgeway, C.L., Berger, J., & Smith, L. (1985). Nonverbal cues and status: An expectation states approach. *American Journal of Sociology, 90*(5), 955-978.

Robichaux, B.P., & Cooper, R.B. (1998). GSS participation: A cultural examination. *Information & Management, 33,* 287-300.

Samarah, I., Paul, S., Mykytyn, P., & Seetharaman, P. (2003, January 6-9). The collaborative conflict management style and cultural diversity in DGSS supported fuzzy tasks: An experimental investigation. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences* (HICSS), Big Island, HI.

Setlock, L.D., Fussell, S.R., & Neuwirth, C. (2004, November 6-10). Taking it out of context: Collaborating within and across cultures in face-to-face settings and via instant messaging. *Proceedings of CSCW'04,* Chicago, IL.

Short, J., Williams, E., & Christie, B. (1976). *The social psychology of telecommunication*. London: John Wiley & Sons.

Souren, P., Priya, S., Imad, S., & Mykytyn, P.P. (2004). Impact of heterogeneity and collaborative conflict management style on the performance of synchronous global virtual teams. *Information and Management, 41*(3), 303-321.

Steiner, I.D. (1972). *Group processes and productivity.* New York: Academic Press.

Straub, D., Loch, K., Evaristo, R., Karahanna, E., & Srite, M. (2002). Toward a theory-based measurement of culture. *Journal of Global Information Management, 10*(1), 13-23.

Tan, B.C., Watson, R., & Wei, K.-K. (1995). National culture and group support systems: Filtering communication to dampen power differentials. *European Journal of Information Systems, 4*(2), 82-92.

Tan, B.C.Y., Watson, R.T., Wei, K.K., Raman, K.S., & Kerola, P.K. (1993, January 5-8). National culture and group support systems: Examining the situation where some people are more equal than others. *Proceedings of the 26th Annual Hawaii International Conference on System Sciences* (HICSS), Wailea, HI.

Tan, B.C.Y., Wei, K.-K., Watson, R.T., Clapper, D.L., & McLean, E.R. (1998a). Computer-mediated communication and majority influence: Assessing the impact in an individualistic and a collectivistic culture. *Management Science, 44*(9), 1263-1278.

Tan, B.C.Y., Wei, K.-K., Watson, R.T., & Walczuch, R.M. (1998b). Reducing status effects with computer-mediated communication: Evidence from two distinct national cultures. *Journal of Management Information Systems, 15*(1), 119-142.

Triandis, H. (1972). *An analysis of subjective culture.* New York: John Wiley & Sons.

Triandis, H.C. (1989). The self and social behavior in differing cultural contexts. *Psychological Review, 96,* 506-520.

Triandis, H.C., Hall, E.R., & Ewen, R.B. (1965). Member homogeneity and dyadic creativity. *Human Relations, 18,* 33-54.

Tung, L.L., & Quaddus, M.A. (2002). Cultural differences explaining the differences in results in GSS: Implications for the next decade. *Decision Support Systems, 33,* 177-199.

Tung, R. (1995). Strategic human resource challenge: Managing diversity. *International Journal of Human Resource Management, 6*(3), 482-494.

Usunier, J.C. (1998). *International and cross-cultural management research*. London: Sage.

Valacich, J., Wheeler, B., Mennecke, B., & Wachter, R. (1995). The effects of numerical and logical size on computer-mediated idea generation. *Organizational Behavior and Human Decision Processes, 62*(3), 318-329.

Veiga, J.F., Floyd, S., & Dechant, K. (2001). Towards modeling the effects of national culture on IT implementation and acceptance. *Journal of Information Technology, 16*(3), 145-158.

Vick, R.M. (1998). Perspectives on and problems with computer-mediated teamwork: Current groupware issues and assumptions. *Journal of Computer Documentation, 22*(2), 3-22.

Vogel, D., Davison, R., & Shroff, R. (2001a). Sociocultural learning: A perspective on GSS-enabled global education. *Communications of AIS, 7*(9), 1-41.

Vogel, D., Van Genuchten, M., Lou, D., Verveen, S., Van Eekout, M., & Adams, A. (2001b). Exploratory research on the role of national and professional cultures in a distributed learning project. *IEEE Transactions on Professional Communication, 44*(2), 114-125.

Walsham, G. (2002). Cross-cultural software production and use: A structurational analysis. *MIS Quarterly, 26*(4), 359-380.

Walther, J.B. (1997). Group and interpersonal effects in international computer-mediated collaboration. *Human Communication Research, 23*(3), 342-369.

Watson, R., Ho, T., & Raman, K. (1994). Culture: A fourth dimension of group support systems. *Communications of the ACM, 37*(10), 44-55.

Watson, R.T. (1987). *A study of group decision support system use in three and four person groups for a preference allocation decision*. Unpublished Doctoral Dissertation, University of Minnesota, USA.

Zakour, A.B. (2004, February 27-28). Cultural differences and information technology acceptance. *Proceedings of the 7th Annual Conference of the Southern Association for Information Systems*, Savannah, GA.

Zhang, D., Lowry, P.B., Zhou, L., & Fu, X. (2007). The impact of individualism-collectivism, social presence, and group diversity on group decision making under majority influence. *Journal of Management Information Systems, 23*(4), 53-80.

Zigurs, I., Poole, M.S., & DeSanctis, G. (1988). A study of influence in computer-mediated group decision making. *MIS Quarterly, 12*(4), 625-644.

APPENDIX A. EXISTING STUDIES ON CULTURE AND COLLABORATIVE SOFTWARE

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Aiken et al. (1993)	Preliminary study comparing North American and Malaysian groups using GSS; had Malaysian groups switch between English and Malay	Laboratory experiment	Creativity task	(IV) language; (DV) production blocking, evaluation apprehension, satisfaction	9 Malaysians speaking Malay; 9 Malaysians speaking English; 16 North Americans speaking English	Not specified	No differences were found between all-North American groups and all-Malaysian groups using GSS in terms of production blocking (all low), evaluation apprehension (all low), and satisfaction (all high).
***Aiken et al. (1995)	Compares the perceived effectiveness and satisfaction of users who use English and Korean versions of the same GSS	Laboratory experiment	Creativity tasks	(IV) language (Korean vs. English); (DV) production blocking, evaluation apprehension, process satisfaction	12 Korean students at University of Mississippi (1 group)	Confucian philosophy in Korean society	No significant differences were found between English and Korean versions of systems in terms of ratings of evaluation apprehension, production blocking, and process satisfaction; GSS reduced negative effects of verbal meetings conducted in Korea.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Anderson & Hiltz (2001)	Compares groups from the same cultural background with groups from varied cultural backgrounds when they use two different communication media systems	Laboratory experiment	A value-based cognitive conflict (negotiation) task	(IV) communication mode (manual F2F and asynchronous distributed) and group composition (culturally homogeneous and heterogeneous); (DV) adaptation factors and outcome factors	A total of 46 groups: 20 homogeneous (U.S.) manual F2F groups and distributed groups; 26 heterogeneous groups consisting of members from non-U.S. countries	Hofstede's cultural dimensions	Face-to-face culturally heterogeneous (mixed) groups had the highest level of post-meeting consensus, while asynchronous culturally homogeneous (U.S.) groups had the lowest level; no significant differences were found based on cultural composition of the groups.
Atkinson & Pervan (1998)	Exploratory study compares productivity of groups from four national cultures using GroupSystems	Exploratory laboratory experiment (4*2 repeated measure design)	Creativity task (idea generation)	(IV) anonymity and culture; (DV) group productivity	Australia (3 groups), Singapore (1 group), Malaysia (1 group), Indonesia (1 group); most groups included 10 participants	Hofstede's model	Exploratory, low sample study indicates higher power-distance cultures may derive greater productivity from anonymity; all groups from different cultures perceive anonymity as an advantage.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Calhoun et al. (2002)	Exploratory survey study that examines the use of IT for organizational decision making in Korea and U.S.	Survey	Respondents were asked to consider the use of CMC to send and receive info in decision making	(IV) intensity of IT use for decision making; (DV) 17 decision attributes	65 Korean participants; 77 U.S. participants; all were employees	Hofstede's model	Exploratory results show that decision makers in Korea and the U.S. had different perceptions of the IT use that impacted their decision-making activities.
Daily et al. (1996) Daily & Steiner (1998)	Examines the influence of a GSS on contribution and commitment levels in culturally homogeneous and CC (cross-cultural) decision-making groups Two papers report on the same study's data.	Laboratory experiment (with 2*2 factorial design)	Three creativity tasks	(IV) GSS support (w/ and w/o GSS), cultural diversity; (DV) perceived contribution, number of unique ideas generated, solution quality, commitment, personal influence	Hispanic, Anglo, and others; 12 groups: 6 heterogeneous and 6 homogeneous groups (4 to 5 members per group)	Not specified	Culturally diverse groups outperformed culturally homogeneous groups on the number of ideas generated, but no significant effect on the solution quality. None of the effects of perceived contribution, commitment, or personal influence were found to be significant.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
De Vreede et al. (1999)	Explores the effective use and acceptance of GSS in a CC context	Field study	Multiple tasks, including preference task and creativity task	(IV) culture; (DV) the use of GSS for decision making	3 countries: Tanzania, Malawi, and Zimbabwe; group sizes varied from 5 to 120 in each project	Hofstede's model and Technology Acceptance Model (TAM)	GSS could lead to significant differences in technology acceptance, use, and diffusion, as well as user satisfaction.
El-Shinnawy & Vinze (1997)	Examines the impact of GSS and culture on the process and outcomes of group decision making (polarization)	Laboratory experiment (2*2 repeated factorial design)	Preference task (the Pentium problem)	(IV) medium (face-to-face vs. CMC) and culture; (DV) polarization, persuasive arguments, novelty, validity	U.S. (24 groups) vs. Singapore (24 groups); 6 members per group	Hofstede's model; Persuasive Arguments Theory (PAT)	Culture and communication medium had significant effects on polarization; neither medium nor culture had main effects on persuasive arguments; the GSS medium had a higher impact on groups in the U.S. than on groups in Singapore.
Griffith (1998)	A cognitive model of CC implementation is tested using GroupSystems and Bulgarian and U.S. students	Laboratory experiment	Preference task (lunar survival problem)	(IV) culture; (DV) innovation, critique, satisfaction	U.S. (16 students) vs. Bulgaria (15 students); data analysis on individual level	Hofstede's model (power distance)	Showed that Bulgarian students may be more likely to challenge authority than their U.S. counterparts. Power distance mediates some effects between culture and satisfaction with GSS.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Ho, Raman, & Watson (1989)	A GSS study (using SAMM) in Singapore that explores the cultural differences between U.S. and Singapore	Laboratory experiment	Preference task (allocation of funds to 6 projects)	(IV) level of support (3 levels); (DV) post-meeting consensus, influence equality	Singapore: 48 5-person groups' results were compared with the findings in a similar U.S. study (Watson, 1987)	Hofstede'sindividualism-collectivism dimension	Singaporean groups were indirect in the communication and seldom expressed disagreement in an open manner; the anonymity feature led to lower influence equality in Singapore.
Mejias et al. (1997a)	Examines the effect of culture on productivity, consensus level, and participation equity during the use of GSS	Field study with 2*2 within-subjects design	Creativity task	(IV) GSS support, anonymity, national culture; (DV) number of ideas and unique ideas, participation equity, consensus level, satisfaction with decision	U.S. (22 groups) vs. Mexico (20 groups); all were divided into 3 treatments	Hofstede's model (first 4 dimensions)	Significant differences were found in the number of ideas generated, consensus levels, and relative levels of user satisfaction across cultures.
Morales et al. (1995)	Explores the application of GroupSystems in regional development in Mexico and compares with findings from the U.S.	Field study	Preference task (regional development)	Not specified	293 Mexican participants from actual organizations	Hofstede's model	Participants agreed that communication within the group was more effective with the use of the GSS; there was disagreement as to whether FtF communication would be more effective than the use of GSS.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Niederman (1997)	Exploratory and atheoretical study compares Mexican and U.S. group facilitators using key elements of meeting success and selection of GSS tools	Interview and tape recordings	Not specified	Not specified	U.S. vs. Mexico: 7 Mexican group facilitators, 37 U.S. facilitators	Hofstede's model	Results found no compelling differences between what Mexican and U.S. facilitators consider important measures of meeting success, expected benefits and concerns, tool selection, and so forth.
Quaddus & Tung (2002)	Compares two cultures in the context of group conflict generation and management via a non-networked GSS	Laboratory experiment (2*2 factorial design)	Preference and creativity tasks: (1) resource allocation, (2) strategic planning	(IV) technical support and task; (DV) amount and type of conflict, conflict resolution strategies, productivity	5 groups for both Australia and Singapore (3 or 4 members per group)	Hofstede's model	Higher levels of conflict were generated in Australia than in Singapore; Australians tended to use fewer avoidance strategies and report more productivity than the Singaporeans.
Reinig & Mejias (2003)	Examines the influence of GSS and national culture on group processes, meeting satisfaction, and group outcomes	Laboratory experiment (2*2 factorial design)	Preference task	(IV) GSS support and national culture; (DV) levels of consensus, production blocking, dominance, satisfaction, participation equality	U.S.: 22 groups (11 GSS and 11 FtF groups, with 7 to 8 members per group); Hong Kong: 18 groups (9 GSS and 9 FtF groups, with 7 to 8 members per group)	Hofstede's model and Social Information Processing (SIP) theory	No substantial differences found between cultures: GSS users reported less production blocking and dominance, and lower levels of consensus and satisfaction than did FTF participants across both the U.S. and Hong Kong samples.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Samarah et al. (2003)	Examines the moderating effect of cultural diversity on the relationship between the conflict management style and group performance	Laboratory experiment	Fuzzy task	(IV) conflict management style; moderator: culture diversity; (DV) degree of agreement, perceived decision quality, participation; moderator: culture	U.S. vs. India: 4 U.S. homogeneous groups; 9 Indian homogeneous groups; 9 heterogeneous groups; 3 to 4 members per group	Hofstede's model	Showed that cultural diversity has a positive moderating effect on the degree of group agreement and perceived decision quality.
Souren et al. (2004)	Investigates the impact of heterogeneity and collaborative conflict management style on the performance of synchronous virtual teams using a Web-based GSS	Laboratory experiment (4*2 factorial design)	Preference task (selecting one option to recommend to a university about adopting a computer-use fee)	(IV) group heterogeneity vs. heterogeneity and conflict management style; (DV) satisfaction, perceived decision quality, perception of participation, group agreement	U.S. vs. India: 15 4-person groups and one 3-person group; U.S. homogeneous groups	Mentions Hofstede's model but not to generate hypothesis or interpret results	Collaborative conflict management style positively impacted satisfaction with the decision-making process, perceived decision quality, and perceived participation of virtual teams; weak evidence links a group's heterogeneity to its collaborative conflict management style.

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Tan et al. (1998a)	Investigates whether CMC can reduce normative influence from majorities in three decision-making settings	Laboratory experiment (3*2*2 factorial design)	(1) An intellectual task; (2) a preference task (mock jury)	(IV) national culture, task type, and communication medium; (DV) the number of rounds taken by the group to reach consensus	U.S. (8 groups in each treatment) vs. Singapore (11 or 12 groups in each treatment); 4-person groups; 6 treatments	Hofstede's model (individualism-collectivism)	Subjects in the unsupported setting took fewer rounds to reach consensus than those in FtF CMC and dispersed CMC; for both tasks, majority influence did not vary with communication medium.
Tan et al. (1998b)	Examines whether CMC can reduce status effects during group communication in two national cultures	Laboratory experiment (2*2*2 factorial design)	(1) An intellectual task; (2) a preference task	(IV) national culture, task type, and communication medium; (DV) status influence, sustained influence, perceived influence	U.S. (45 groups) vs. Singapore (48 groups); five-person groups; four treatments, with 10 to 12 groups per treatment	Hofstede's model (individualism-collectivism and power distance)	Task type and communication medium had significant main effects on status influence; status influence was not significantly stronger in Singapore groups than in U.S. groups; status influence was higher in preference task groups than in intellective task groups, etc.
Tung & Quaddus (2002)	Conducts a comparable study on the use of GSS in two different countries to explain the impact of culture on differences in results	Laboratory experiment (2*2 factorial-repeated measure)	Preference task and creativity task: (1) resource allocation; (2) strategic planning	(IV) task and technical support; (DV) outcome in terms of the "productivity" of the conflicts	Australian: 6 groups (3 to 5 members per group); Singapore: 20 groups (3 or 4 members per group)	Hofstede's model	Revealed differences in the significance of technical support and tasks with productivity in the Singaporean and Australian studies (higher avoidance behaviors in Singaporean groups and higher levels of interpersonal conflict in Australian groups).

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Vogel et al. (2001a)	Explores how GroupSystems can facilitate virtual teams in an educational environment	Field study	7 joint projects (identify the impact of software defects and CSFs) (intellectual task)	Not specified	The Netherlands, Greece, and Hong Kong; no between-group comparison; each project consists of one group only, with at least 48 students from 2 different regions	Hofstede's model and Sociocultural Learning Theory	Encountered some communication problems; richer interaction led to higher performance; attraction to work with different cultures varied greatly among students; cultural differences emerged in team feelings.
Vogel et al. (2001b)	Reports the cultural difference reflected in group member behavior in a CC course project	Field study	A 7-week joint project on a specific IT-related subject, resulting in a joint report (intellectual task)	Not specified	32 Hong Kong students and 39 Dutch students;10 CC groups, with 6 to 10 members per group	Hofstede's model and Sociocultural Learning Theory	A cultural effect existed, reflected by different behaviors of members from different cultures (e.g., Hong Kong students tended to resolve issues by discussing them with their local teammates, while Dutch students were more inclined to address teammates from both cultural backgrounds).

Authors	Research Focus	Research Methodology	Task(s) Used	Independent (IV) and Dependent (DV) Variables	Cultures Involved and Group Size	Underlying Theory or Model	Major Findings
Walther (1997)	Examines the interplay of culture with long-term and short-term groups in FtF and distributed CMC conditions	Laboratory experiment (2*2 factorial design)	Writing a paper summarizing, critiquing, and commenting on 5 articles	(IV) team duration (long term vs. short term), identity (social vs. individual); (DV) social attractiveness, task attractiveness, physical attractiveness	54 students, in groups of 5 to 6, drawn from U.S. and Britain	Recent interaction theories	Distributed groups were just as effective when examining social attractiveness, task attractiveness, and physical attractiveness.
Watson et al. (1994)	Evaluates the cultural effect on change in consensus and influence equality in three different communication settings	Laboratory experiment (3*2 factorial design)	Preference task: allocation of money to 6 projects	(IV) type of decision support (CS, manual, baseline), national culture; (DV) change in consensus, influence equality	U.S. vs. Singapore: U.S.: 3 to 4 members per group, group sizes for 3 decision support treatments were 27, 26, and 29; Singapore: 5-person groups; group sizes for 3 decision-support treatments were 14, 16, and 15	Hofstede's model and Adaptive Structuration Theory (AST)	Singaporean groups had higher pre-meeting consensus than U.S. groups; all groups in both cultures had the same level of post-meeting consensus; change in consensus was greater in U.S. than in Singaporean groups.

This work was previously published in the Journal of Global Information Management, edited by F. Tan, Volume 16, Issue 1, pp. 61-84, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 5.25
Online Behavior Modeling:
An Effective and Affordable Software Training Method

Charlie Chen
Appalachian State University, USA

Terry Ryan
Claremont Graduate University, USA

Lorne Olfman
Claremont Graduate University, USA

ABSTRACT

Organizations need effective and affordable software training. In face-to-face settings, behavior modeling (BM) is an effective, but expensive, training method. Can BM be employed effectively, and more affordably, for software training in the online environment? An experiment was conducted to compare the effectiveness of online BM with that of face-to-face (F2F) BM for software training. Results indicate that online BM and F2F BM provide essentially the same outcomes in terms of knowledge near transfer, immediate knowledge far transfer, delayed knowledge far transfer, perceived ease of use, perceived usefulness and satisfaction. Observed differences were not significant, nor were their patterns consistent, despite sufficient power in the experimental design to detect meaningful differences. These results suggest that organizations should consider online BM as a primary method of software training.

INTRODUCTION

Investment in software training can improve productivity, boost employee morale (Bell, 2004) and reduce employee turnover rate (Heller, 2003). End users who have not received proper software training often feel insecure about their jobs, and this insecurity can contribute to turnover costs and productivity losses (Aytes & Connolly, 2004). The departure of a newly hired IT employee within 180 days of hiring can cost a company as much

as $100,000 (Brown, 2000). The departure of employees who leave their companies due to a lack of proper training can have a variety of negative consequences (McEvoy & Cascio, 1987).

In contrast, properly trained end users often feel confident and secure, with positive implications for productivity. Increases in individual performance can add up to substantial improvements for businesses. The American Society for Training and Development (ASTD) conducted a study of 575 United States (U.S.)-based, publicly traded firms between 1996 and 1998 to examine the relationship between organizational training investments and total shareholder return. This study found an 86% higher return on such investments for the top half of firms (in terms of training investment) than for the bottom half of firms (Bassi, Ludwig, McMurrer, & Van Buren, 2000).

Software training requires a significant financial outlay. The most effective software training at present involves F2F behavior modeling, but such training is expensive to deliver. One possible way to reduce delivery costs is by offering similar software training, but through less expensive online delivery.

Allen and Seaman (2003) forecast that online learning would grow at a rate approaching 20% per year. The world corporate online learning market has been predicted to grow to nearly $24 billion by 2006 from $6.6 billion in 2002, an annual increase of 35.6% (International Data Corporation, 2002). The continuous growth of the online training market has prompted discussion about the effectiveness of Web-based virtual learning environments (Piccoli, Ahmad, & Ives, 2001).

While it is commonly agreed that online software training is less expensive and more flexible, it may also be less effective. Online software training continues to be of great interest to organizations, but significant challenges remain in implementing online solutions. These challenges include: (1) the cost of acquiring online learning systems, (2) the time for developing online learning materials, and (3) the need to be convinced of online learning's effectiveness compared to other training models (Bloom, 2004).

Three general training methods have been compared experimentally in F2F settings: instruction based, exploration based and behavior modeling. Instruction-based training occurs when trainers tell trainees about software, but do not model the use of it. Exploration-based training teaches trainees through practice by trainees on relevant examples, also without trainer modeling of software use. BM training teaches trainees via demonstrations, in which trainers model the use of software for trainees. Evidence exists that BM is the most effective method for F2F software training (Compeau & Higgins, 1995; Simon, Grover, Teng, & Whitcomb, 1996).

This research compares experimentally the relative effectiveness of F2F BM and online behavior modeling. Since prior research has indicated that the BM method dominates the instruction-based and exploration-based methods in F2F settings, this study does not include the latter two methods. Online asynchronous methods of software training, because they allow more favorable ratios of trainers to trainees and do not require training participants to meet, have the potential to achieve significant cost savings over F2F approaches. On the other hand, given that live trainers are not present in online asynchronous software training, there can be no direct interaction between trainers and trainees. This difference in direct interaction could mean that F2F training might be more effective than online training. Knowledge about the relative effectiveness of these methods will be valuable to people who must make decisions about how to provide software training.

THEORETICAL BACKGROUND

Software Training Method

As mentioned, three methods are common in F2F software training: exploration-based, instruction-based and BM (Simon et al., 1996). In exploration-based training, the assumption is made that learning is "a matter of rearranging or transforming evidence in such a way that one is enabled to go beyond the evidence so reassembled to additional new insights" (Burner, 1966, p. 22). Exploration-based training involves an inductive process through which individuals learn general concepts by trying to solve specific tasks (Taba, 1963). In instruction-based training, "the entire content of what is to be learned is presented to the learner in the final form" (Ausubel, 1963, p. 16). Instruction-based training is deductive and programmed, with low trainee control and a focus on software features (Davis & Davis, 1990). The BM method is in some ways a hybrid of exploration- and instruction-based training, and is centered on having trainees treat the behavior of their trainer as a model for their own (Simon et al., 1996).

When assessing the applicability of training methods for online asynchronous software training, researchers must bear in mind that some key elements of F2F software training may be lost in a movement to the online asynchronous setting. The opportunities for direct interactions between trainers and trainees are necessarily fewer, if they exist at all, in online asynchronous software training. Thus, the beneficial effects of trainer-trainee interactions typical of F2F software training may be missing. Videotapes, transcriptions, simulations or virtual reality are unlikely to serve as complete substitutes for live interactions between trainer and trainee. Along these lines, the features that distinguish BM training from exploration- and instruction-based training may perhaps be less evident in the online asynchronous software training situation.

Online asynchronous software training does not provide close monitoring of trainees by the trainer, as is common in F2F software training. This lack of monitoring can be expected to lead to increased levels of distraction among trainees. Trainees in online asynchronous software training might be inclined to attend to matters other than their training, such as surfing the Web, to a much greater extent than they would in F2F software training. Furthermore, online asynchronous software training may deliver content less tailored to trainees' interests than content provided in F2F software training. As a result, trainees may experience a higher degree of boredom in online asynchronous settings than in F2F settings, leading to poorer performance and more negative reactions to training experiences.

Online Training Modes

There are two temporal modes of online training: synchronous and asynchronous. Either mode can be used for software training. Text messaging, audio conferencing and video conferencing are examples of online applications that can be used for training purposes in a synchronous mode. Web pages, files to be downloaded, e-mail, newsgroups and discussion forums are examples of applications that can support training asynchronously.

Horton (2000) suggests that synchronous training and asynchronous training be designed differently. Synchronous training demands the control of schedule, time, people, class size, video and audio equipment, and place. These factors limit the possibilities for reaching trainees in a cost-effective manner. This study focuses on online asynchronous software training in part to avoid the influence of these factors, and in part to concentrate on the methods most likely to keep costs low. This choice reflects the reality that most online training delivered across continents is provided in the asynchronous mode and that use of the online asynchronous mode is destined to grow.

Behavior Modeling in Face-to-Face and Online Asynchronous Software Training

Social cognitive theory (Bandura, 1986) serves as a theoretical basis for behavior modeling. According to this theory, "most human behavior is learned by observation through modeling" (p. 47). Observational learning allows one to form rules to guide future behavior by watching what others do and noting what consequences such behavior has for them. Further, observational learning can make use of symbolic models, allowing people to consider words and images in coming to appreciate what happens to others when they behave in particular ways, thereby extending what can be learned beyond the immediate environment. Learning by modeling involves four kinds of processes: (1) attention, (2) retention, (3) production, and (4) motivation (Bandura, 1986). These

Table 1. F2F BM vs. online behavior modeling: strengths and weaknesses

	BM Approaches	
	F2F behavior modeling	Online behavior modeling
Strengths	• Regardless of learning styles, trainees are motivated to use the training approach. • The approach is less influenced by learning style and is the most effective. • The teaching quality is contingent on both the quality of entire course materials packaged in final form and supplementary materials (e.g., handouts). • Trainer's teaching abilities and trainees' participation are major determinants for training outcomes. • Real-time two-way feedback. • Both trainers and trainees have equal power to control the learning pace.	• Regardless of learning styles, trainees may be motivated to use the training approach. • The approach may be less influenced by learning style and should be an effective training approach. • Teaching quality may be contingent on both the quality of entire course materials packaged in final form and supplementary materials on the Internet. • Trainer's teaching abilities and trainees' participation may be major determinants for the training outcomes. • Web-based peer-to-peer interaction may improve commitment and participation. • Both trainers and trainees may have equal power to control the learning pace.
Weaknesses	• Needs longer time to exercise the training approach. • Constrained by the length of time, a trade-off between instruction and exploration learning is needed. • A trainer's experience will influence the decision quality of the trade-off, thereby affecting trainees' motivation and their learning outcomes.	• Trainees may have trouble modeling the behavior of the trainer without direct interaction. • Demonstrative lecturing may need to rely on videotaped or scripted course materials. • Asynchronous, one-way communication may lower motivation, thereby degrading learning outcomes. • Real-time reiterative learning process to confirm understanding may be lost.

processes are influenced both by characteristics of the events observed and characteristics of the observer.

Learning by modeling or observing people's behaviors may be more effective than learning by trial and error, because the former approach can avoid unnecessary mistakes and harm. Modeling a trainer's behavior empowers trainees to (1) learn new behavior from the trainer, (2) self-evaluate their behavior against the trainer's, and (3) reinforce their current adequate behaviors. The BM approach is different from learning by adaptation. The former approach teaches via demonstration, while the latter approach influences the behaviors of learners by reward and punishment (Skinner, 1938).

An early application of BM training was in the area of interpersonal communication and management skills (Decker & Nathan, 1985). In the realm of training for software and computer usage, BM training has been shown consistently to be more effective than instruction- or exploration-based training (Compeau & Higgins, 1995; Gist, Schwoerer & Rosen, 1989; Simon, Grover, Teng & Whitcomb, 1996). Yi and Davis (2001) found that the effects of BM could be enhanced by the provision of training features to support retention enhancement and practice.

BM is readily employed in F2F training, but may be difficult to apply in online settings, which may be less suited to demonstrations of behavior. Limitations of the media typically used in online synchronous instruction in terms of their richness constitute one possible constraint. Another is a reduced level of reinforcement possible in online settings, compared with F2F instruction. For example, in a live training class, the trainer is able to demonstrate a software process and immediately ask the trainees to repeat the activity under the trainer's close supervision. However, in an online asynchronous situation, in which there is no live trainer, demonstrations lose the benefit of immediate feedback. In an online synchronous situation, bandwidth constraints and compromised reciprocity may undermine the effectiveness of demonstrations. In online environments, effectiveness can be further compromised with the absence of learning by doing, another key element of F2F BM training (McGehee & Tullar, 1978). Table 1 summarized the strengths and weaknesses of F2F BM and online behavior modeling.

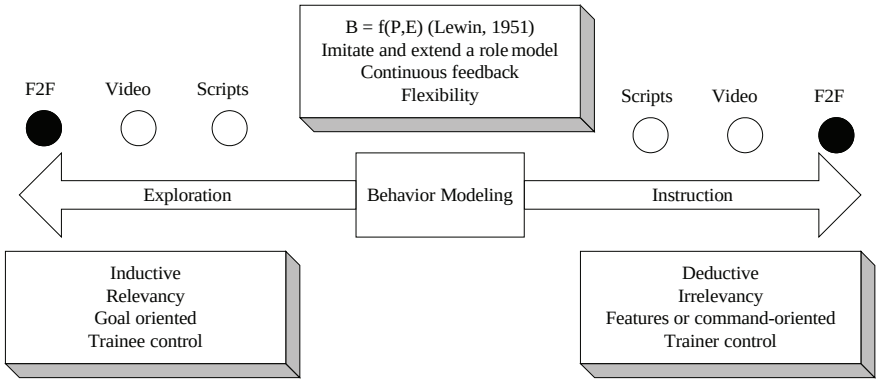
Trying to use BM for online software training would involve the issues mentioned above. Therefore, there is a strong possibility that the BM approach cannot be fully replicated in the online asynchronous setting and, therefore, will not be as effective in the online environment as in the traditional environment. On the other hand, if online BM were to prove as effective as F2F behavior modeling, organizations would prefer it, because it would cost less and promise higher returns on investment. The increasing use of online asynchronous software training by businesses and schools raises questions: What training methods should be used under what circumstances in the online asynchronous environment?

RESEARCH MODEL AND HYPOTHESES

This study tested the relative efficacy of BM training, in two different environments: experimentally and in a field setting. Experimentation can allow testing of causal relationships among variables. A field setting can provide greater confidence in the meaning of experimental tasks than in a laboratory setting. A field experiment methodology has the merits of being able to test theory and answer practical questions (Kerlinger & Lee, 2000).

The independent variable for the experiment was *training method*, set at two levels of behavior modeling – F2F and online. Training materials were designed to operationalize each level by integrating key elements of BM training, as illustrated in Figure 1.

Figure 1. Key elements of behavior modeling training



The online BM treatment was designed to be video-demonstration oriented. Trainees watched a recording of the trainer using the software to perform tasks. All communication between the trainer and trainees was one-way. Following the video, trainees completed tasks to demonstrate their level of mastery of the training materials. Online reference sources were available to them as they completed the tasks. Control over what occurred was shared between the system and trainees. Online behavior modeling, as conceived here, shares features with both exploration and instruction approaches, including both inductive and deductive aspects.

The F2F BM treatment was designed to be direct-demonstration oriented. Trainees watched the trainer performing tasks in person. Two-way communication between the trainer and trainees was possible. Following the demonstration, trainees performed a task assigned by the trainer. No online reference sources were available. Control over what occurred was shared between the trainer and trainees. The F2F condition included a trainer (other than one of the researchers, to reduce chances of awareness of the hypotheses being tested). The trainer followed the same script as the one designed for the online condition.

The length of time allotted to training was the same for both conditions. Both conditions included the same pre- and post-training tests. All training sessions were conducted in the same computer classroom.

Dependent Variables

Regardless of teaching environment, most training is intended to instill a competency of some kind. Software competency depends on the kind of knowledge acquired in training. Learning effectiveness can be evaluated through trainees' reactions and knowledge transfer (Kirpatrick, 1967). Knowledge levels can be categorized as near transfer or far transfer (Simon et al., 1996). Near transfer of knowledge is necessary for understanding of basic software commands and procedures. Far transfer of knowledge allows the solving of problems different from those worked out in training. The measurement of near-transfer knowledge involves direct assessment of what was learned about the specific objects (such as software

features and commands) covered in training. The measurement of far-transfer knowledge has to do with evaluating the extent to which what was learned is available to trainees in their solution of problems similar to those included in training. For far-transfer knowledge, it is interesting to assess learning both immediately after training and after some delay, because software competency is intended to be a long-term effect of training.

In this study, software competency was measured in terms of near-transfer knowledge, far-transfer knowledge (assessed immediately) and far-transfer knowledge (assessed with delay). The measure of *near transfer (NT)* consisted of 10 multiple-choice questions concerning details of the software covered in training. *Immediate far transfer (IFT)* was measured with a problem to be solved with the software during the experimental session. *Delayed far transfer (DFT)* was measured with a problem administered later in the academic term as part of the final exam.

In addition to how well it instills software competency, software training should also be judged by the reactions that trainees have to it. It is common to use satisfaction as a surrogate for the effectiveness of information systems (Ives, Olson, & Baroudi, 1983) and it has been adopted as an indicator of success in software training (Simon et al., 1996). Perceived usefulness and perceived ease of use have been shown to predict attitudes and behaviors with information systems (Davis, Bagozzi, & Warshaw, 1989).

In this study, the reactions of trainees were captured through three measurement scales: *satisfaction (SAT), perceived usefulness (PU)* and *perceived ease of use (PEOU)*. SAT, PU and PEOU were measured with scales administered during the experimental session, as described below.

Hypotheses

As discussed above, the online BM approach replaces the live instructor with the scripted demonstration, and some key elements of the F2F BM approach may be lost. Characteristics of the online asynchronous environment—particularly limitations on trainer-trainee interactions—suggest that BM training done in a F2F manner should be superior to BM training done in the online mode. BM training in the F2F mode may be more effective at improving the learning outcome for a trainee than the BM approach in the online asynchronous mode. Proving that the F2F BM is more or less effective than online BM could justify the validity of replicating the same pattern in the online asynchronous environment. Hence, it is only hypothesized that the F2F BM approach is more effective than the online BM to improve learning outcomes for trainees of all learning styles. The hypotheses listed below correspond to this expectation with respect to NT, IFT, DFT, SAT, PU and PEOU.

- H1: NT scores will be greater with F2F BM than with online BM.
- H2: IFT scores will be greater with F2F BM than with online BM.
- H3: DFT scores will be greater with F2F BM than with online BM.
- H4: SAT scores will be greater with F2F BM than with online BM.
- H5: PU scores will be greater with F2F BM than with online BM.
- H6: PEOU scores will be greater with F2F BM than with online BM.

Trainees and Setting

The setting for the experiment was an introductory computer course requiring trainees to learn spreadsheet software (Excel 2000). Trainees in this course were freshmen, majoring in management information systems or accounting. Participation in the study was voluntary. The faculty for the course agreed to run the experiment near the end of the academic term. As a result of this timing,

none of the trainees in the experiment were complete novices with respect to computers; all had some literacy and experience.

It was not desirable in this study for experimental trainees to have prior knowledge of spreadsheets. Therefore, as part of the experimental procedure, a pre-test of 10 multiple-choice questions about Excel 2000 was given to each trainee; trainees who scored higher than 50% were excluded.

Training Procedure

The trainer conducted the traditional F2F BM by following the same procedures as delineated in the online BM training package (Figure 2). The first stage of the package provides examples of how to manage a database using Excel 2000. The second stage lists online asynchronous training tools that can be used to assist the trainee. These tools are: (1) examples that are relevant to trainees'

Figure 2. F2F BM and online BM procedures

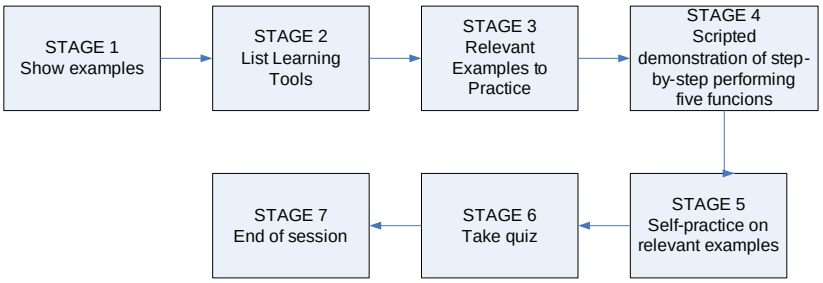
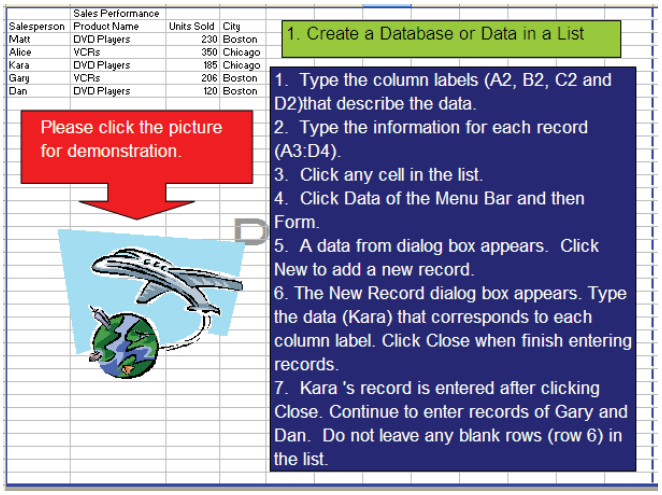


Figure 3. Scripted demonstration of step-by-step, performing five functions: The fourth step of online BM method (enabled by Excel's macro functions)



backgrounds, (2) a self-practice worksheet for each function of database management, and (3) online reference sources. The third stage allows trainees to choose relevant examples to use for practice throughout the training session.

Two examples were prepared for each major. Trainees were encouraged to practice with examples relevant to their majors. The training covered five database management features of Excel. These features allow one to (1) create a database without using a data form, (2) create a database with a data form, (3) sort data in a data list, (4) filter data in a data list, and (5) add subtotals to a data list. Step-by-step instructions were adopted to illustrate each function.

Additionally, trainees could choose to watch or not to watch a demonstration for all database functions. On the same page as the demonstration, a self-practice worksheet was presented. Practice results could also be carried over to the next practice. Trainees of the online BM group took the same quizzes as the F2F group and then concluded the study. Figures 3 and 4 are two screenshots of course materials used for the online BM approach. The demonstration of a live instructor was substituted for fixed-time (12 seconds) transition of page presentations. Each scripted demonstration focused on one particular function of the database management topic. At each learning point of a particular function, trainees were encouraged to experiment by practicing the related exercises.

RESULTS

A total of 114 trainees completed the study by submitting valid questionnaires. This amounted to approximately 46% of the trainees who had been registered for the course. Although a higher rate of participation in the experiment was expected, the achieved rate may have been due to the experiment being a voluntary activity at the end of an academic term. Of the trainees who did participate, 31 had too much prior spreadsheet knowledge to qualify as trainees for the experiment, leaving 83 trainees whose data were analyzed.

Table 2 shows the number of trainees for each experimental treatment. Note that the counts of trainees in each condition, although not exactly the same, are similar enough not to cause analytic difficulties. The sample size, although not

Figure 4. The first step of scripted demonstration for the first subject: Sort data in a list

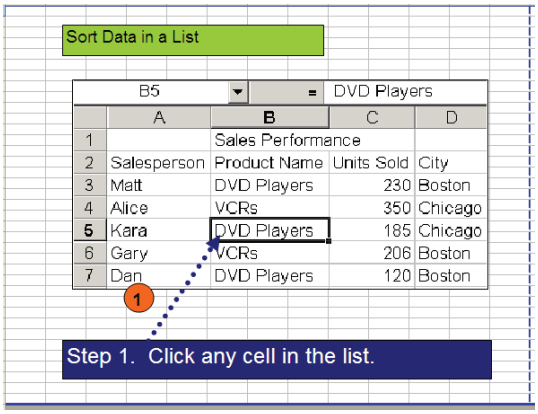


Table 2. Number of trainees by treatments

Training Approaches	*Total Completing Study*	*Excess Prior Knowledge*	*Number for Data Analysis*
Online	*44*	*9*	*35*
F2F	*70*	*22*	*48*
Total	*114*	*31*	*83*

as large as might be desired, also is not a source of analytic problems.

Multivariate analysis of covariance (MANCOVA) has advantages over analysis of variance (ANOVA) in removing some systematic errors and uncontrolled individual differences. Although the researchers planned to adopt MANCOVA, a cursory investigation showed no interaction effect between two dependent variables—user satisfaction and learning performance. This indicates that learning performance and satisfaction effects are separated; interaction among them is not the issue in this study. The possibility of inflating type I error due to the analyses of multiple univariate ANOVAs is minimum. Furthermore, due to the small sample size of some cells and the unequal cell sizes, complying with the assumptions of two-way ANOVA can improve the reliability of data analysis. Each dependent variable was treated and analyzed independently with ANOVA as a result. Data analysis using a histogram graph shows that each dependent variable complies with univariate

Table 3. Descriptive statistics by treatments

Dependent Variable	*Mean/Standard Deviation*	*Online BM*	*F2F BM*
NT	*mean*	*45.33*	*49.38*
	s.d.	*19.43*	*16.94*
IFT	*mean*	*39.33*	*25.00*
	s.d.	*24.90*	*18.22*
DFT	*mean*	*36.00*	*44.63*
	s.d.	*39.23*	*36.41*
SAT	*mean*	*2.86*	*3.05*
	s.d.	*0.61*	*0.63*
PU	*mean*	*2.72*	*2.97*
	s.d.	*0.69*	*0.62*
PEOU	*mean*	*3.02*	*2.93*
	s.d.	*0.32*	*0.34*

Table 4. Results for training methods

Variable	*Hypothesis*	*Result in Correct Direction?*	*Significant p-value?*
NT	*Face-to-face > online*	*T*	*n.s.*
IFT	*Face-to-face > online*	*F*	*n.s.*
DFT	*Face-to-face > online*	*T*	*n.s.*
SAT	*Face-to-face > online*	*T*	*n.s.*
PU	*Face-to-face > online*	*T*	*n.s.*
PEOU	*Face-to-face > online*	*F*	*n.s.*

normality assumptions. Additionally, Levene's test at $\alpha = 0.10$ shows that the null hypothesis, *the error variance of the dependent variable is equal across groups,* is not violated. The tests indicate that the data is normally distributed. An ANOVA was conducted for each dependent variable. ANOVA is robust for situations having a limited number of data points (Moore & McCabe, 1989). Table 3 provides descriptive statistics for dependent variables by experimental treatment. Table 4 summarizes ANOVA results for training methods in terms of learning outcomes. Direction and significance of differences between treatments are indicated.

From a hypothesis-testing standpoint, four out of six hypotheses, all concerning the superiority of F2F BM over online BM (H1, H3, H4, and H5), were in the direction hypothesized. Despite this, these hypotheses were not supported in a statistical sense.

DISCUSSION

Although none of the hypothesized relationships are fully supported, the results obtained are interesting. The most intriguing result is that—contrary to expectations—there are no statistical reasons for preferring F2F BM to online BM for software training of this kind. The pattern of results indicates that while F2F BM results in better outcomes than online BM for four of the six dependent variables, it never does so at a statistically significant level. One interpretation of this is that online BM training is no worse than F2F BM training across all dependent variables. (Trainees in the online BM condition actually score higher than F2F BM trainees in IFT.) The pattern of results for F2F BM suggests that trainers might choose online BM—which ought to be a less costly alternative to F2F BM—without making any significant sacrifice in either learning or trainee reaction outcomes. The complex picture of the implications of these four treatments must be more clearly illustrated.

Two methods can illustrate this complexity. The first is the "insufficient difference" finding between online BM and F2F BM. The second is the beginning of a strategy for online asynchronous software training. As a first result, the conclusion of "insufficient difference" between online BM and F2F BM depends on being able to say there is not enough difference between their effects to justify the difference in their costs. A practical difference between F2F BM and online BM—one that matters in cost/benefit terms—must have some minimum size. Specifying a practical difference involves knowing the costs of F2F BM

and online BM, as well as how effect size maps to benefits.

Ability to detect effect sizes is nothing more than statistical power (Cohen, 1977). In information systems research, "studies are unlikely to display large effects and that, typically, small- to medium-effect sizes should be anticipated" (Baroudi & Orlikowski, 1989, p. 90). Because this study exercised due care with experimental procedure and made use of reliable instruments, there is justification in addressing statistically insignificant results. Before executing the experiment, efforts were made to maximize the difference between F2F BM and online BM conditions; a Delphi study was conducted regarding the design of course materials to reflect the different training approaches. Despite this careful control over operationalization, there was not enough difference between F2F BM and online BM effects to justify the difference in their costs.

Due to the undeveloped nature of research in this area, it may be inappropriate to establish an index for effect size based on prior research on software-training strategy in the traditional environment (Mazen et al., 1987). To explain the phenomenon carefully, we employ Cohen's (1977) approach to estimate proxy effect-size levels based on the standardized difference d of two populations taking different training approaches (see Table 5).

Table 5. Effect size estimation

Dependent Variables	Standardized Difference between F2F BM and online BM	* Estimated Effect Size
Knowledge NT (KNT)	25.47%	0.20
Knowledge FT (KFT)	-46.45%	0.50
OS	58.82%	0.50
PEOU	-21.05%	0.20
PU	57.97%	0.50

** Calculated based on Cohen's (1977) Effect Size Conventions*

The estimated effect size is 0.5 for all dependent variables except KNT and PEOU. Since PEOU is related to the design of the e-learning system rather than the treatment of training approach, smaller effect size across different groups is understandable. However, the study cannot detect differences of effect size for KNT. This indicates that it makes no difference whether F2F BM or online BM is employed to improve KNT. Online trainers can choose either F2F BM or online BM to improve end-users' KNT if either approach has relatively similar costs.

Contrary to the expectation of hypotheses, a larger effect size was detected for KFT in the short term and long term. This practical difference indicates that the benefits of online BM outweigh F2F BM for KFT in the short and long term. Larger effect size was also detected for the measures of end-user satisfaction: OS and PU. This practical difference supports that F2F BM is a better approach than online BM to improve end-user satisfaction. The difference between knowledge absorption capability and end-user satisfaction poses many interesting questions.

As a second result, this study can offer concrete suggestions about the beginning of a strategy for online asynchronous software training. One result of interest is that F2F BM might be better than online behavior modeling. Of the six hypotheses concerning relationships between these two methods, four are in the expected direction, none significantly so.

These findings indicate that use of online BM may be the best software-training strategy for the online asynchronous setting. To confidently offer such suggestions, the study needs to discuss the design decisions that trainers face in the online asynchronous environment. The study provides support for using online BM over exploration- and

instruction-based training, given that the prior contribution makes the point of favoring online BM over F2F BM. Since our suggestions are a start on an online asynchronous software training strategy, we will present the outline of the strategy that includes "to-do" and "not-to-do" lists. This online asynchronous software training strategy will allow trainers and vendors to capitalize on these opportunities and avoid costly mistakes.

The largest implication for practice is that online BM may provide a cost-effective substitute for F2F BM without significant reductions in training outcomes. Compared to F2F BM, online BM allows trainees to have more control over their learning. Cognitive learning theory indicates that the learning process can be improved via active learning (Shuell, 1986) and problem-solving learning (Alavi, 1994). In the virtual learning environment (VLE), trainees have higher control of learning and can choose to use exploration- or instruction-based training approaches depending on tasks and individual needs. For instance, trainees with more experience and knowledge related to a particular trainee may resort to meaningful learning and use relevant examples to practice. Trainees with little knowledge about another trainee may resort to rote learning and use generic examples to practice. The VLE allows trainees to switch freely between meaningful and rote learning, to their advantages.

Since trainees have control flexibility, online BM can be viewed as more effective than the F2F BM in helping trainees perform well on near-transfer and far-transfer tasks. In the VLE, the individualized and contextual learning provides anchoring for trainees to transform their mental models. While more must be learned about this relationship, it is encouraging to see evidence that there may be a desirable leverage from online asynchronous software training.

Another thing trainers need to bear in mind when designing an online asynchronous software training strategy is that the effectiveness of online asynchronous software training methods does not necessarily go hand-in-hand with overall satisfaction, perceived ease of use and perceived usefulness. In particular, it may still be the case that learning effectiveness is neutral to learning style. Improving satisfaction by customizing learning approaches may be the right decision to make, but performance might not be the deciding factor.

Online BM and F2F BM allow trainees to have some control of the learning process and information acquisition regarding its content, accuracy, format, ease of use and timeliness (Doll, Xia, & Torkzadeh, 1994), which leads to somewhat higher satisfaction levels. In itself, higher levels of satisfaction may be justification for online BM and F2F BM use, but much remains to be learned about the effects of these methods for training performance.

Assimilation Theory suggests that being receptive to new procedural knowledge on how to operate a new target system is the prerequisite to meaningful learning (Mayer, 1981) or far-knowledge transfer. With the time constraints, a more focused learning approach can be useful at assimilating new knowledge. Hence, the online BM approach is a logical solution for meaningful learning, because the approach allows trainees to not only acquire new knowledge, but also gives them flexibility to search their long-term memory for "appropriate anchoring ideas or concepts" (p. 64) and to use the ideas to interact with new knowledge (Davis & Bostrom, 1993).

LIMITATIONS

While it seems unlikely, given the care taken in the design of the study, there is always the possibility that the independent variable, training method, could inadequately represent its intended construct. With any complex treatment, such as the establishment of a training method here, there is a chance that operationalization can be less than what is needed for effects to occur. Additional re-

search is required to refine and perfect the training method treatments as much as possible. There is no simple manipulation check for verifying the efficacy of this kind of treatment, but continued investigation should reveal the extent to which the manipulation is a successful one.

Future research can attempt to improve the reliability of the findings by controlling the experimental environment more tightly (e.g., equal cell size, larger cell size and longer training sessions) or by improving the strategy's generalizability through the examination other variables (e.g., trainees vs. professional workers, number of training duration sessions, type of training media, self efficacy, experiences of using the online learning system and software types).

IMPLICATIONS FOR RESEARCH

The findings here raise additional questions for research. Some that might be addressed in the immediate future include:

- To replicate the experimental equivalence of F2F BM and online BM methods of software training with different software and trainees. With this, to demonstrate a practical (i.e., cost-based) advantage of online BM over F2F BM for software training in practical settings.
- To study the impact of training duration on performance and trainee reactions. Trainees should be exposed to the same training methods for different durations.
- To improve the reliability of the study by manipulating some useful blocking variables. A series of comparative studies can be conducted to assess the impact of individualism as a cultural characteristic, computer self-efficacy, task complexity, professional backgrounds, and the ratio of the training duration to the quantity of information to be processed, among others.
- To investigate the impacts of social presence and information richness (SPIR) (Fulk, 1993) features of online asynchronous software training media on training outcomes. Future studies might vary the SPIR features of training media (e.g., F2F vs. online asynchronous scripted or Web cam modes).
- To conduct longitudinal studies of the influence of learning style on learning performance and trainee reaction.
- To continue to study the relationship between learning style, training methods and training outcomes. Learning style is somewhat associated with the cultural backgrounds of online trainees. Trainees with varying cultural backgrounds may prefer to adopt training media with different SPIR features. Cultural differences, such as relative degree of individualism, may affect preference for SPIR characteristics. Some combination of training methods, learning style and SPIR attributes may jointly determine learning outcomes.

Implications for Practice

The largest implication for practice is that online BM may provide a cost-effective substitute for F2F BM without significant reductions in training outcomes. While more must be learned about this relationship, it is encouraging to see evidence that there may be a desirable leverage from online asynchronous software training. Also, when designing an online asynchronous software training strategy, trainers need to bear in mind that both F2F BM and online BM are equally effective to improve learning outcomes (including satisfaction), and performance might not be the decision factor if these two approaches need to be chosen from. Other decision factors, such as trainer's preference, equipment availability, budget and scheduling, could be more important than the efficacy issue. Online BM and F2F BM allow trainees to have some control of

the learning process, leading to somewhat higher satisfaction levels. This advantage in itself may be justification for their use, but much remains to be learned about the effects of these methods for training performance.

CONCLUSION

The success of an online asynchronous software training strategy depends on its effectiveness in improving learning outcomes. This study builds on a well-accepted framework for training research (Bostrom, Olfman & Sein, 1990; Simon, Grover, Teng & Whitcomb, 1996), examines the relative effectiveness of four training methods and begins to derive a strategy for online asynchronous software training. Testing the following hypotheses provides an empirical basis for the development of an online asynchronous software training strategy: (1) F2F BM is more effective than online BM for learning performance and trainee reactions, and (2) online BM is more cost effective than F2F BM.

While these hypotheses are not fully supported statistically, and while many of the observed results are difficult to interpret, the study discovers important potential implications for practitioners and researchers. The formulated online asynchronous software training strategy suggests that trainers customize their training methods based on desired learning outcomes.

What is learned from this study can be summarized as follows: When conducting software training, it may be as effective to use an online BM method as it is to use a more costly F2F BM method. Although somewhat better results are sometimes evident for F2F BM, observed differences are not significant, nor are their patterns consistent.

The study has accomplished its major goal—it provides evidence as to the relative effectiveness of various methods, particularly those of an online asynchronous nature, for software training. Within its limits, this research takes a first step in developing a strategy for online asynchronous software training.

REFERENCES

Allen, I. E. & Seaman, J. (2003). *Sizing the opportunity: The quality and extent of online education in the United States, 2002-2003*. Needham, MA: The Sloan Consortium.

Ausubel, D. P. (1963). *The psychology of meaningful verbal learning.* New York: Grune and Stratton.

Aytes, K., & Connolly, T. (2004, July-September). Computer Security and Risky Computing Practices: A Rational Choice Perspective. *Journal of Organizational and End User Computing 16*(3), 22-40.

Bandura, A. (1986). *Social foundations of thought & action*. Englewood Cliffs: Prentice-Hall.

Baroudi, J. J., & Orlikowski, W. J. (1988). A short form measure of user information satisfaction: A psychometric evaluation and notes. *Journal of Management Information Systems, 4*, 45-59.

Bassi, L.J., Ludwig, J., McMurrer, D.P. & Van Buren, M. (2000, September). *Profiting from learning: Do firms' investments in education and training pay off?* The American Society for Training and Development (ASTD) and Saba, Alexandria, VA. Retrieved January 3, 2006, from http://www.astd.org/NR/rdonlyres/15C0E2AB-B16D-4E3C-A081-4205B865DA3F/0/PFLWhitePaper.pdf

Bell, J. (2004, March/April). Why software training is a priority? *Booktech the Magazine, 7*, 8.

Bielefield, A., & Cheeseman, L. (1997). *Technology and copyright law.* New York: Neal-Schuman Publishers.

Bloom, M. (2003, April 2). *E-learning in Canada, Findings from 2003 E-survey: Top line findings from a survey of the conference board of Canada's customers on current e-learning practices.* The Conference Board of Canada. Retreived June 6, 2005, from http://www.conferenceboard.ca/education/reports/pdfs/TopLine_report.pdf

Brown, J. (2000, December 15). Employee turnover costs billions annually. *Computing Canada, 26,* 25.

Bruner, J. (1966). *Toward a theory of instruction.* New York: Norton.

Cohen, J. (1977). *Statistical power analysis for the behavioral sciences.* New York: Academic Press.

Compeau, D. R. & Higgins, C. A. (1995). Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly, 19,* 189-211.

Compeau, D. R., Higgins, C. A., & Huff, S. (1999). Social cognitive theory and individual reactions to computing technology: A longitudinal study. *MIS Quarterly, 23*(2), 145-158.

Davis, D. L., & Davis, D. F. (1990). The effect of training techniques and personal characteristics on training end users of information systems. *Journal of Management Information Systems, 7*(2), 93-110.

Davis, F. D. (1989, September). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly,* 319-339.

Decker, P. J. & Nathan, B. R. (1985). *Behavior modeling training.* New York: Praeger.

Fulk, J. (1993). Social construction of communication technology. *Academy of Management Journal, 36,* 921-950.

Gist, M. E., Schwoerer, C., & Rosen, B. (1989). Effects of alternative training methods on self-efficacy and performance in computer software training. *Journal of Applied Psychology, 74*(6), 884-891.

Heller, M. (2003, November 15). Six ways to boost morale. *CIO Magazine, 1.*

Horton, W. (2000). *Designing Web-based training.* New York: John Wiley & Sons.

International Data Corp. (2002, September 30). *While Corporate Training Markets Will Not Live up to Earlier Forecasts, IDC Suggests Reasons for Optimism, Particularly eLearning.* International Data Corporation. Retrieved March 5, 2003, from http://www.idc.com/getdoc.jhtml?containerId=pr2002_09_17_150550

Ives, B., Olson, M., & Baroudi, S. (1983). The measurement of user information satisfaction. *Communications of the ACM, 26,* 785-793.

Kerlinger, F. N., & Lee, H. B. (2000). *Foundations of behavioral research.* New York: Harcourt Brace College Publishers.

Kirpatrick, D. L. (Ed.). (1967). Evaluation of training. *Training and development handbook.* New York: McGraw Hill.

Leidner, D. E., & Jarvenpaa, S. L. (1995). The use of information technology to enhance management school education: A theoretical view. *MIS Quarterly, 19,* 265-291.

Mazen, A. M., Graf, L. A., Kellogg, C. E., & Hemmasi, M. (1987). Statistical Power in Contemporary Management Research. A*cademy of Management Journal, 30,* 369–380.

McEvoy, G. M., & Cascio, W. F. (1987, December). Do good or poor performers leave? A meta-analysis of the relationship between performance and turnover. *Academy of Management Journal, 30*(4), 744-762.

McGehee, W., & Tullar, W. (1978). A note on evaluating behavior modification and behavior modeling as industrial training techniques. *Personal Psychology, 31,* 477-484.

Piccoli, G., Ahmad, R., & Ives, B. (2001). Web-based virtual learning environments: A research framework and a preliminary assessment of effectiveness in basic IT skills training. *MIS Quarterly, 25*(4).

Simon, S. J., Grover, V., Teng, J. T. C., & Whitcomb, K. (1996, December). The relationship of information system training methods and cognitive ability to end-user satisfaction, comprehension, and skill transfer: A longitudinal field study. *Information Systems Research, 7*(4), 466-490.

Skinner, B. F. (1938). *The behavior of organisms: An experimental analysis.*New York: B. F. Skinner Foundation.

Taba, H. (1963). Learning by discovery: Psychological and educational rationale. *Elementary School,* 308-316.

Wexley, K. N., & Baldwin, T. T. (1986). Posttraining strategies for facilitating positive transfer: An empirical exploration. *Academy of Management Journal, 29,* 503-520.

Yi, M. Y. & Davis, F. D. (2001). Improving computer training effectiveness for decision technologies: Behavior modeling and retention enhancement. *Decision Sciences, 32*(3), 521-544.

Yi, Y. (Ed.). (1990). *A critical review of consumer satisfaction: Review of marketing.* Chicago: American Marketing Association.

This work was previously published in the International Journal of Web-Based Learning and Teaching Technologies, edited by L. Esnault, Volume 1, Issue 4, pp. 36-53, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 5.26
Lack of Skill Risks to Organizational Technology Learning and Software Project Performance

James Jiang
University of Central Florida, USA

Gary Klein
United States Air Force Academy, USA

Phil Beck
Southwest Airlines, USA

Eric T. G. Wang
National Central University, Taiwan

ABSTRACT

To improve the performance of software projects, a number of practices are encouraged that serve to control certain risks in the development process, including a lack of essential skills and knowledge related to the application domain and system development process. A potential mediating variable between the lack of skill risk and project performance is the ability of an organization to acquire the essential domain knowledge and technology skills through learning, specifically organizational technology learning. However, the same lack of knowledge that hinders good project performance may also inhibit learning. This study examines the relationship between information system personnel skills and domain knowledge, organizational technology learning, and software project performance with a sample of professional software developers. Indications are that the relationship between information systems (IS) personnel skills and project performance is

partially mediated by organizational technology learning.

INTRODUCTION

The importance of technical and business skills and knowledge for information systems personnel has been advocated in the IS literature for decades (Cheney & Lyons, 1980; Jiang, Klein, Van Slyke, & Cheney, 2003). In spite of the recognized importance, empirical investigations that examined the relationship between IS personnel skills and project performance, as measured comprehensively by attainment of goals and budgets, have been lacking in the IS research literature. To address this lack, Byrd and Turner (2001) conducted a study using the perceptions of chief information officers (CIOs) to evaluate IS personnel skills and the success of information systems in building competitive posture. To one's surprise, their study did not find a significant positive relationship between IS personnel skills and eventual success of the developed system. Why should an empirical study contradict experience? They suspect that the relationships in an organization where IS personnel skills are applied have too many complexities to be modeled accurately. Could there be a mediating variable between IS personnel skills and IS project performance that further explains how to overcome this essential lack?

Researchers have observed that activities during information system development and implementation offer an opportunity for organizational technology learning, or the ability and practice of bringing new skills and knowledge into the organization related to IS development and the application of IS tools to business domains (Ko, Kirsch, & King, 2005; Stein & Vandenbosch, 1996). For a successful IS implementation, skills must be brought to bear from the application domain and technical domains, which can best happen when the organization encourages the learning of newer skills and knowledge, and has practices to incorporate these newly acquired assets into current and future projects. In short, organizational technology learning is a critical factor for predicting final IS project performance, and a base of knowledge and skills in the IS project team are a necessary condition for organizational technology learning to occur. This suggests that organizational technology learning is a mediator between IS skills and knowledge and the performance of the IS project. Unfortunately, no empirical study has attempted to validate this reasoning.

The focus of this study is, therefore, to examine the relationship from IS staff development skills and domain knowledge to project performance with organizational technology learning as a mediator. A positive result of this study will provide additional insights on IS skill research and provide an alternative explanation to the unsolved IS skills puzzle of Byrd and Turner (2001). From a survey sample of 212 Institute of Electrical and Electronics Engineers Computer Society members, the results indicate that the lack of system development skills and knowledge in the application domain have a direct negative impact on organizational technology learning and project performance. Furthermore, organizational technology learning has a significant positive impact on final project performance, showing that the impact of IS personnel skill levels on project performance is partially mediated by organizational technology learning.

BACKGROUND AND RESEARCH MODEL

Broad categories of critical IS personnel skills are identified, including (1) technical specialties/ technology management skills and (2) business domain knowledge and skills (Jiang et al., 2003). Unfortunately, given decades of emphasis, these IS skills were still not linked to IS project performance (Byrd & Turner, 2001). This may be

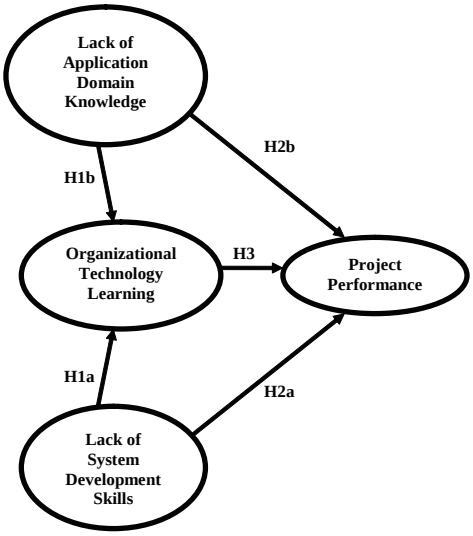
due to the lack of an intervening variable similar to an established relationship between IS staff competency and firm performance where learning is a mediator (Tippins & Sohi, 2003). This study investigates the possibility of a variation on learning as a mediating variable in the project context between IS personnel knowledge and skills and IS project success.

This research considers learning from the perspective of general technology skills acquired by the firm in the project context (Cooprider & Henderson, 1990). Organizational technology learning is an organization's furthered understanding of its operational business procedures and information system technology capabilities (Lee & Choi, 2003). The IS application development process itself can be viewed as an intensive activity of skill learning and knowledge acquisition (Rus & Lindvall, 2002). Such learning results in associations, cognitive systems, and memories that are developed and shared by members of an organization. These learned skills can then be used to enhance the performance of a software project and, thus, the organization. From concept generation to implementation, integration of skills and knowledge to achieve a desired product can be viewed as the central theme of the software development process. IS projects require such an in-depth set of skills and knowledge of application domains (Badaracco, 1991).

Various project stakeholders (e.g., IS developers and end-users) bring different repositories of expertise, skills, knowledge, and perspectives to the project teams. This potential pool of skills held collectively by project stakeholders represents *embedded* knowledge (Schramer, 2000). The delivered system represents *embodied* (i.e., learned) knowledge – referring to how the technical knowledge and knowledge of application domain is formalized and incorporated in the design, functionality, policies, and features of a system. Okhuysen & Eisenhardt (2002) indicated that the successful integration and learning of embedded knowledge into embodied knowledge is the key to successful product and service developments. The information system development process is a process of converting embedded knowledge into embodied knowledge (Robillard, 1999). Proposed organizational practices to achieve this conversion can be found in various literature on knowledge management, organizational learning, and employee education (Chiva & Alegre, 2005; Kayes, Kayes & Kolb, 2005). In particular, practices have been proposed to enhance knowledge management via information systems, promote learning during development, and educating through training (Majchrzak, Beath, Lim, & Chin, 2005; Olfman, Bostrom, & Sein, 2003; Sher & Lee, 2004). As a result, a relatively stable and accessible body of embedded knowledge that exists within the project team may become a necessary condition for an occurrence of organizational technology learning and improved project performance.

The notion of organizational entities, like project teams as vehicles for integrating fragmented knowledge, is the core of knowledge-based theory (Grant, 1996). Some knowledge is codified in documents or embodied in procedures and policies, and much is held tacitly in the minds of individuals. The development of new products and services require combining both tacit and explicit knowledge held by many individuals (Kogut & Zander, 1992). Many of the "run-away" IS projects can be related to either embedded customer or technical knowledge that are difficult to be embodied in a system: vague requirements, new technologies/methods, new development methodologies, new ideas, and changing user needs. To be successful, technical knowledge and knowledge about user functions and needs must be learned by the organization and embodied in the system design and implementation. Based upon the above discussion, we propose the following research model (see Figure 1). We propose that the lack of system development technical knowledge and the lack of application domain knowledge will negatively impact organizational technology learning, and, in turn, negatively impact final project outcomes.

Figure 1. Research model of risk impact on learning and performance



The detailed hypotheses examined in this study are discussed in the following section.

HYPOTHESES

For learning to occur, it is accepted that certain conditions must be in place in the organization. One seeming catch-22 is that existing knowledge must be in place for learning to occur (Cohen & Levinthal, 1990). In this instance, learning is not merely the sum of the basic knowledge and skills of the individuals, but the structure in place to retain and transfer knowledge throughout the organization (Schulz, 2001). This by itself indicates that individual knowledge is not sufficient to guarantee that organizational technology learning will take place. Knowledge and skills must be the collective of a project team that can be directed to accomplish the goals of a software development project (Schulz, 2001).

Absorptive capacity is a theory-based explanation of this phenomena (Cohen & Levinthal, 1990). The basic premise of absorptive capacity is that the IS project team must have prior related knowledge and skills to assimilate and utilize the new knowledge. Research in other fields supports this premise. Memory development research suggests that an accumulation of prior knowledge improves the ability to store new knowledge and the ability to recall and use knowledge as skills (Bower & Hilgard, 1981). Furthermore, lack of knowledge in the appropriate context limits the ability to make new knowledge intelligible (Lindsay & Norman, 1977). New knowledge must be exploitable as skills and depends upon the transfer of knowledge within the organization (Sarin & McDermott, 2003). Essentially, a lack of knowledge blocks organizational access to further knowledge and the ability to utilize the skills derived from the knowledge. A previous knowledge base is important to the rapid dissemination and

assimilation of the knowledge required to exploit technology and adopt new technology into productive domain applications (Ko et al., 2005). In addition, diversity of knowledge is important for further learning in order to place new knowledge in a corporate context (Fichman & Kemerer, 1997). Based upon absorptive capacity theory and existing studies demonstrating the importance of previous knowledge, we expect:

- **H1a:** There is a negative relationship between the lack of system development skills and organizational technology learning.
- **H1b:** There is a negative relationship between the lack of application domain knowledge and organizational technology learning.

Risk-based software engineering describes why software project uncertainties have an adverse impact on performance (Boehm, 1991). According to this theory, project uncertainties can be viewed as risk drivers that increase the performance risk of the project. The purpose of software engineering is to manage the particular sources of project risks and lead to a successful project development. If this holds, then project performance depends upon the reduction or control of the various risks in place, including those associated with knowledge deficiencies on the part of the IS development staff. Research based upon the analysis of risk in project environments supports this notion (Jiang & Klein, 2000). Researchers and practitioners have indicated that the lack of knowledge on the part of the IS staff is a significant project risk that could negatively impact a project's outcome (Schmidt, Lyytinen, Keil, & Cule, 2001). In the IS skill literature, Jiang et al. (2003) found that the lack of skills was correlated with project failures. Based upon the risk management theory and above discussion, we expect that:

- **H2a:** There is a negative relationship between the lack of system development skills of IS staff and project performance.
- **H2b:** There is negative relationship between the lack of application domain knowledge of IS staff and project performance.

Various learning models incorporate the concept of feedback (Argyris, 1999). Single-loop learning is defined as matching expected consequences to those incurred from action and correcting the actions when a match is not present. Double-loop learning occurs when the mismatch causes a reflection on the underlying rules and principles of the system. This mechanism is similar to the feedback and control mechanisms of cybernetic (control) theory (Henderson & Lee, 1992). In projects, when expectations are not met, this creates feedback that is then used in a learning process to prevent the errors in the future. Thus, when organizational technology learning occurs, subsequent errors in the software development projects are reduced through either corrective action or reformulation of the principles of practice.

Performance improvement by learning is a general assumption in learning research (Lee & Choi, 2003; Sarin & McDermott, 2003). A study on shared knowledge has found positive relationships to group performance (Nelson & Cooprider, 1996). Project teams pursuing project goals are usually defined to be groups along the lines of those whose performance can be improved by shared knowledge (Klein, 1991). Organizations benefit from knowledge management practices implanted during system development and completed system projects (Fedor, Ghosh, Caldwell, Maurer, & Singhal, 2003; Sher & Lee, 2004). This process of sharing knowledge is considered one form of learning (Schulz, 2001). Thus, based on limited empirical results and the adaptive learning model, we expect support for:

- **H3:** There is a positive relationship between organizational technology learning and project performance.

RESEARCH METHODS

Sample

Questionnaires were mailed to 1,000 randomly selected IEEE Computer Society members. The letter salutation was directed to software engineers. This sample is likely familiar with software development activities and, thus, appropriate for study. Instructions in the letter told the targets to consider their responses in light of their most recently completed software development project. From the initial mailing and a follow-up, 221 responses were received. Nine questionnaires were eliminated due to missing data, leaving a final sample of 212 used in the data analysis. Demographic features of the sample population are in Table 1.

Non-response bias occurs when the opinions and perceptions of the survey respondents do not accurately represent the overall sample to which the survey was sent. One test for non-response bias is to compare the demographics of early versus late respondents to the survey (Armstrong & Overton, 1977). T-tests on the means of key demographics (work experience, gender, recent project duration, and team sizes) to examine whether significant differences existed between early and late respondents in the first mailing, and from those received in the first mailing to those in the follow-up mailing, found no significant differences.

Since all independent and dependent variables were collected from the same subject at the same time, common method bias was assessed using Harman's one-factor test (Podsakoff & Organ, 1986). Two models, a single factor model and a four-factor model, were created and tested in EQS. The model fit indices of the single factor model (GFI=0.514, CFI=0.564, Chi-Square = 1343.264, df =152, RMR=0.231, and RMSEA=0.193) were worse than the model fit indices of the four factors model (GFI = 0.849, CFI = 0.928, Chi-Square = 343.6, df =146, RMR=0.068, RMSEA=0.08) indicating that common method bias will not be a potential problem in the following analysis.

Constructs

Project performance is a comprehensive view that includes meeting project goals, budget, schedule, user requirement, and operational efficiency considerations. This is also reflected in the information systems literature in terms of meeting system budget, meeting delivery schedule, fulfilling user requirements, the amount of work produced, the quality of work produced, and an ability to meet project goals (Jones & Harrison, 1996). The items used in this study were adopted from a previous study (Henderson & Lee, 1992). The questionnaire asked respondents' satisfaction with the project team performance when developing information systems. The specific items are listed in Table 2. Each item was scored using a five-point scale ranging from criterion not met at all (1) to criterion fully met (5). All items were presented such that the greater the score, the greater the satisfaction of the particular item.

Organizational technology learning describes the technology knowledge being acquired by the firm (Cooprider & Henderson, 1990). This instrument was applied in previous studies involving organizational technology learning (Cooprider & Henderson, 1990). Three items were used to measure this construct and are shown in Table 2. Respondents were asked to indicate the extent of the items typically incurred when developing IS applications in their organization. Each item was scored using a five-point scale ranging from never occurring (1) to always occurring (5). All items were presented such that the greater the score, the greater the extent of the particular item occurring during the system development.

Table 1. Demographic information

1. Gender	
Male	189
Female	23
2. Position:	
IS Manager	55
Project Leader	71
IS Professional	75
Others	8
3. The industry type of your company:	
Service	110
Manufacturing	78
Education	12
Others	6
4. Average IS project duration in your organization:	
1 years and under	85
1 – 2 years	75
2 – 3 years	23
3 – 5 years	11
6 or more years	13
5. Average size of IS project teams in your organization:	
7 and under	113
8 – 15	69
16 – 25	13
26 and over	13

Lack of application domain knowledge relates to the IS staff's knowledge of the new application areas. *Lack of system development skills* is the IS staff's overall lack of expertise of the development methods used during system development. These two risks are all based on items proposed in earlier studies (Barki, Rivard, & Talbot, 1993; Jiang, Klein, & Means, 2000). The items associated with each of these risks are in Table 2. All items were presented such that the greater the score, the greater the extent of the particular item present during the system development projects.

Analytical Procedures

The analysis followed a two-step procedure. In the first step, confirmatory factor analysis (CFA) was applied to develop a measurement model describing the nature of the relationship between a number of latent factors and the mani-

Table 2. Measurement model—confirmatory factor analysis

Construct Indicators	***Loadings***	***T-value***
Project Performance (α=.90)		
Ability to meet project goals	.79	12.64*
Expected amount of work completed	.76	11.98*
High quality of work completed	.82	13.43*
Adherence to schedule	.75	11.87*
Adherence to budget	.71	10.30*
Efficient task operations	.76	12.09*
Organizational Technology Learning (α=.80)		
Knowledge is acquired by your organization about use of key technologies	.86	13.31*
Knowledge is acquired by your organization about use of development techniques	.83	12.69*
Knowledge is acquired by your organization about supporting users' business	.57	7.95*
Lack of System Development Skills (α=.87)		
Lack of expertise in the development methodology used in projects	.84	13.86*
Lack of expertise in the development support tools used in projects (e.g., DFD, flowcharts, ER model, CASE tools)	.90	15.41*
Lack of expertise in the project management tools used in projects (e.g., PERT charts, Gantt diagrams, walk-throughs, project management software)	.79	12.68*
Lack of expertise in the implementation tools used in projects (e.g., programming languages, data base inquiry languages, screen generators)	.62	9.03*
Lack of Application Domain Knowledge (α=.86)		
The members of the development team are unfamiliar with the application types	.68	10.25*
Insufficient knowledge of organizational operations	.89	14.97*
Lack of knowledge of the functioning of user departments	.95	14.04*
Lack of expertise in the specific application areas of new systems	.71	10.71*

fest indictor variable that measures those latent variables. In step two, the measurement model serves to test the theoretical model of interest. The indicators used to present the latent variables in the theoretical model tested are identical to those presented in the measurement model. This analysis step may be described as a path analysis with latent variables. The path model consists of the unobservable constructs and the theoretical relationships among them (the paths). It evaluates the explanatory power of the model and the significance of paths in the structural model, which represent hypotheses to be tested. The estimated path coefficients indicate the strength and the sign of the theoretical relationships.

Three important assumptions associated with path analysis are: 1) normal distribution of the variables examined; 2) absence of multicollinearity among the variables; and 3) a limit on the maximum number of variables in the model. To test for normality, Mardia's multivariate kurtosis and normalized multivariate kurtosis tests were conducted. No violation was found. Multicollinearity is present when one or more variables exhibit very strong correlations with one another. The correlations between variables (see Table 3) were all less than .80, thus no significant violation of multicollinearity was present (Anderson & Gerbing, 1988).

When conducting a CFA, if the model provides a reasonably good approximation to reality, it should provide a good fit to the data. The CFA for the measurement model resulted in a root mean square residual of .07 (<= .10 is recommended), a chi-square/degree of freedom ratio of 2.41 (<= 3 is recommended), a comparative fit index of .90 (>= .90 recommended), and a non-normed fit index of .89 (>= .90 recommended) using the SAS CALIS procedure. The recommended values are based on research traditions and established authors in the field of structural equation modeling (Bentler, 1990). The measurement model was adequate for the data set.

Convergent validity is demonstrated when different instruments are used to measure the same construct, and scores from these different instruments are strongly correlated. Convergent validity can be assessed through t-tests on the factor loadings, such that the loadings are greater than twice their standard error (Anderson & Gerbing, 1988). The t-tests for the loadings of each variable are in Table 2. The results show that the constructs demonstrate a high convergent validity since all t-values are significant at the .05 levels. In addition, the reliability of each construct is examined by the Cronbach alpha value, all of which exceeded the recommend level of .70 (Nunnally, 1978).

A threat to external validity occurs if the sample shows systematic biases in terms of demographics. An ANOVA was conducted by using project performance as the dependent variable against each demographic category (as independent variables). Results did not indicate any significant relationship to project performance. Similar results were found for the remaining variables in the model. The external validity of the findings is also threatened if the sample is systematically biased – for example, if the responses were generally from projects that are more successful. Table 3 shows the descriptive statistics for the construct. The responses had a good distribution for project performance since the mean and median were similar, skewness was less than two, and kurtosis was less than five (Ghiselli, Campbell, & Zedeck, 1981). Similar results held for the remaining variables. Lastly, external validity is improved if the measures are similar to those found in other studies. In our case, the project performance measure matches within 2% to two other studies (Henderson & Lee, 1992; Jones & Harrison, 1996).

Discriminant validity is demonstrated when different instruments are used to measure different constructs and the correlations between the measures of those different constructs are relatively weak. Discriminant validity is assessed

Table 3. Descriptive analysis and correlations of measures

	Organizational Technology Learning	*Lack of Development Skills*	*Lack of Application Knowledge*	*Project Performance*
Mean	3.64	2.64	2.47	3.52
Std. Dev.	.83	.93	.8	.75
Median	3.67	2.75	2.50	3.67
Skewness	-.34	.09	.07	-.51
Kurtosis	-.26	-.56	-.50	.22
Correlations				
Lack of Development Skills	-.36 (.0001)			
Lack of Application Knowledge	-.37 (.0001)	.64 (.0001		
Project Performance	.54 (.0001)	-.46 (.0001)	-.36 (.0001)	

by using the confidence interval test (Anderson & Gerbing, 1988). A confidence interval test involves calculating a confidence interval of plus or minus two standard errors around the correlation between factors and determining whether this interval includes 1.0 (or -1.0). If the interval (for each pair of constructs) does not include 1.0, discriminant validity is demonstrated. No violations were found.

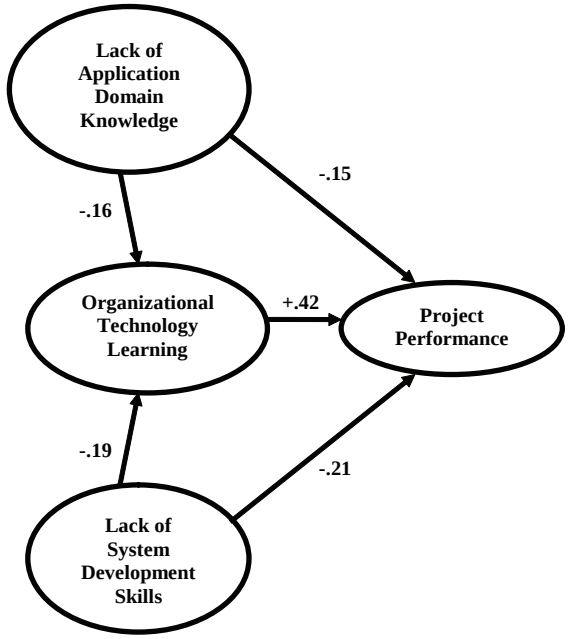
RESULTS

The theorized model in Figure 1 fit the data reasonably well, with a root mean square residual of .08, a chi-square/degree of freedom fit of 2.47, a comparative fit index of .88, and a non-normed fit index of .86. Hypotheses H1a, H1b, H2a, H2b, and H3, were all supported at the .05 significance level. The high R-square value (.49) indicated that the independent variables included in this model were all critical to the project performance. The strengths of these relationships, path coefficients, are on Figure 2.

DISCUSSION

The importance of system development skills and business domain knowledge for IS personnel has been advocated in the IS literature for decades. However, empirical examinations that investigated the relationship between these skills and system success provided conflicting results

Figure 2. Research model path coefficients



(Byrd & Turner, 2001). To provide an alternative explanation of the IS skill puzzle in the literature, this study proposed that organizational technology learning is a mediator between the lack of system development technical skills and application domain knowledge and project performance. Results indicate that risk of lack of skills and knowledge adversely impacts IS software development projects and interferes with the organizational technology learning required to help overcome the risks. Furthermore, organizational technology learning was found to promote project performance.

For researchers, this study complements existing literature in several respects. First, learning does not take place unless a set of knowledge and skills is in place to use as a foundation (Cohen & Levinthal, 1990). This result has specific consequences to the limited framework of the study but also puts forth the idea that risks thought to be associated with project performance may impact other organizational processes and remedies. Second, lack of knowledge and skills is shown to be a large risk to the performance of projects. This study confirms the IS skill literature that a lack of skills does have a *negative* impact on project performance. Third, the *positive* relationship between IS skills and project performance may be explained by the mediator -- organizational technology learning. This extends the mediation results found in the organizational setting to the project environment (Tippins & Sohi, 2003). It provides an alternative explanation for a lack of positive relationships between IS skills and project performance in previous studies. Some natural actions to promote learning of the technology may be needed for the newer developments (Schulz,

2001). Additionally, a direct link between organizational technology learning and the performance of IS projects is established. This goes beyond the literature that considers success limited to the completion of learning, the dissemination of new ideas, or the adoption of new methodologies.

For practitioners, the results encourage management to seek ways to overcome risks associated with knowledge deficiencies. The IS personnel literature contains studies of composing teams that have a variety of knowledge and skills (Jiang, Klein & Balloun, 1998). This goes along with a need for a diversity of knowledge and skills. The knowledge management literature supports this knowledge diversity through examples of loyalty in personnel and cross-fertilization of specialties (Nonaka & Takeuchi, 1995). A set of knowledge must be present in order to acquire more knowledge and must be present in the individuals in order for the organization to acquire and utilize the knowledge as applied skills (Cohen & Levinthal, 1990). Hiring practices must be set to ensure the requisite variety and background in the organization as a whole and in the individual expected to grow the knowledge and skills of the organization.

Organizational policies or practice may also contribute to the ability to overcome lack of knowledge and skills. Training is still an effective tool in the preparation of employees; however, the organization must appropriately target the training. Employees, or groups of employees, with limited background may not gain as much as employees with a stronger background. Employees with a broad technical, business, and liberal arts background have a larger learning set to grow from than do employees with strictly a technical background. Appropriate structures that promote growth of knowledge should serve to leverage existing knowledge (Nonaka & Takeuchi, 1995). Knowledge systems within an organization serve to distribute the knowledge more widely (Davenport & Prusak, 2000). Each of these techniques must be carefully weighed to determine their value in overcoming deficiencies.

The study is limited by its scope and sample. The scope is narrowly defined to include only certain components and only learning that occurs about technology application. However, though this study took a unique posture in its scope, it does confirm existing learning theory. A bigger limitation is on the risks considered. Other risk factors are well known to play a critical role in project performance (Barki et al., 1993). What must be remembered is that risks are not isolated and are often interrelated. The control of one risk factor may lead to an associated increase in another. This study does not account for all of the possible interactions inherent in risk interrelationships.

REFERENCES

Anderson, J. C., & Gerbing, G. W. (1988). Structural equation modeling in practice: A review and recommended two-step approach. *Psychological Bulletin, 103*(3), 411-423.

Argyris, C. (Ed.). (1999). *On organizational learning* (Second ed.). Malden: Blackwell Publishers, Inc.

Armstrong, J. S., & Overton, T. S. (1977). Estimating non-response bias in mail surveys. *Journal of Marketing Research, 14*, 396-402.

Badaracco, J. (1991). *The knowledge link: How firms compete through strategic alliances*. Boston: Harvard Business School Press.

Barki, H., Rivard, S., & Talbot, J. (1993). Toward an assessment of software development risk. *Journal of Management Information Systems, 10*(2), 202-223.

Bentler, P. M. (1990). Comparative fit indexes in structural models. *Psychological Bulletin, 107*(2), 238-246.

Boehm, B. W. (1991). Software risk management: Principles and practice. *IEEE Software*, 32-41.

Bower, G. H., & Hilgard, E. R. (1981). *Theories of learning.* Englewood Cliffs, NJ: Prentice-Hall.

Byrd, T. A., & Turner, D. E. (2001). An exploratory analysis of the value of the skills of IT personnel: Their relationship to IS infrastructure and competitive advantage. *Decision Sciences*, *32*(1), 21-54.

Cheney, P. H., & Lyons, N. R. (1980). Information systems skill requirements: A survey. *MIS Quarterly*, *4*(1), 35-43.

Chiva, R., & Alegre, J. (2005). O*rganizational learning* and *organizational* knowledge: Towards the integration of two approaches. *Management Learning, 36*(1), 49-68.

Cohen, W. M., & Levinthal, D. A. (1990). Absorptive capacity: A new perspective on learning and innovation. *Administrative Science Quarterly, 35*, 128-152.

Cooprider, J. G., & Henderson, J. C. (1990). Technology-process fit: Perspectives on achieving prototyping effectiveness. *Journal of Management Information Systems*, *7*(3), 67-87.

Davenport, T. H., & Prusak, L. (2000). *Working knowledge*. Boston: Harvard University Press.

Fedor, D. B., Ghosh, S., Caldwell, S. D., Maurer, T. J., & Singhal, V. R. (2003) The effects of knowledge management on team members' ratings of project success and impact. *Decision Sciences*, *34*(3), 513-539.

Fichman, R. G., & Kemerer, C. F. (1997). The assimilation of software process innovations: An organizational learning perspective. *Management Science*, *43*(10), 1345-1363.

Fiol, C. M., & Lyles, M. A. (1985). Organizational learning. *Academy of Management Review*, *10*(4), 803-813.

Ghiselli, E.E., Campbell, J.P., & Zedeck, J.P. (1981). *Measurement theory for the behavioral sciences*, San Francisco, CA: Freeman.

Grant, R. M. (1996). Prospering in dynamically-competitive environment: organizational capability as knowledge integration. *Organization Science*, *7*(4), 375-387

Green, G. I. (1989). Perceived importance of systems analysts' job skills, roles, and non-salary incentives. *MIS Quarterly*, *13*(2), 115-133.

Harlow, H. F. (1949). The formation of learning sets. *Psychological Review, 56*, 51-65.

Henderson, J. C., & Lee, S. (1992). Managing IS design teams: A control theory perspective. *Management Science*, *38*(6), 757-777.

Jiang, J. J., & Klein, G. (1999). Risks to different aspects of system success. *Information & Management*, *36*, 263-272.

Jiang, J. J., & Klein, G. (2000). Software development risks to project effectiveness. *Journal of Systems and Software*, *52*, 3-10.

Jiang, J. J., Klein, G., & Balloun, J. (1998). Systems analysts' attitudes toward information systems development. *Information Resources Management Journal*, *11*(4), 5-10.

Jiang, J. J., Klein, G., & Means, T. (2000). Project risk impact on software development team performance. *Project Management Journal*, *31*(4), 19-26.

Jiang, J. J., Klein, G., Van Slyke, C., & Cheney, G. (2003). A note on interpersonal and communication skills for IS professionals: Evidence of positive influence. *Decision Sciences*, *34*, 4, 1-15.

Jones, M. C., & Harrison, A. W. (1996). IS project team performance: An empirical assessment. *Information & Management*, *31*(2), 57-65.

Kayes, A. B., Kayes, D. C., & Kolb, D. A. (2005). Experiential learning in teams. *Simulation & Gaming, 36*(3), 330-354.

Klein, H. (1991). Further evidence on the relationship between goal setting and expectancy theories. *Organizational Behavior and Human Decision Processes, 49*, 230-257.

Ko, D., Kirsch, L., & King, W. (2005). Antecedents of knowledge transfer from consultants to clients in enterprise system implementations. *MIS Quarterly, 29*(1), 59-85.

Kogut, B., & Zander, U. (1992). Knowledge of the firm, competitive capabilities, and the replication of technology. *Organization Science 3*, 383-397.

Lee, H., & Choi, B. (2003). Knowledge management enablers, processes, and organizational performance: An integrative view and empirical examination. *Journal of Management Information Systems, 20*(1), 179-228.

Lindsay, P. H., & Norman, D. A. (1977). *Human information processing.* Orlando, FL: Academic Press.

Majchrzak, A., Beath, C. M., Lim, R. A., & Chin, W. W. (2005). Managing client dialogues during information system design to facilitate client learning. *MIS Quarterly, 29*(4), 653-672.

Nelson, K. M., & Cooprider, J. G. (1996). The contribution of shared knowledge to IS group performance. *MIS Quarterly, 20*(4), 409-432.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company: how Japanese companies create the dynamics of innovation.* Oxford: Oxford University Press.

Nunnally, J. C. (1978). *Psychometric theory (2nd ed.).* New York: McGraw-Hill.

Okhuysen, G., & Eisenhardt, K. (2002). Integrating knowledge in groups: How formal interventions enable flexibility. *Organization Science 13*(4), 370-386.

Olfman, L., Bostrom, R. P., & Sein, M. K. (2003). A best-practice model for information technology learning strategy formulation. In *Proceedings of the 2003 SIGMIS Conference on Computer Personnel Research*, ACM Digital Library, 75-86.

Podsakoff, P. M., & Organ, D. W. (1986). Self-reports in organizational research: Problems and prospects. *Journal of Management, 12*, 531-544.

Robillard, R. (1999). The role of knowledge in software development. *Communications of the ACM, 42*(1), 87-92.

Rus, L., & Lindvall, M. (2002). Knowledge management in software engineering. *IEEE Software*, 26-38.

Sarin, S., & McDermott, C. (2003) The effect of team leader characteristics on learning, knowledge application, and performance of cross-functional new product development teams. *Decision Sciences, 34*(4), 707-739.

Schmidt, R. C., Lyytinen, K., Keil, M., & Cule, P. E. (2001). Identifying software project risks: An international Delphi study. *Journal of Management Information Systems, 17*(4), 5-36.

Schramer, C. (2000). Organizing around not-yet-embedded knowledge. In G. Von Krogh, I. Nonaka, & T. Nishiguchi (Eds.), *Knowledge creation: A source of wealth* (pp. 36-60). London: Palgrave MacMillan.

Schulz, M. (2001). The uncertain relevance of newness: Organizational learning and knowledge flows. *Academy of Management Journal, 44*(4), 661-681.

Sher, P. J., & Lee, V. C. (2004). Information technology as a facilitator for enhancing dynamic capabilities through knowledge management. *Information & Management, 41*(8), 933-945.

Stein, E. W., & Vandenbosch, B. (1996). Organizational learning during advanced system

development: Opportunities and obstacles. *Journal of Management Information Systems*, *13*(2), 115-136.

Tippins, J., & Sohi, R. (2003). IT competency and firm performance: Is organizational learning a missing link. *Strategic Management Journal*, *24*, 745-761.

This work was previously published in the Information Resources Management Journal, edited by M. Khosrow-Pour, Volume 20, Issue 3, pp. 32-45, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.27
Patterns for Organizational Modeling

Manuel Kolp
Université Catholique de Louvain, Belgium

Stéphane Faulkner
University of Namur, Belgium

ABSTRACT

Organizational modeling is concerned with analyzing and understanding the organizational context within which a software system will eventually function. This paper proposes organizational patterns motivated by organizational theories intended to facilitate the construction of organizational models. These patterns are defined from real world organizational settings, modeled in i* and formalized using the Formal Tropos language. Additionally, the paper evaluates the proposed patterns using desirable qualities such as coordinability and predictability. The research is conducted in the context of Tropos, a comprehensive software system development methodology.

INTRODUCTION

Analyzing the organizational and intentional context within which a software system will eventually operate has been recognized as an important element of the organizational modeling process also called early requirements engineering (see e.g., Anton, 1996; Dardenne, van Lamsweerde, & Fickas, 1993; Yu, 1995). Such models are founded on primitive concepts such as those of actor and goal. This paper focuses on the definition of a set of organizational patterns that can be used as building blocks for constructing such models. Our proposal is based on concepts adopted from organization theory and strategic alliances literature. Throughout the paper, we use i* (Yu, 1995) as the modeling framework in terms of which the proposed patterns are presented and accounted for. The research reported in this paper is being conducted within the context of the Tropos project

(Giorgini, Kolp, Mylopoulos, & Pistore, 2004; Giorgini, Kolp, Mylopoulos, & Castro, 2005), whose aim is to construct and validate a software development methodology for agent-based software systems. The methodology adopts ideas from multi-agent system technologies, mostly to define the implementation phase of our methodology. It also adopts ideas from Requirements Engineering, where actors and goals have been used heavily for early requirements analysis. The project is founded on the premise that actors and goals are used as fundamental concepts for modeling and analysis during all phases of software development, not just early requirements, or implementation. More details about Tropos can be found in Giorgini, et al., 2005. The present work continues the research in progress about social abstractions for the Tropos methodology. In Kolp, Giorgini and Mylopoulos (2002a), we have detailed a social ontology for Tropos to consider information systems as social structures all along the development life cycle. In Giorgini, Kolp, and Mylopoulos (2002); Kolp, Giorgini, and Mylopoulos (2002b); and Kolp, Giorgini, and Mylopoulos (2006), we have described how to use this Tropos social ontology to design multi-agent systems architectures, notably for e-business applications (Kolp, Do, & Faulkner, 2004). As a matter of fact, multi-agent systems can be considered structured societies of coordinated autonomous agents. In the present paper, which is a extended and revised version of Kolp, Giorgini, and Mylopoulos (2003), we emphasize the use of organizational patterns based on organization theory and strategic alliances for early requirements analysis, with the concern of modeling the organizational setting for a system-to-be in terms of abstractions that could better match its operational environment (e.g., an enterprise, a corporate alliance, etc.).

The paper is organized as follows: The second section describes organizational and strategic alliance theories, focusing on the internal and external structure of an organization. The third section details two organizational patterns—the structure-in-5 and the joint venture—based on real world examples of organizations. These patterns are modeled in terms of social and intentional concepts using the i* framework and the Formal Tropos specification language. The fourth section identifies a set of desirable non-functional requirements for evaluating these patterns and presents a framework to select a pattern with respect to these identified requirements. The fifth section overviews the *Tropos* methodology. Finally, The sixth section summarizes the contributions of the paper and provides an overview of related work.

STRUCTURING ORGANIZATIONS

Organizational structures are primarily studied by *organization theory* (e.g., Mintzberg, 1992; Scott, 1998; Yoshino & Rangan, 1995), that describes the structure and design of an organization and *strategic alliances* (e.g., Dussauge & Garrette, 1999; Gomes-Casseres, 1996; Morabito, Sack, & Bhate, 1999; Segil, 1996), that model the strategic collaborations of independent organizational stakeholders who have agreed to pursue a set of agreed upon business goals.

Both disciplines aim to identify and study organizational patterns that describe a system at a macroscopic level in terms of a manageable number of subsystems, components, and modules interrelated through dependencies.

In this article, we are interested in identifying, formalizing and applying organizational modeling patterns that have been already well-understood and precisely defined in organizational theories. Our purpose is neither to categorize them exhaustively nor to study them on a managerial point of view. The following sections will thus only insist on patterns that have been found, due to their nature, interesting candidates, also considering the fact that they have been studied in great detail in the organizational literature and presented as fully-formed patterns.

Organization Theory

"An organization is a consciously coordinated social entity, with a relatively identifiable boundary, that functions on a relatively continuous basis to achieve a common goal or a set of goals" (Morabito et al., 1999). Organization theory is the discipline that studies both structure and design in such social entities. Structure deals with the descriptive aspects, while design refers to the prescriptive aspects of a social entity. Organization theory describes how practical organizations are actually structured, offers suggestions on how new ones can be constructed, and suggests how old ones can change to improve effectiveness. To this end, since Adam Smith, schools of organization theory have proposed models and patterns to try to find and formalize recurring organizational structures and behaviors.

In the following, we briefly present organizational patterns identified in Organization Theory. The structure-in-5 will be studied in detail in the next section.

The Structure-in-5

An organization can be considered an aggregate of five substructures, as proposed by Minztberg (1992). At the base level sits the *Operational Core,* which carries out the basic tasks and procedures directly linked to the production of products and services (acquisition of inputs, transformation of inputs into outputs, and distribution of outputs). At the top lies the *Strategic Apex* which makes executive decisions ensuring that the organization fulfils its mission in an effective way and defines the overall strategy of the organization in its environment. The *Middle Line* establishes a hierarchy of authority between the Strategic Apex and the Operational Core. It consists of managers responsible for supervising and coordinating the activities of the Operational Core. The *Technostructure* and the *Support* are separated from the main line of authority and influence the operating core only indirectly. The Technostructure serves the organization by making the work of others more effective, typically by standardizing work processes, outputs, and skills. It is also in charge of applying analytical procedures to adapt the organization to its operational environment. The Support provides specialized services at various levels of the hierarchy, outside the basic operating workflow (e.g., legal counsel, R&D, payroll, and cafeteria). We describe and model examples of structures-in-5 in the next section.

The Pyramid Pattern

A well-known hierarchical authority structure. Actors at lower levels depend on those at higher levels. The crucial mechanism is the direct supervision from the Apex. Managers and supervisors at intermediate levels only route strategic decisions and authority from the Apex to the operating (low) level. They can coordinate behaviors or take decisions on their own, but only at a local level.

The Chain of Values

The chain of values merges, backward or forward, several actors engaged in achieving or realizing related goals or tasks at different stages of a supply or production process. Participants who act as intermediaries, add value at each step of the chain. For instance, for the domain of goods distribution, providers are expected to supply quality products, wholesalers are responsible for ensuring their massive exposure, while retailers take care of the direct delivery to the consumers.

The Matrix

The matix proposes a multiple command structure: vertical and horizontal channels of information and authority operate simultaneously. The principle of unity of command is set aside, and competing bases of authority are allowed to jointly govern the workflow. The vertical lines are typi-

cally those of functional departments that operate as "home bases" for all participants, the horizontal lines represent project groups or geographical arenas where managers combine and coordinate the services of the functional specialists around particular projects or areas.

The Bidding Pattern

The bidding pattern involves competitive mechanisms, and actors behave as if they were taking part in an auction. An auctioneer actor runs the show, advertises the auction issued by the auction issuer, receives bids from bidder actors and ensures communication and feedback with the auction issuer who is responsible for issuing the bidding.

Strategic Alliances

A strategic alliance links specific facets of two or more organizations. At its core, this structure is a trading partnership that enhances the effectiveness of the competitive strategies of the participant organizations by providing for the mutually beneficial trade of technologies, skills, or products based upon them. An alliance can take a variety of forms, ranging from arm's-length contracts to joint ventures, multinational corporations to university spin-offs, and franchises to equity arrangements. Varied interpretations of the term exist, but a strategic alliance can be defined as possessing simultaneously the following three necessary and sufficient characteristics:

- The two or more organizations that unite to pursue a set of agreed upon goals remain independent, subsequent to the formation of the alliance.
- The partner organizations share the benefits of the alliances and control over the performance of assigned tasks.
- The partner organizations contribute on a continuing basis in one or more key strategic areas, e.g., technology, products, and so forth.

In the following, we briefly present organizational patterns identified in strategic alliances. The joint venture will be studied in detail in the third section.

Joint Venture Pattern

The joint venture pattern involves agreement between two or more intra-industry partners to obtain the benefits of larger scale, partial investment, and lower maintenance costs. A specific joint management actor coordinates tasks and manages the sharing of resources between partner actors. Each partner can manage and control itself on a local dimension and interact directly with other partners to exchange resources, such as data and knowledge. However, the strategic operation and coordination of such an organization, and its actors on a global dimension, are only ensured by the joint management actor in which the original actors possess equity participations. We describe and model examples of joint ventures in the third section.

Arm's-Length Pattern

The arm's-length pattern implies agreements between independent and competitive, but partner, actors. Partners keep their autonomy and independence but act and put their resources and knowledge together to accomplish precise common goals. No authority is lost or delegated from one collaborator to another.

Hierarchical Contracting Pattern

The hierarchical contracting pattern identifies coordinating mechanisms that combine arm's-length

agreement features with aspects of pyramidal authority. Coordination mechanisms developed for arm's-length (independent) characteristics involve a variety of negotiators, mediators, and observers at different levels handling conditional clauses to monitor and manage possible contingencies, negotiate and resolve conflicts, and finally deliberate and take decisions. Hierarchical relationships, from the executive apex to the arm's-length contractors, restrict autonomy and underlie a cooperative venture between the parties.

Co-Optation Pattern

The co-optation pattern involves the incorporation of representatives of external systems into the decision-making or advisory structure and behavior of an initiating organization. By co-opting representatives of external systems, organizations are, in effect, trading confidentiality and authority for resource, knowledge assets, and support. The initiating system has to come to terms with the contractors for what is being done on its behalf; each co-optated actor has to reconcile and adjust its own views with the policy of the system it has to communicate.

MODELING ORGANIZATIONAL PATTERNS

We will define an organizational pattern as a metaclass of organizational structures offering a set of design parameters to coordinate the assignment of organizational objectives and processes, thereby affecting how the organization itself functions. Design parameters include, among others, goal and task assignments, standardization, supervision and control dependencies, and strategy definitions.

This section describes two of the organizational patterns presented in the second section: the structure-in-5 and the joint-venture.

Structure-in-5

To detail and specify the structure-in-5 as an organizational pattern, this section presents two case studies: LDV Bates (Bates, 2006) and GMT (GMT, 2006). They will serve to propose a model and a semi-formal specification of the structure-in-5.

LDV Bates

Agate Ltd. is an advertising agency located in Belgium that employs about fifty staff, as detailed in Table 1.

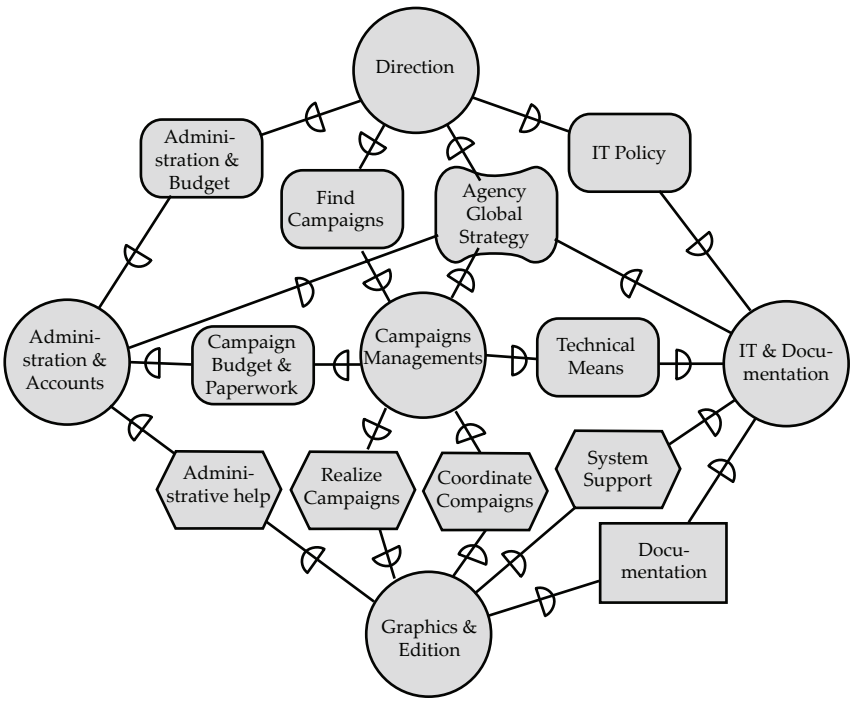
The *Direction*—four directors responsible for the main aspects of LDV Bates's *Global Strategy* (advertising campaigns, creative activities, administration, and finances)—forms the *Strategic Apex*. The *Middle Line*, composed of the *Campaigns Management* staff, is in charge of finding and coordinating advertising campaigns (marketing, sales, edition, graphics, budget, etc.). It is supported in these tasks by the *Administration and Accounts* and *IT and Documentation* departments. *Administration and Accounts* constitutes the *Technostructure* handling administrative tasks and policy, paperwork, purchases, and budgets. The Support groups the IT and Documentation departments. It defines the *IT policy* of Agate, provides *technical means* required for the management of campaigns, and ensures services for *system support* as well as information retrieval (*documentation* resources). The *Operational Core* includes the *Graphics and Edition* staff in charge of the creative and artistic aspects of realizing *campaigns* (texts, photographs, drawings, layout, design, and logos).

Figure 1 models LDV Bates in structure-in-5 using the i* strategic dependency model. i* is a framework for organizational modeling (Yu, 1995), which offers goal-and-actor-based notions, such as *actor, agent, role, position, goal, softgoal, task, resource and belief, as well as* different kinds

Table 1. Organization of LDV Bates

Direction	***Edition***	
1 Campaigns Director	2 Editors	***IT***
1 Creative Director	4 Copy writers	1 IT manager
1 Administrative Director		1 Network administrator
1 Finance Director	***Documentation***	1 System administrator
	1 Media librarian	1 Analyst
Campaigns Management	1 Resource librarian	1 Computer technician
2 Campaign managers	1 Knowledge worker	
3 Campaign marketers		
1 Editor in Chief	***Administration***	***Accounts***
1 Creative Manager	3 Direction assistants	1 Accountant manager
	4 Manager Secretaries	1 Credit controller
Graphics	2 Receptionists	2 Accounts clerks
6 Graphic designers	2 Clerks/typists	2 Purchasing assistants
2 Photographers	1 Filing clerk	

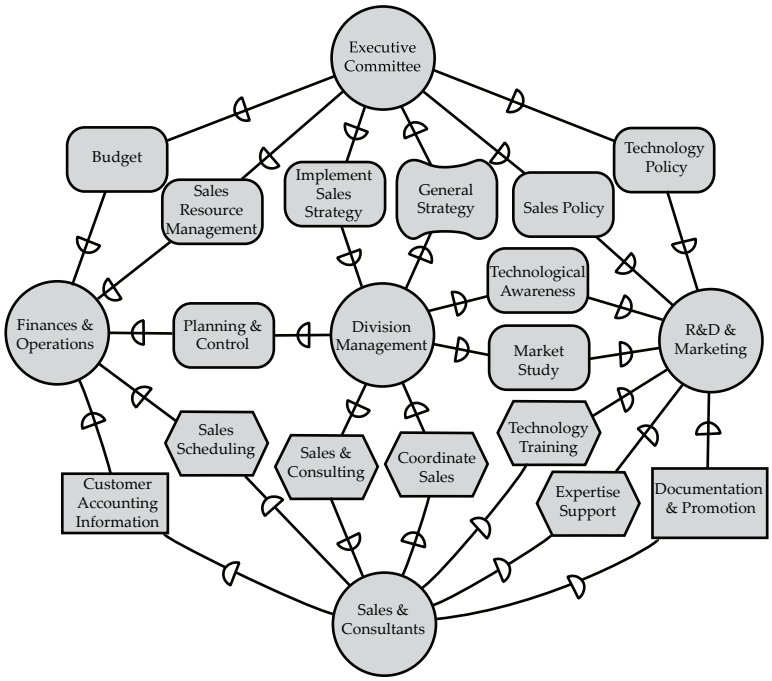
Figure 1. LDV Bates as a Structure-in-5



of social dependency between actors. Its strategic dependency model describes the network of social *dependencies* among actors. It is a graph, wherein each node represents an actor, and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The depender is the depending actor, and the dependee, the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely (for instance, the appreciation is subjective or fulfillment is obtained only to a given extent); *task* dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender. As shown in Figure 1, actors are represented as circles; dependums–goals, softgoals, tasks, and resources–are represented as ovals, clouds, hexagons, and rectangles, respectively, and dependencies have the form *depender* → *dependum* → *dependee*.

GMT is a company specialized in telecom services in Belgium. Its lines of products and services range from phones & fax, conferencing, line solutions, internet & e-business, mobile solutions, and voice & data management. As shown in Figure 2, the structure of the commercial organization follows the structure-in-5. An *Executive Committee* constitutes the *Strategic Apex*. It is responsible for defining the *general strategy* of the organization. Five chief managers (*finances*, *operations*, *divisions management*, *marketing*,

Figure 2. GMT's sales organization as a structure-in-5



and *R&D*) apply the specific aspects of the *general strategy* in the area of their competence: *Finances & Operations* is in charge of *Budget* and Sales *Planning* & *Control, Divisions Management* is responsible for *Implementing Sales Strategy*, and *Marketing* and *R&D* define *Sales Policy* and *Technological Policy.*

The *Divisions Management* groups managers that coordinate all managerial aspects of product and service sales. It relies on *Finance & Operations* for handling *Planning* and *Control* of products and services, it depends on *Marketing* for accurate *Market Studies,* and on R&D for *Technological Awareness.*

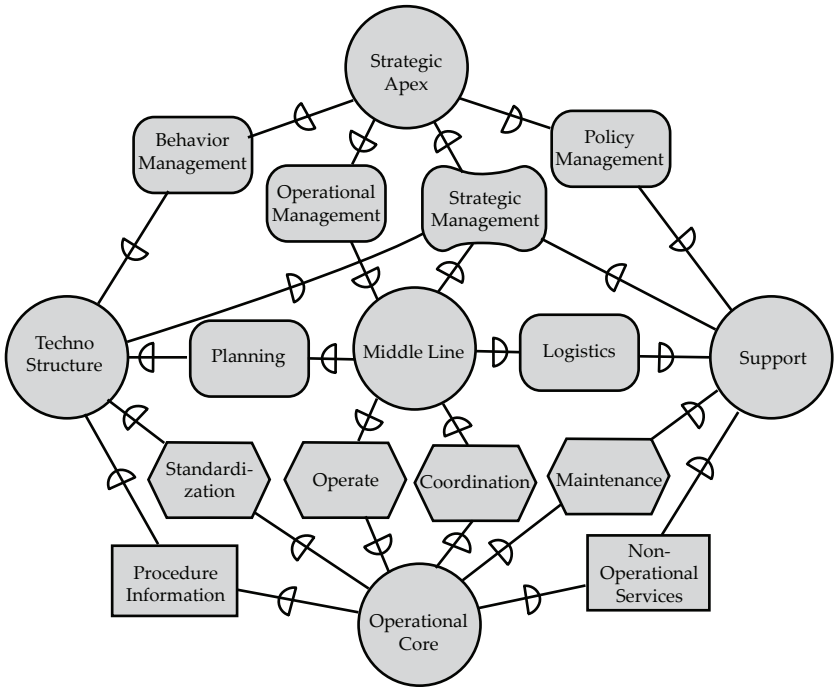
The *Finances & Operations* departments constitute the *technostructure* in charge of management *control* (financial and quality audit) and sales *planning,* including *scheduling* and *resource management.*

Support involves the staff of *Marketing* and *R&D.* Both departments jointly define and support the *Sales* Policy. The Marketing department coordinates *Market Studies* (customer positioning and segmentation, pricing, sales incentive, etc.) and provides the *Operational Core* with *Documentation* and *Promotion* services. The *R&D* staff is responsible for defining the technological policy, such as *technological awareness* services. It also assists *Sales people* and *Consultants* with *Expertise Support* and *Technology* Training.

Finally, the *Operational Core* groups *the Sales people* and *Line consultants* under the supervision and coordination of *Divisions Managers.* They are in charge of selling products and services to actual and potential customers.

Figure 3 abstracts the structures explored in the case studies of Figures 1 and 2 as a structure-in-5 pattern composed of five actors. The case

Figure 3. The structure-in-5 pattern



studies also suggested a number of constraints to supplement the basic pattern:

- Dependencies between the *Strategic Apex* as depender and the *Technostructure*, *Middle Line*, and *Support* as dependees must be of type goal
- A softgoal dependency models the strategic dependence of the *Technostructure*, *Middle Line*, and *Support* on the *Strategic Apex*
- Relationships between the *Middle Line* and *Technostructure* and *Support* must be of goal dependencies
- *Operational Core* relies on the *Technostructure* and *Support* through task and resource dependencies
- Only task dependencies are permitted between the *Middle Line* (as depender or dependee) and the *Operational Core* (as dependee or depender)

To specify the formal properties of the pattern, we use *Formal Tropos* (Fuxman, Liu, Mylopoulos, Roveri, & Traverso, 2004), which extends the primitives of *i** with a formal language comparable to that of KAOS (Dardenne, et al., 1993). Constraints on *i** specifications are thus formalized in a first-order linear-time temporal logic. *Formal Tropos* provides three basic types of metaclasses: *actor, dependency, and entity* (Giorgini, Kolp, & Mylopoulos, 2002). The attributes of a *Formal Tropos* class denote relationships among different objects being modeled.

Metaclasses

Actor : = **Actor** name[attributes] [creation-properties] [invar-properties][actor-goal]
With subclasses:
Agent(with attributes occupies: Position, play: Role)
Position(with attributes cover: Role)
Role
Dependency : =**Dependency** name type mode **Depender** name **Dependee** name [attributes] [creation-properties] [invar-properties] [fulfill-properties]
Entity:=**Entity** name [attribute] [creation-properties][invar-properties]
Actor-Goal := (**Goal**|**Softgoal**) name mode **FulFillment**(actor-fulfill-property)

Classes: Classes are instances of Metaclasses.

In Formal Tropos, constraints on the lifetime of the (meta)class instances are given in a first-order linear-time temporal logic (see Fuxman et al., 2004 for more details). Special predicates can appear in the temporal logic formulas: predicate *JustCreated*(*x*) holds in a state if element *x* exists in this state but not in the previous one; predicate *Fulfilled*(*x*) holds if *x* has been fulfilled; and predicate *JustFulfilled*(*x*) holds if *Fulfilled*(*x*) holds in this state, but not in the previous one.

In the following, we only present some specifications for the *Strategic Management* and *Operational Management* dependencies.

Actor StrategicApex
Actor MiddleLine
Actor Support
Actor Technostructure
Actor OperationalCore

Dependency StrategicManagement
Type SoftGoal
Depender te: Technostructure, ml: MiddleLine, su: Support
Dependee sa: StrategicApex
Invariant
$\forall dep : Dependency\ (JustCreated(dep) \rightarrow Consistent(self, dep))$
$\forall ag : Actor - Goal\ (JustCreated(ag) \rightarrow Consistent(self, ag))$

Fulfillment

$\forall dep : Dependency\ (dep.type = goal \wedge dep.depender = sa \wedge (dep.dependee = te\ dep.dependee = ml\ dep.dependee = su)) \wedge Fulfilled(self) \rightarrow \blacklozenge Fulfilled(dep)$

[*Invariant properties specify, respectively, that the strategic management softgoal must be consistent with any other dependency of the organization and with any other goal of the actors in the organization. The predicate Consistent depends on the particular organization we are considering and it is specified in terms of goals' properties to be satisfied. The fulfillment of the dependency necessarily implies that the goal dependencies between the Middle Line, the Technostructure, and the Support as dependees, and the Strategic Apex as depender have been achieved some time in the past]*

Dependency OperationalManagement **Type** Goal
Mode achieve
Depender sa: StrategicApex
Dependee ml: MiddleLine
Invariant

$Consistent(self, StrategicM\ anagement)$
$\exists\ c : Coordination\ (c.type = task \wedge c.dependee = ml \wedge c.depender = OperationalCore \wedge ImplementedBy(self, c))$

Fulfillment

$\forall ts : Technostructure, dep : Dependency\ (dep.type = goal \wedge dep.depender = ml \wedge dep.dependee = ts) \wedge Fulfilled(self)) \rightarrow \blacklozenge Fulfilled(dep)$

[The fulfillment of the Operational management goal implies that all goal dependencies between the Middle Line as depender and the Technostructure as dependee have been achieved some time in the past. Invariant properties specifies that the Operational Management goal has to be consistent with Strategic Management softgoal and that there exists a coordination task (a task dependency between MiddleLine and Operational Core) that implements (ImplementedBy) the OperationalManagaemnt goal.]

In addition, the following structural (global) properties must be satisfied for the Structure-in-5 pattern:

- $\forall inst1, inst2 : StrategicApex \rightarrow inst1 = inst2$

[There is a single instance of the Strategic Apex (the same constraint also holds for the Middle Line, the Technostructure, the Support, and the Operational Core)]

- $\forall sa : StrategicApex, te : Technostructure, ml : MiddleLine, su : Support, dep : Dependency\ (dep.dependee = sa \wedge (dep.depender = te \vee dep.depender = ml \vee dep.depender = su) \rightarrow dep.type = softgoal)$

[Only softgoal dependencies are permitted between the Strategic Apex as dependee and the Technostructure, the Middle Line, and the Support as dependers]

- $\forall sa : StrategicApex, te : T echnostructure, ml : M iddleLine, su : Support, dep : Dependency : (dep.depender = sa \wedge (dep.dependee = te \vee dep.dependee = ml \vee dep.dependee = su) \rightarrow dep.type = goal)$

[Only goal dependencies are permitted between the Technostructure, the Middle Line, and the Support as dependee, and the Stategic Apex as depender]

- $\forall su : Support, ml : M iddleLine, dep : Dependency$
  $((dep.dependee = su \wedge dep.depender = ml) \rightarrow dep.type = goal)$

  [Only task dependencies are permitted between the Middle Agency and the Operational Core]

- $\forall te : T echnostructure, oc : OperationalCore, dep : Dependency$
  $((dep.dependee = te \wedge dep.depender = oc) \rightarrow$
  $(dep.type = task \vee dep.type = resource))$

[Only resource or task dependencies are permitted between the Technostructure and the Operational Core (the same constraint also holds for the Support)]

- $\forall a : Actor, ml : M iddleLine,$
  $(\exists dep : Dependency(dep.depender = a \wedge dep.dependee =$
  $ml) \vee (dep.dependee = a \wedge dep.depender = ml) \rightarrow$
  $((\exists sa : StrategicApex(a = sa)) \vee (\exists su : Support(a = su) \vee$
  $(\exists te : T echnostructure(a = te)) \vee (\exists op : OperationalCore$
  $(a = op))$

[No dependency is permitted between an external actor and the Middle Agency (the same constraint also holds for the Operational Core)]

This specification can be used to establish that a certain *i** model does constitute an instance of the structure-in-5 pattern. For example, the *i** model of Figure 1 can be shown to be such an instance, in which the actors are instances of the structure-in-5 actor classes (e.g., *Direction* and *IT&Documentation* are instances of the *Strategic Apex* and the *Support*, respectively), dependencies are instances of structure-in-5 dependencies classes (e.g., *Agency Global Strategy* is an instance of the *Strategic Management*), and all above global properties are enforced (e.g., since there are only two task dependencies between *Campaigns Management* and *Graphics&Edition*, the fourth property holds).

Joint Venture

We describe here two alliances—Airbus (Dussauge & Garrette, 1999) and a more detailed one, Carsid (Wautelet, Kolp, & Achbany, 2006)—that will serve to model the joint venture structure as an organizational pattern and propose a semi-formal specification.

Airbus. The Airbus Industrie joint venture coordinates collaborative activities between European aeronautic manufacturers to build and market airbus aircraft. The joint venture involves four partners: British Aerospace (UK), Aerospatiale (France), DASA (Daimler-Benz Aerospace, Germany), and CASA (Construcciones Aeronauticas SA, Spain). Research, development, and production tasks have been distributed among the partners, avoiding any duplication. Aerospatiale is mainly responsible for developing and manufacturing the cockpit of the aircraft and for system integration. DASA develops and manufactures the fuselage, British Aerospace the wings, and CASA the tail unit. Final assembly is carried out in Toulouse (France) by Aerospatiale. Unlike production, commercial and decisional activities have not been split among partners. All strategy, marketing, sales, and after-sales operations are entrusted to the Airbus Industrie joint venture, which is the only interface with external stakeholders, such as customers. To buy an Airbus, or to maintain their fleet, customer airlines could not approach one or another of the partner firms directly, but have to deal with Airbus Industrie. Airbus Industrie, which is a real manufacturing company, defines the alliance's product policy and

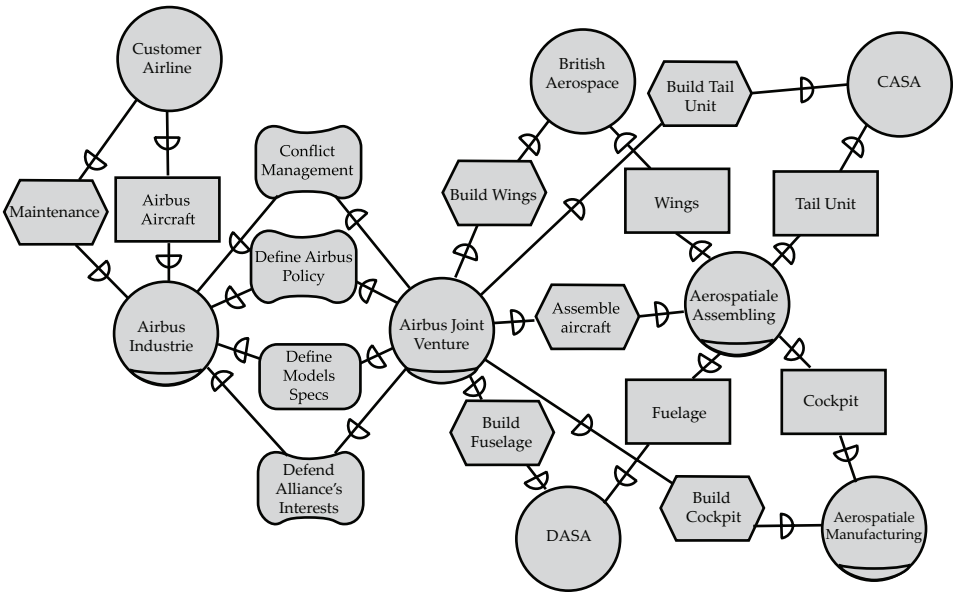
elaborates the specifications of each new model of aircraft to be launched. Airbus defends the point of view and interests of the alliance as a whole, even against the partner companies themselves when the individual goals of the latter enter into conflict with the collective goals of the alliance.

Figure 4 models the organization of the Airbus Industrie joint venture using the *i** strategic dependency model. Airbus assumes two roles: Airbus Industrie and Airbus Joint Venture.

Airbus Industrie deals with demands from customers, *Customer* depends on it to receive airbus aircrafts or maintenance services. The *Airbus Joint Venture* role ensures the interface for the four partners (*CASA*, *Aerospatiale*, *British Aerospace*, and *DASA*) with *Airbus Industrie* defining Airbus strategic policy, managing conflicts between the four Airbus partners, defending the interests of the whole alliance, and defining new aircraft specifications. *Airbus Joint Venture* coordinates the four partners, ensuring that each of them assumes a specific task in the building of Airbus aircrafts: wings building for *British Aerospace*, tail unit building for *CASA*, cockpit building and aircraft assembling for *Aerospace*, and fuselage building for *DASA*. Since Aerospatiale assumes two different tasks, it is modeled as two roles: *Aerospatiale Manufacturing* and *Aerospatiale Assembling*. *Aerospatiale Assembling* depends on each of the four partners receiving the different parts of the planes.

Carsid (Carolo-Sidérurgie) is a joint venture that has recently arisen from the global concentration movement in the steel industry. The alliance, physically located in the steel basin of Charleroi in Belgium, has been formed by the steel companies Duferco (Italy), Usinor (France)—that also partially owns Cockerill-Sambre (Belgium) through the Arcelor group—and Sogepa (Belgium), a public investment company representing the Walloon Region Government. Usinor has also brought its subsidiary Carlam into the alliance.

Figure 4. The Airbus Industrie joint venture

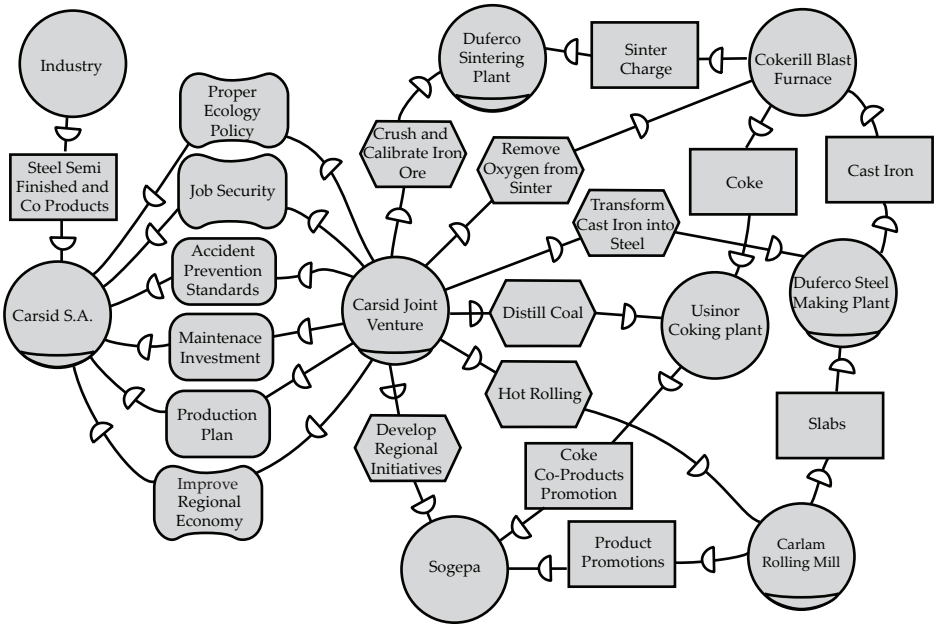


Roughly speaking, the aim of a steel manufacturing company like CARSID is to extract iron from the ore and to turn it into semi-finished steel products. Several steps compose the transformation process, each step is generally assumed by a specific metallurgic plant:

- **Sintering plant:** Sintering is the preparation of the iron ore for the blast furnace. The minerals are crushed and calibrated to form a sinter charge.
- **Coking plant:** Coal is distilled (i.e., heated in an air-impoverished environment in order to prevent combustion) to produce coke.
- **Blast furnace:** Coke is used as a combustion agent and as a reducing agent to remove the oxygen from the sinter charge. The coke and sinter charge are loaded together into the blast furnace to produce cast iron.
- **Steel making plant:** Different steps (desulphuration, oxidation, steel adjustment, cooling, etc.) are necessary to turn cast iron into steel slabs and billets. First, elements other than iron are removed to create molten steel. Then, supplementary elements (e.g., titanium, niobium, and vanadium) are added to make a more robust alloy. Finally, the result—finished steel—is solidified to produce slabs and billets.
- **Rolling mill:** The manufacture of semi-finished products involves a process known as hot rolling. Hot-rolled products are of two categories: flat (plates, coiled sheets, sheeting, strips, etc.) produced from steel slabs and long (wire, bars, rails, beams, girders, etc.) produced from steel billets.

Figure 5 models the organization of the Carsid joint venture in *i**. Carsid assumes two roles: Carsid S.A. ("Société Anonyme"–the English equivalent is "Ltd") and Carsid Joint Venture.

Figure 5. The Carsid joint venture

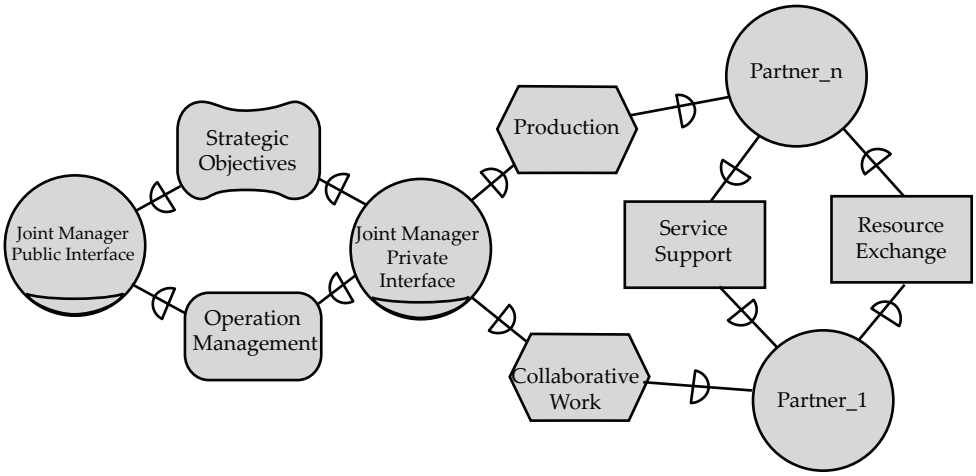


Carsid S. A. is the legal and contractual interface of the joint venture. It handles the sales of *steel semi-finished products* (bars, plates, rails, sheets, etc., but also slabs and billets) and *co-products* (coke that does not meet blast furnace requirements, rich gases from the different plants, godroon, naphtalin, etc.) to external *industries* such as vehicle manufacturers (e.g., automobiles, trains, and boats), foundries, gas companies, and building companies. It is also in charge of the *proper environment policy*, a strategic aspect for steelworks that are polluting plants. Most important, Carsid has been set up with the help of the Walloon Region to guarantee *job security* for about 2000 workers in the basin of Charleroi. Indeed, the steel industry in general and the Walloon metallurgical basins in particular are sectors in difficulty with high unemployment rates. As a corollary, the joint venture is committed to *improve regional economy* and maintain work in the region. Carsid has then been contractually obliged to plan *maintenance investment* (e.g., blast furnace refection, renovation of coke oven batteries, etc.) and develop *production plans* involving regional subcontractors and suppliers. Since steelmaking is a hard and dangerous work sector, Carsid, like any other steelworks, is legally committed to respect, develop, and promote *accident prevention standards.*

The *Carsid joint venture* itself coordinates the steel manufacturing process. The sintering phase to *prepare iron ore* is the responsibility of *Duferco Sintering Plant* while *Usinor Coking Plant, distills coal* to turn it onto *coke*. The *sinter charge* and *coke* are used by the Cokerill Blast Furnace to produce *cast iron* by *removing oxygen from sinter.* The *Duferco Steel Making Plant transforms cast iron into steel* to produce slabs and billets for the *Carlam Rolling Mill* in charge of the *hot rolling* tasks. Carlam (Carolo-Laminoir). *Sogepa*, the public partner, has the responsibility to *develop regional initiatives* to promote Carsid activities, particularly in the Walloon Region and in Belgium.

Figure 6 abstracts the joint venture structures explored in the case studies of Figures 4 and 5. The case studies suggest a number of constraints to supplement the basic pattern:

Figure 6. The joint venture pattern



- Partners depend on each other for providing and receiving resources
- Operation coordination is ensured by the joint manager actor, which depends on partners for the accomplishment of these assigned tasks
- The joint manager actor must assume two roles: A private interface role to coordinate partners of the alliance and a public interface role to take strategic decisions, define policy for the private interface, and represent the interests of the whole partnership with respect to external stakeholders

Part of the Joint Venture pattern specification is in the following:

Role JointManagerPrivateInterface Goal CoordinatePatterns
Role JointManagerPublicInterface
Goal TakeStrategicDecision
SoftGoal RepresentPartnershipInterests
Actor Partner

and the following structural (global) properties must be satisfied:

- ∀jmpri1, jmpri2 : J ointManagerPrivateInterface
  (jmpri1 = jmpri2)

  [Only one instance of the joint manager]

- *∀p1, p2 : P artner, dep : Dependency*
  (((dep.depender = p1 ∧ dep.dependee = p2)
  ∨ (dep.depender =
  p2 ∧ dep.dependee = p1)) → (dep.type = resource))

  [Only resource dependencies between partners]

- *∀jmpri : JointManagerPrivateInterface, p : Partner, dep : Dependency((dep.dependee = p ∧ dep.depender = jmpri)*
  →dep.type = task)

  [Only task dependencies between partners and the joint manager, with the joint manager as depender]

- *∀jmpri : JointM anagerPrivateInterf ace, jmpui : JointM anagerPublicInterf ace, dep : Dependency ((dep.depender = jmpri ∧ dep.dependee = jmpui)*
  → (dep.type = goal ∨ dep.type = softgoal))

[Only goal or softgoal dependencies between the joint manager roles]

- *∀dep : Dependency, p1 : Partner ((dep.depender = p1 ∨ dep.dependee = p1) →*
  ((∃p2 : P artner(p1≠ p2 ∧ (dep.depender = p2 ∨ dep.dependee = p2))
  ∨ (∃jmpi : J ointM anagerP rivateInterf ace ((dep.depender = jmpi ∨dep.dependee = jmpi))))

  [Partners only have relationships with other partners or the joint manager private interface]
- *∀dep : Dependency, jmpi : J ointM anagerP rivateInterf ace ((dep.depender = jpmi ∨dep.dependee = jpmi) →*
  ((∃p : P artner((dep.depender = p ∨ dep.dependee = p))) ∨
  (∃jmpui : J ointM anagerP ublicInterf ace ((dep.depender = jmpui ∨dep.dependee = jmpui))))

  [The joint manager private interface only has relationships with the joint manager public interface or partners]

EVALUATION

Patterns can be compared and evaluated with quality attributes (Shaw & Garlan, 1996), also called non-functional requirements (Chung, Nixon, Yu, & Mylopoulos, 2000). For instance, the requirements seem particularly relevant for organizational structures (Do, Faulkner, & Kolp, 2003; Kolp, et al., 2006):

- **Predictability** (Woods & Barbacci, 1999): Actors can have a high degree of autonomy (Wooldridge & Jennings, 1995) in the way that they undertake action and communication in their domains. It can be then difficult to predict individual characteristics as part of determining the behavior of the system at large. Generally, predictability is in contrast with the actors' capabilities to be adaptive and responsive: Actors must be predictable enough to anticipate and plan actions while being responsive and adaptive to unexpected situations.
- **Security:** Actors are often able to identify their own data and knowledge sources and they may undertake additional actions based on these sources (Woods & Barbacci, 1999). Strategies for verifying authenticity for these data sources by individual actors are an important concern in the evaluation of overall system quality since, in addition to possibly misleading information acquired by actors, there is the danger of hostile external entities spoofing the system to acquire information accorded to trusted domain actors.
- **Adaptability:** Actors may be required to adapt to modifications in their environment. These may include changes to the component's communication protocol or possibly the dynamic introduction of a new kind of component previously unknown or the manipulations of existing actors.

Generally, adaptability depends on the capabilities of the single actors to learn and predict the changes of the environments in which they act (Weiss, 1997), and also their capability to make a diagnosis (Horling, Lesser, Vincent, Bazzan, & Xuan, 1999), that is, being able to detect and determine the causes of a fault based on its symptoms. However, successful organization environments tend to balance the degree of reactivity and predictability of the single actors with their capabilities to be adaptive.

- **Coordinability:** Actors are not particularly useful unless they are able to coordinate with other agents. Coordination is generally (Jennings, 1996) used to distribute expertise, resources, or information among the actors (actors may have different capabilities, specialized knowledge, different sources of information, resources, responsibilities, limitations, charges for services, etc.), solve interdependencies between actors' actions (interdependence occurs when goals undertaken by individual actors are related), meet global constraints (when the solution being developed by a group of actors must satisfy certain conditions if is to be deemed successful), and to make the system efficient (even when individuals can function independently, thereby obviating the need for coordination, information discovered by one actor can be of sufficient use to another actor that both actors can solve the problem twice as fast).

Coordination can be realized in two ways:

1. **Cooperativity:** Actors must be able to coordinate with other entities to achieve a common purpose or simply align their local goals. Cooperation can either be communicative in that the actors communicate (the intentional sending and receiving of signals) with each

other in order to cooperate or it can be non-communicative (Doran, Franklin, Jennings, & Norman, 1997). In the latter case, actors coordinate their cooperative activity by each observing and reacting to the behavior of the other. In deliberative organizations, actors jointly plan their actions so as to cooperate with each other.

2. **Competitivity:** Deliberative negotiating organization (Doran et al., 1997) are like deliberative ones, except that they have an added dose of competition. The success of one actor implies the failure of others.

- **Availability:** Actors that offer services to other actors must implicitly or explicitly guard against the interruption of offered services.
- **Fallibility-tolerance:** A failure of one actor does not necessarily imply a failure of the whole organization. The organization then needs to check the completeness and the accuracy of information and knowledge transactions and workflows. To prevent failure, different actors can have similar or replicated capabilities and refer to more than one actor for a specific behavior.
- **Modularity:** (Shehory, 1998) increases efficiency of service execution, reduces interaction overhead, and usually enables high flexibility. On the other hand, it implies constraints on inter-organization communication.
- **Aggregability:** Some actors are parts of other actors. They surrender to the control of the composite entity. This control results in efficient workflow execution and low interaction overhead. However, it prevents the organization to benefit from flexibility.

As an illustration, we evaluate the patterns with respect to coordinativity, predictability, fallibility-tolerance, and adaptability. The evaluation can be done in a similar way for the other non-functional requirements. Due to the lack of space, we refer the author to the bibliography for the other attributes.

The structure-in-5 improves coordinativity among actors by differentiating the data hierarchy (the support actor) from the control hierarchy—supported by the operational core, technostructure, middle agency, and strategic apex. The existence of three different levels of abstraction (1) Operational Core; (2) Technostructure, Middle Line, and Support; (3) Strategic Apex addresses the need for managing predictability. Besides, higher levels are more abstract than lower levels: Lower levels only involve resources and task dependencies while higher ones propose intentional (goal and soft-goal) relationships. Checks and control mechanisms can be integrated at different levels of abstraction assuming redundancy from different perspectives and considerably increase fallibility-tolerance. Since the structure-in-5 separates data and control hierarchies, integrity of these two hierarchies can also be verified independently. The structure-in-5 separates independently the typical components of an organization, isolating them from each other and allowing them dynamic adaptability. But since it is restricted to no more than five major components, more refinement has to take place inside the components themselves.

The joint venture supports coordinativity in the sense that each partner actor interacts via the joint manager for strategic decisions. Partners indicate their interest, and the joint manager either returns the strategic information immediately or mediates the request to some other partners. However, since partners are usually heterogeneous, it could be a drawback to define a common interaction background. The central position and role of the joint manager is as a means for resolving conflicts and preventing unpredictability. Through its joint manager, the joint-venture proposes a central communication controller. It is less clear

how the joint venture pattern addresses fallibility-tolerance, notably reliability. However, exceptions, supervision, and monitoring can improve its overall score with respect to these qualities. Manipulation of partners can be done easily to adapt the structure by registering new ones to the joint manager. However, since partners can also exchange resources directly with each other, existing dependencies should be updated as well. The joint manager cannot be removed due to its central position. Table 2 summarizes the strengths and weaknesses of the reviewed patterns.

To cope with non-functional requirements and select the pattern for the organizational setting, we go through a means-ends analysis using the non functional requirements (NFRs) framework (Chung et al., 2000). We refine the identified requirements to sub-requirements that are more precise and evaluates alternative organizational patterns against them, as shown in Figure 7. The analysis is intended to make explicit the space of alternatives for fulfilling the top-level requirements. The patterns are represented as operationalized requirements (saying, roughly, "model the organizational setting of the system with the *pyramid*, *structure-in-5*, *joint venture*, *arm's-length* ... pattern").

The evaluation results in contribution relationships from the patterns to the non-functional requirements, labeled "+", "++", "–", "– –". Design rationale is represented by claims drawn as dashed clouds. They make it possible for domain characteristics (such as priorities) to be considered and properly reflected into the decision making process, e.g., to provide reasons for selecting or rejecting possible solutions (+, –). Exclamation marks (! and !!) are used to mark priority requirements while a check-mark "√" indicates an accepted requirements and a cross " X " labels a denied requirement.

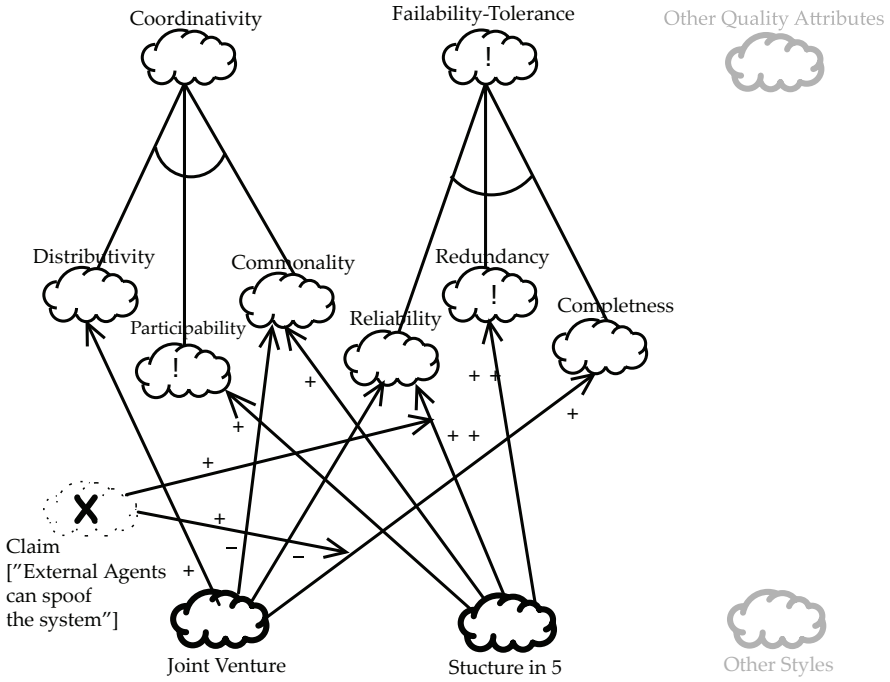
Relationships types (AND, OR, ++, +, –, and – –) between NFRs are formalized to offer a tractable proof procedure. AND/OR relationships correspond to the classical AND/OR decomposition relationships: if requirement R0 is AND-decomposed (respectively, OR-decomposed) into *R*1,*R*2,...,*Rn*, then all (at least one) of the requirements must be satisfied for the requirement R0 to be satisfied. So, for instance, in Figure 7 Coordinativity is AND decomposed into Distributivity, Participability, and Commonality. Relationships "+" and "–" model respectively a situation where a requirement contributes positively or negatively towards the satisfaction of another one. For instance, in Figure 7 Joint Venture contributes positively to the satisfaction of Distributivity and negatively to the Reliability. In addition, relationships "++" and "– –" model a situation where the satisfaction of a requirement implies the satisfaction or denial of another goal. In Figure 7 the satisfaction of structure-in-5 implies the satisfaction of requirements Reliability and Redundancy.

The analysis for selecting an organizational setting that meets the requirements of the system to build is based on propagation algorithms presented in Giorgini, Mylopoulos, Nicchiarelli, and Sebastiani (2002). Basically, the idea is to assign a set of initial labels for some requirements of the graph, about their satisfiability and deniability, and see how this assignment leads to the labels propagation for other requirements. In particular, we adopt from Giorgini, Mylopoulos, Nicchiarelli

Table 2. Strengths and weaknesses of some patterns

	Structure-in-5	Joint-Venture
Coordinativity	++	+
Predictability	+	+
Fallibility-Tolerance	++	+
Adaptability	+	+

Figure 7. Partial evaluation for organizational patterns



and Sebastiani (2002) both qualitative and a numerical axiomatization for goal (requirements) modeling primitives and label propagation algorithms that are shown to be sound and complete with respect to their respective axiomatization. The following is a brief description of the qualitative algorithm.

To each requirement *R*, we associate two variables. *Sat*(*R*), *Den*(*R*), ranging in {*F, P, N}* (full, partial, none) such that $F > P > N$, represents the current evidence of satisfiability and deniability of the requirement *R*. For example, $Sat(R_i) >= P$ states there is at least partial evidence that A_i is satisfiable. Starting from assigning an initial set of input values for $Sat(R_i)$, $Den(R_i)$ to (a subset of) the requirements in the graph, we propagate the values through the propagation rules of Table 3. Propagation rules for AND (respectively OR) relationships are min-value function for satisfiability (max-value function) and max-value function (min-value function) for deniability. A dual table is given for deniability propagation.

The schema of the algorithm is described in Figure 8. *Initial*, *Current*, and *Old* are arrays of pairs $Sat(R_i)$, $Den(R_i)$, one for each R_i of the graph, representing respectively the initial, current, and previous labeling status of the graph.

The array *Current* is first initialized to the initial values *Initial* given in input by the user. At each step, for every requirement R_i, $Sat(R_i)$, $Den(R_i)$ is updated by propagating the values of the previous step. This is done until a fixpoint is reached, that is, no updating mode is possible (*Current == Old*). The updating of $Sat(R_i)$,

Table 3. Propagation rules for satisfiability in the qualitative framework. A dual table is given for deniability propagation.

	+ $G2 \rightarrow G1$	- $G2 \rightarrow G1$	++ $G2 \rightarrow G1$	-- $G2 \rightarrow G1$
Sat(*G*1) *Den*(*G*1)	*min{ Sat*(*G*2), *P }* *N*	*min{ Sat*(*G*2), *P }*	*Sat*(*G*2) N	*N* *Sat*(*G*2)

Figure 8. Schema of the label propagation algorithm

```
1 Current=Initial;
2 do
3 Old=Current;
4 for each R_i do
5 Current[i] = Update label(i, Old);
6 until not (Current==Old);
7 return Current;
8 for each Rel_i s.t. target(Rel_i ) == Ri
do
9 satij = Apply Rules Sat(i, Rel_i , Old);
10 denij = Apply Rules Den(R_i, Rel_i ,
Old);
11 return max(max_i (sati_i ), Old[i].
sat),
12          max(max_i (deni_i ), Old[i].
den);
```

Den(R_i)works as follows. For each relation Rel_i incoming in G_i, the satisfiability and deniability values $sati_i$ and $deni_i$ derived from the old values of the source requirements are computed by applying the rules of Table 3. Then, it returns the maximum value between those computed and the old values.

A REQUIREMENTS-DRIVEN METHODOLOGY

This research is conducted in the context of the *early requirements* phase of *Tropos* (Giorgini et al., 2004; Giorgini et al., 2005), a software development methodology for building multi-agent systems founded on the concepts of actor and goal.

The *Tropos* methodology adopts ideas from multi-agent systems technologies, mostly to define the detailed design and implementation phase, and ideas from requirements engineering and organizational modeling, where agents/actors and goals have been used heavily for early requirements analysis (Dardenne et al., 1993; Yu, 1995). In particular, the *Tropos* project adopts Eric Yu's *i** model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for analyzing an application during organizational modeling. The key assumption which distinguishes *Tropos* from other methodologies is that actors and goals are used as fundamental concepts for analysis and design during *all phases of software development*, not just requirements analysis. That means that, in the light of this paper, *Tropos* describes the organizational environment within which a system will eventually operate, as well as the system itself in terms of the same concepts and patterns. *Tropos* spans four phases of software development:

- Organizational modeling, concerned with the understanding of a problem by studying an organizational setting; the output is an organizational model which includes relevant actors, their goals and dependencies.
- Requirements analysis, in which the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design, in which the system's global architecture is defined in terms of subsystems, interconnected through data, control, and dependencies.
- Detailed design, in which behavior of each architectural component is defined in further detail.

CONCLUSION

Modelers need to rely on patterns, styles, and idioms to build their models, whatever the purpose. We argue that, as with other phases of software development, organizational modeling can be facilitated by the adoption of organizational patterns. This paper focuses on two such patterns and studies them in detail through examples, a formalization using Formal Tropos and an evaluation with respect to desirable requirements. There have been many proposals for software patterns (e.g., Kolp, Do, & Faulkner, 2005) since the original work on design patterns (Gamma, Helm, Johnson, & Vlissides, 1995). Some of this work focuses on requirements patterns. For example, Konrad and Cheng (2002) propose a set of requirements patterns for embedded software systems. These patterns are represented in UML and cover both structural and behavioral aspects of a requirements specification. Along similar lines, Fowler (1997) proposes some general patterns in UML. In both cases, the focus is on requirements analysis, and the modeling language used is UML. On a different path, Gross and Yu (2002) propose a systematic approach for evaluating design patterns with respect to non-functional requirements (e.g., security, performance, and reliability). Our approach differs from this work primarily in the fact that our proposal is founded on ideas from organization theory and strategic alliances literature. We have already described organizational patterns, but to be used for designing multi-agent system architectures (Kolp et al., 2006) and e-business systems (Kolp et al., 2004). Considering real world organizations as a metaphor, systems involving many software actors, such as multi-agent systems, could benefit from the same organizational models. In the present paper we have focused on patterns for modeling organizational settings rather than software systems, and emphasized the need for organizational abstractions to better match the operational environment of the system-to-be during organizational modeling.

REFERENCES

Anton, A. I. (1996). Goal-based requirements analysis. In *Proceedings of the 2nd International Conferance on Requirements Analysis, ICRE'96*, Colorado Spring (pp. 136-144).

Bates, L.D.V. (2006). *Advertising agency.* At http://www.ldv.be

Chung, L.K., Nixon, B.,Yu, E., & Mylopoulos, J. (2000). *Non-functional requirements in software engineering.* Kluwer Publishing.

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming 20*(1-2), 3-50.

Do, T. T., Faulkner, S., & Kolp, M. (2003). Organizational multi-agent architectures for information systems. In *Proceedings of the 5th International Conferance on Enterprise Information Systems, ICEIS'03*, Angers, France (pp. 89-96).

Doran, J. E., Franklin, S., Jennings, N.R., & Norman, T.J. (1997). On cooperation in multi-agent systems.*Knowledge Engineering Review, 12*(3), 309-314.

Dussauge, P., & Garrette, B. (1999). *Cooperative strategy: Competing successfully through strategic alliances.* Wiley & Sons.

Fowler, M. (1997). *Analysis patterns: Reusable object models.* Addison-Wesley.

Fuxman, A., Liu, L., Mylopoulos, J., Roveri, M., & Traverso, P. (2004). Specifying and analyzing early requirements in Tropos. *Requirements Engineering, 9*(2), 132-150.

Gamma, E., Helm, R., Johnson, J., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.*Addison-Wesley.

Giorgini, P., J. Mylopoulos, E. Nicchiarelli, & R. Sebastiani (2002). Reasoning with goal models. In*Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002)*, Tampere, Finland (pp. 167-181).

Giorgini, P., Kolp, M., & Mylopoulos, J. (2002). Multi-agent and software architecture: A comparative case study. In *Proceedings of the 3rd International Workshop on Agent Software Engineering, AOSE'02*, Bologna, Italy (pp. 101-112).

Giorgini, P., Kolp, M., Mylopoulos, J., & Castro, J. (2005). A requirements-driven methodology for agent-oriented software. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent oriented methodologies* (pp. 20-46). Hersey, PA: Idea Group Publishing.

Giorgini, P., Kolp, M., Mylopoulos, J., & Pistore, M. (2004). The Tropos methodology. In M-P. G.F. Bergenti & F. Zambonelli (Eds.), *Methodologies and software engineering for agent systems* (pp. 89-105). Kluwer.

GMT. (2006). GMT Consulting Group. http://www.gmtgroup.com/

Gomes-Casseres, B. (1996). *The alliance revolution: The new shape of business rivalry.* Harvard University Press.

Gross, D., & Yu, E. (2002). From non-functional requirements to design through patterns. *Requirements Engineering*, 6(1), 18-36.

Horling, B., Lesser, V.,Vincent, R., Bazzan, A., & P. Xuan (1999). *Diagnosis as an integral part of multi-agent adaptability* (Technical Report UM-CS-1999-003), University of Massachusetts.

Jennings, N. R. (1996). Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare & N. R. Jennings (Eds.), *Foundations of distributed artificial intelligence* (pp. 187-210). Wiley.

Kolp, M., Giorgini, P., &Mylopoulos, J. (2002a). Information systems development through social structures. In *Proceedings of the 14th International Conferance on Software Engineering and Knowledge Engineering, SEKE'02*, Ishia, Italy (pp. 183-190).

Kolp, M., Giorgini, P., & Mylopoulos, J. (2002b). Organizational multi-agent architecture: A mobile robot example. In *Proc. of the 1st International Conferance on Autonomous Agent and Multi Agent Systems, AAMAS'02*, Bologna, Italy (pp. 94-95).

Kolp, M., Giorgini, P., & Mylopoulos, J. (2003). Organizational patterns for early requirements analysis. In *Proceedings of the 15th International Conferance on Advanced Information Systems, CAiSE'03*, Velden, Austria (pp. 617-632).

Kolp, M., Giorgini, P., & Mylopoulos, J. (2006). Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems, 13*(1), 3-25.

Kolp, M., Do, Y., & Faulkner, S. (2004). A social-driven design of e-business system. In *Software Engineering for Multi-Agent Systems III, Research*

Issues and Practical Applications, Edinburg, UK (pp. 70-84).

Kolp, M., Do, T., & Faulkner, S. (2005). Introspecting agent-oriented design patterns. In S. K. Chang (Ed.), *Handbook of software engineering and knowledge engineering, Vol. 3, Recent Advances* (pp. 151-177). World Scientific.

Konrad, S., & Cheng, B. (2002). Requirements patterns for embedded systems. In *Proceedings of the 10th IEEE Joint International Requirements Engineering Conference, RE'02,* Essen, Germany (pp. 127-136).

Mintzberg, H. (1992). *Structure in fives : Designing effective organizations.* Prentice-Hall.

Morabito, J., Sack, I., & Bhate, A. (1999). *Organization modeling: Innovative architectures for the 21st century.* Prentice Hall.

Scott, W.R. (1998). *Organizations: Rational, natural, and open systems.* Prentice Hall.

Segil, L. (1996). *Intelligent business alliances : How to profit using today's most important strategic tool.* Times Business.

Shaw, M., & Garlan, D. (1996). *Software architecture: Perspectives on an emerging discipline.* Prentice Hall.

Shehory, O. (1998). *Architectural properties of multi-agent systems* (Technical Report CMU-RI-TR-98-28), Carnegie Mellon University.

Wautelet, Y., Kolp, M., & Achbany, Y. (2006). *S-tropos: An iterative spem-centric software project management process* (Technical Report IAG Working paper 06/01), IAGISYS Information Systems Research Unit, Catholic University of Louvain, Belgium (http://www.iag.ucl.ac.be/wp/).

Weiss, G. (Ed.). (1997). *Learning in DAI systems.* Springer Verlag.

Woods, S.G., & Barbacci, M. (1999). *Architectural evaluation of collaborative agent-based systems* (Technical Report SEI-99-TR-025), SEI, Carnegie Mellon University, Pittsburgh.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review, 2*(10).

Yoshino, M.Y., & Rangan, U.S. (1995). *Strategic alliances: An entrepreneurial approach to globalization.* Harvard Business School Press.

Yu, E. (1995). *Modelling strategic relationships for process reengineering.* Unpublished doctoral thesis, University of Toronto, Department of Computer Science.

This work was previously published in the International Journal of Enterprise Information Systems, edited by A. Gunasekaran, Volume 3, Issue 3, pp. 1-22, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.28
A Multi–Methodological Approach to Study Systems Development in a Software Organization

Paivi Ovaska
South Karelia University of Applied Sciences, Finland

ABSTRACT

Large-scale systems development is a complex activity involving number of dependencies that people working together face. Only a few studies concentrate on the coordination of development activities in their organizational context. This research study tries to fill at least part of this gap by studying how systems development process is coordinated in practice. The study uses a multimethodological approach to interpret coordination of systems development process in a contemporary software organization in Finland. The methodology is based on the empirical case-study approach in which the actions, conceptions, and artefacts of practitioners are analyzed using within-case and cross-case principles. In all the three phases of the study, namely multi-\site coordination, requirement understanding, and working with systems development methods, both the qualitative and quantitative methods were used to an understanding of coordination in systems development. The main contribution of this study is to demonstrate that contemporary systems development is much more complex and more driven by opportunity than is currently acknowledged by researchers. The most challenging part of the research process was the combination of qualitative and quantitative methods, because of the lack of multimethodological work done in IS discipline.

INTRODUCTION

Large-scale systems development is a complex activity involving number of dependencies that people working together face. Furthermore, in

distributed and multiparty information systems projects, there is even a larger number of stakeholders involved, and a great number of dependencies. These dependencies create a need for coordination that requires continuous effort by developers. Broadly defined, coordination is management of interdependencies between activities (Malone & Crowston, 1994). This definition assumes that if there are no interdependencies, there is nothing to coordinate. The activities can be activities or objects; everything that has dependencies requires coordination (Malone & Crowston, 1994). Coordination is an inherent aspect of work in any organization and takes place in the form of meetings, scheduling, milestones, planning, and processes.

To systematize coordination, many methods and process models have been proposed over the years. These models mainly focus on the sequence of steps used by developers to develop information systems. The status of methods as a whole has been described as a "method jungle," as "an unorganized collection of methods more or less similar to each other" (Jayaratna, 1994). Though these methods and process models have helped companies to gain certification and attain global standards, they do not take into account interpersonal interactions and various other social aspects of development organizations. The development of new methods has tended to be more technology driven, often being influenced by the introduction of improved techniques and software tools (Nandhakumar & Avison, 1999).

Only a few studies concentrate on the process of systems development and coordination of development activities in their organizational context. Kraut and Streeter (1995) found that coordination becomes much more difficult as project size and complexity increases. Apparently, complexity increases when the project is located in multiples sites. Communication is a salient part of coordination, and it has been observed (e.g., Allen, 1977) that distance affects the frequency of communication. Communication delays and breakdowns taking place in software development projects are discussed in several studies (Curtis, Krasner, & Iscoe, 1988; Kraut & Streeter, 1995; Herbsleb et al., 2000).

Systematic surveys of the existing literature in both Information Systems (Wynekoop & Russo, 1997) and Software Engineering (Glass, Vessey, & Ramesh, 2002) fields revealed that most of the research papers in these fields consist of normative research in which concept development is not based on empirical grounding or theoretical analysis, but merely upon the author's opinions. Because of that, many researchers (e.g., Curtis et al., 1988; Orlikowski, 1993) call for more empirical studies in order to understand how information systems are developed in today's organizations and how development work is coordinated in various types of organizations before development of new methods.

This research study tries to fill at least part of this gap by further clarifying how systems development process is coordinated in practice. This objective is reached by conducting a series of empirical studies of two systems development projects in a contemporary organization that competes in the information technology business. We study the early systems development, which we consider to be most important phases in development process: requirement elicitation and architecture design. In this study, the actions, conceptions, and artefacts of practitioners are interpreted and analyzed using multimethodological approach. By multimethodological approach is meant research approach that uses different research methods structured as a set of guidelines or activities to assist in generating valid and reliable research results (Mingers, 2001). The objective for this research is twofold: (1) to understand how practitioners coordinate the systems development process and (2) to make a contribution to the theory and practice of systems development.

The rest of this chapter is structured as follows. First we describe the other research using multiple research methods and then the research

framework is created and the research process used is explained. After that, the research findings are summarized. The discussion of using this methodology by comparing it to framework proposed by Mingers (2001) as well the general experiences in using the methodology is given in the sixth section. Finally, we summarize the used methodology and research findings and give direction for future research.

RELATED RESEARCH

Combining different research methods has been the subject of much debated after 1990s. Mingers (2001) advocated multimethodological research on the grounds that both the target of the research and the research process were complex and multidimensional, requiring range of different approaches.

There have been numerous reviews of the information system research literature, each with different purpose, but few of them specifically considered combining methodologies. Mingers (2003) reviewed systematically the six IS top journal published in the time period from the year 1993 to 2000 and studied the extent of multimethodological research. The results of this study clearly show a lack of multimethodological research published within IS. In the following table is shown the proportion of multimethodological research papers in these journals.

Mingers also looked at the particular combinations of methods used. He found that the vast majority (70%) involve only observation, survey, case study, and interview. In these studies, the mixture of hard (quantitative) and soft (qualitative) methods were rare.

Example of studies in IS discipline using multimethodological approaches are Markus (1994), Ngwenyama and Lee (1997), Trauth and Jessup (2000), and Ormerod (1995), but the literature around the theme is quite scarce.

In relation to IS field, in the area of Management Science, a research of Munro and Mingers (2002) showed that the use of multimethodological

Table 1. Proportion of multimethodological papers (1993, 2000) adapted from Mingers (2003)

Year	ISR (%)	ISJ (%)	MISQ (%)	EJIS (%)	AMIT (%)	JIT (%)	Average (%)
1993	40	0	50	0	-	21	22
1994	13	20	35	29	-	13	25
1995	0	11	7	18	33	8	13
1996	28	14	18	15	60	0	23
1997	18	27	19	7	63	17	25
1998	0	0	20	18	25	16	13
1999	0	22	24	21	56	21	24
2000	10	31	13	0	33	33	20

Note: Information Systems Research (ISR); Information Systems Journal (ISJ); MIS Quarterly (MISQ); European Journal of Information Systems (EJIS); Accounting, Managements and Information Technology (AIMT); Journal of Information Technology (JIT)

methods were very common among practitioners within organizational interventions and was driven by the demands of complex real-world problem situations. According to this study, practitioners used two, three or even more methods together in this area.

DEVELOPING THE METHODOLOGY

Studied Organization

The research study was carried out in a software development department of an international ICT company. The software development department was an internal partner for the company's business units. The development of applications and services was assigned to an in-house software development unit (Internal Development Unit, later referred as IDU) or outsourced companies. The use of IDU for development of new services was mandated by the company's top management. IDU had approximately 150 employees that had formerly focused on R&D work in the company. During the past few years, it had tried to improve its software skills and processes in order to make its development more effective and also to prove its capability to other business units. All the business units of the company did not agree with IDU's processes and did not trust in its software development capability. Their attitudes towards IDU competencies in software development were quite suspicious, mainly because of IDU's history as an R&D department. Quite often business units preferred outsourcing instead of developing in-house.

The projects, called here the "DS project" and the "EC project" respectively developed mobile services to both the global and domestic telecommunication markets. A major goal of both projects was the renewal of old platform architecture to allow the services to be better and easier modifiable, maintainable, and scalable.

The DS project developed a directory service platform for international markets. The project was partitioned into two subprojects to facilitate easier management. Partitioning was carried out on the basis of the architecture and technology: one subsystem had a highly distributed, component-based architecture (Server) and the other was a centralized subsystem (Client), which handled authentication, authorization and user interfaces. The functionality of the services required subsystems to communicate only through an extensible and configurable interface.

The goal of the EC project was to develop an Electronic Commerce mobile service platform. The system was intended to enable organizers or their sponsors to promote their products in all kinds of events, such as ice hockey and football games. The system was composed of two subsystems: the platform in which the services are run (Platform subsystem) and the toolbox (Tool subsystem). The Tool subsystem allowed adding, configuring and simulating of services to run in a Windows PC and with a service platform in an UNIX environment.

The actual systems development projects took place in IDU during the year 2001 and they were planned and organized according to a traditional waterfall model with distinct requirement elicitation, analysis, software design, implementation, and testing phases.

Shaping the Research Problem

The basic notion of systems development, namely systems development as a process that involves real people in real environments (e.g., Lyytinen, 1987), formed the ground for constructing our research methodology. To truly understand systems development, it is imperative to study people-systems development practitioners as they solve real development problems in real environments. Therefore, as

Rosen (1991) puts it, "to understand social process one must get inside the world of those generating it." This kind of goals favoured interpretive approach that enables researcher to understand human thought and action in social and organizational contexts (Walsham, 1995).

Systems development in organizational environment arouses different kinds of dependencies. The structure of the software system itself creates dependencies between software elements, while the structure of the development process creates dependencies between software developers. Each of these both shapes and reflects the development process; therefore, the other objective of selecting the research methodology to this study was to focus on different aspects of systems development (both technical and social dependencies) and therefore to get richer understanding of the dependencies.

Yet another goal was also to be more convinced of information accuracy also discussed in Yin (1994). Quantitative evidence can indicate relationships which may not be salient to the researcher. It also can keep researchers from being carried away by vivid, but false impressions in qualitative data, and it can bolster findings when it corroborates those findings from qualitative evidence (Eisenhardt, 1989).

Above objectives favoured selecting the interpretive approach (Walsham, 1995) and integrating different methods according to multimethodological approach (Eisenhardt, 1989; Mingers, 2003). Used methods are described in another section along with description of research process. In the following, the basic principles of the shaping the research methodology are explained in more detail.

Phases Adopted in Research

The research included three phases: studies on how architecture affects a multi site development project, studies on how requirements were shaped and interpreted during the systems development and how this process is to be estimated, and a study on how practitioners work with systems development methods. It is briefly explained how these three phases shaped the research problem:

Phase One: Multisite Coordination

The objective of this phase was to clarify the systems development problems related to software architecture and investigate how practitioners cope with these problems in systems development. This phase consisted of two parts: a qualitative study about social complexities and a quantitative analysis of technical complexities. During the phase, the problems found in the qualitative study evolved more to coordination and communication problems for which architecture provided a tool. In the quantitative study, the understanding of the architecture as a size predictor in the project cost estimation got its basic shape. The results from this phase of study are given in another section.

Phase Two: Requirement Understanding Process

In the beginning of phase two, we tried to find coordination problems or problems related to software architecture, but observed that the problems were more related to requirement understanding and organizational conflicts. This observation shaped the research problem towards the interpretation of the requirement understanding process and how this could be measured to get better estimates of the project timetable along with the architecture measures from phase one. The results from this phase are given in another section. At the end of this phase, the observations so far suggested that methods in the organization played an important role in the case study projects. This led us to shape the study towards the interpretation of the role of methods and their use in the studied organization.

Phase Three: Working with Systems Development Methods

In this phase, the comparisons of the results of phase one and phase two according to their similarities and differences (cross-case analysis). During this analysis, it appeared that the coordination and the requirements understanding in the projects were the result of using and adapting methods based on the practitioner's background, experience and the development situation at hand. The results from this phase are given in another section.

RESEARCH PROCESS

In this section, the research process is explained. Figure 1 explains the flow of research phases and tasks. After that the process is explained in a more detailed level.

Preparing for the Study

The beginning of our research study included an initial definition of the research question, a selection of cases and crafting instruments and protocols.

Each of the phases of the study had the research questions of its own. Table 2 summarizes the research questions of each three phases.

The selection of cases relied on the theoretical sampling principle (Glaser & Strauss, 1967), in which cases are chosen as extreme situations and polar types in which the process of interest is "transparently observable." The sampling strategy of the current study was designed to be built around projects displaying problems in systems

Figure 1. Research process

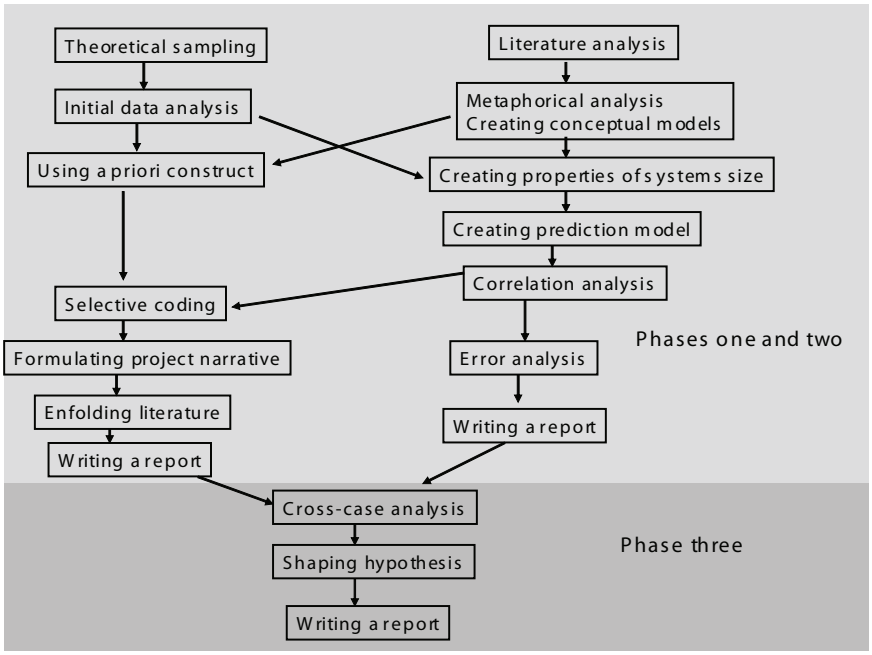


Table 2. Research questions in each three phases of the study

Phase one	**Phase two**	**Phase three**
What kind of coordination problems related to software architecture was present during the systems development? How did these problems differ in the same-site and multi-site environments?	How were software requirements shaped and interpreted during systems development?	How practitioners use systems development methods in projects? How methods support systems development practitioners in projects?

development, big problems that caused delays to the project's timetable. Within these projects in the studied organization, we chose projects of polar types: one project had problems inside the project, the other problems with the customer; one was smaller and the other one bigger; they both produced service platforms for different business areas. The analysis revealed that the projects had even more different features, such as the orientation, attitudes, and experience of the participants, and the communication between participants that extended the emergent theory (Eisenhardt, 1989). To facilitate iteration and comparison, which is an inevitable feature of the grounded theory method (Locke, 2003), these two projects were analyzed one by one.

Data Collection

During the study, most of the data was collected from project extensive documentation (see Table 3) based on the dynamic process of data collection (Glaser & Strauss, 1967), where samples were extended and focused according to the emerging needs of the theoretical sampling. In both case projects, the project documentation data was complemented with interviews among project participants (customer, project managers, designers, etc.).

The interviews were all tape-recorded and completely transcribed. The length of the interviews varied from half an hour (focused interviews) to 2 hours (group interview). Several hundreds of pages of project documentation, the transcribed interviews, and 170,000 lines of program source codes were analysed during the studies.

The data for the quantitative statistical analysis in both phases one and two was collected from the architecture and component design specifications, source code, project management database, and bills from subcontractors. Some metrics (or property values) of the software system were calculated based on these documents. In the phase this metric was related to software architecture and in the phase two to requirements evolution. In the project management database, the data included the time spent on each task (development effort) by the project participants. In the cases where foreign consultants were involved in the development work, the development effort data was taken from the subcontractors' bills.

Seed Categories

Specification of a priori constructs (Eisenhardt, 1989) or also called seed categories (Miles & Huberman, 1984) can help research. In phase one, a notion of the common object from Malone

Table 3. Data available from the case projects

Data/Document/Artifact DS project	**Data/Document/Artifact EC project**
15 Progress Report (from Project Manager)	15 Progress Reports (from Project Manager)
Project management Software: Plan vs. Actual costs	Project management Software (Niku Workbench): Plan vs. Actual costs, development effort
11 Project Steering Group Meeting Minutes	16 Project Steering Group Meeting Minutes
46 Project Group Meeting Minutes	26 Project Group Meeting Minutes
Project Plan	Project Plans
Functional Specifications	Requirement Specification document
Requirement Catalogue	Project Quality Criteria document
Risk analysis document	Architecture Specification
Project Quality Criteria document	26 Module Specifications
Architecture Descriptions	22 Tool subsystem UI specifications
Module Specifications	Kick-off presentation, Steering Group kick-out meeting minutes
Group Interview with project participants	Focused interviews of three BU members (development manager, team leader, designer)
Group interview slides	Focused interviews of four IDU members (steering group representative, project architect, 2 project designers)
Source code (Lines of Codes) 138,000 LOC	Source code (Lines of Codes) 32,000 LOC

and Crowston's coordination theory (Malone & Crowston, 1990, 1994) was used to interpret the coordination in the project. In phase two, the concept of a technology frame of reference (Orlikowski & Gash, 1994) was used to interpret the requirement understanding in the project.

Data Analysis

In these studies, we used the principle of within-case and cross-case analysis to interpret the findings in different phases of this thesis (Eisenhardt, 1989). In the within-case analysis "the overall idea is to become intimately familiar with each case as a standalone entity" allowing unique patterns to emerge before trying to generalize patterns across cases (Eisenhardt, 1989). In the first two phases of the research, the qualitative data analysis was based on the grounded theory (Glaser & Strauss, 1967; Strauss & Corbin, 1990). In those phases, quantitative data analysis with a simple linear regression method (Lawson, 1995) was carried out. The principal researcher conducted the qualitative analysis alone in all the phases, but, in phases one and two, two other researchers did the quantitative data analysis and provided complementary insights (Eisenhardt, 1989) into the qualitative analysis in our discussions. As outsiders of the qualitative data collection, they were able to look at the data more with greater objectivity (Nandhakumar & Jones, 1997), which facilitated a more reliable data analysis as a whole.

In the first and second phase of the study, the qualitative data analysis started quite early, right after the data on the project meeting minutes was collected. In the first phase, data collection continued with design specifications simultaneously with the analysis of the data on the project meeting minutes. In phase two, the collection of requirement specification data started while the analysis of the data on the project meeting minutes was still being performed. The quantitative data collection started as soon as some initial findings of the qualitative data analysis were made. Such overlap of data collection and analysis is strongly recommended by Eisenhardt (1989) and Strauss and Corbin (1990). To help manage the quite extensive amount of information and the analysis process, the Atlas.ti (Scientific Software, 2001) tool was used. It helped in the analysis process, for example in the retrieval of categories, memo making and recording of semantic relationships.

The quantitative data analysis was hypotheses testing in nature and naturally used a priori constructs. Both hypotheses were based on the initial findings of the corresponding qualitative studies. To the statistical data analysis, we used the Matlab Optimization Toolbox (MathWorks Inc., 2003).

In the following we describe the data analysis process in the three phases of the study.

Phase One

Data analysis started with open coding according to Miles and Huberman (1984). In open coding, a researcher tries to find relevant categories based on research questions. Open coding started with the finding of problems and deviations related to coordination and software architecture (relevant categories in this study) in the project progress, using mainly project meeting minutes and the group interview as a data source. We further used architecture and design specifications to help pinpoint the problems. We observed in total 329 deviations and problems related to software architecture and coordination.

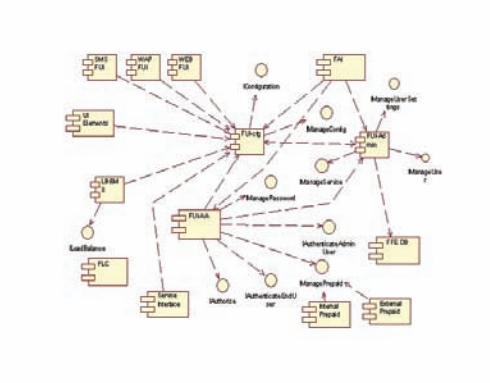
Data analysis continued with the axial coding, in which categories found in open coding were refined, developed further, and related. In this phase, we used a notion of common object as a seed category based on Malone and Crowston's coordination theory (Malone & Crowston, 1990, 1994) to help in the interpretation of coordination problems in the project. We identified two types of common objects from the study, namely component and development activity. The following example (Figure 2) shows a translated excerpt of a transcript after axial coding. This example shows how common objects were interpreted from the project material.

In the selective coding phase, where the "core category," or central category that ties all other categories in the theory together, was identified and related to other categories (Glaser & Strauss, 1967). The identification of common objects helped in finding the interdependencies between activities that caused coordination problems in the project. We identified three interdependencies between components and three interdependencies between development activities. These interdependencies are explained in another section.

After finding the interdependencies, we tried to find an answer to the research question of how the problems in coordination differed in multisite and same same-site environments (cross-case analysis). To find these differences, we compared the two subprojects and analyzed the differences in the coordination processes between them.

In the quantitative analysis, we used metaphorical analysis (Frost & Morgan, 1983; Lakoff & Johson, 1980; Schultze & Orlikowski, 2001) to help understand the architecture of the system. According to our metaphor of architecture drawing, we identified three categories that described the architecture of our case study system best. Within these categories, we attempted to select the properties, which were simple and could be

Figure 2. Translated excerpt of a transcript after axial coding in the phase one

”On the Server side, we gained experience from new technologies, like XML, XSL and CORBA”	Technical orientation of the Server Common object: development activity
”We had a lot of problems with Client and Server synchronization. The Client was first and the Server was behind, it should be wice versa ”	Interface and interdependence problems Common object: component
”The Client-Server interface was dependent on core resources”	
”AA and MS&LB need communication with architect and designer”	Communication problems Common object: development activity
	Interface problems Common object: component
”The second actual build was made 24th August, FFE not ready for testing”	Assembly order problems Common object: development activity

calculated based on project design specifications. We chose the simple linear prediction model to analyze the correlation between architecture properties and systems development effort. From these 7 property values for six components, a total of 42 property values were calculated. From these property values, we calculated coefficient values using Matlab Optimization toolbox. In the end of this quantitative process, we calculated the model errors to determine the quality of our cost effort estimation model and analyzed the results based on coefficient values.

Phase Two

Phase two started with the identification of problems and deviations in the project progress. During the development, these were issues that were brought to the project meetings for discussion and decision-making. The steering and project group meeting minutes were the main sources for problem and change identification. We observed in total 150 problems and deviations related to project progress. Most of the concerns brought to the steering group were related to the

other subsystem and its requirements. Also the system development styles and strategies caused concerns.

To better understand the requirements of the system, we investigated in more detail the requirements specification document. We were able to extract only four initial requirements that were related to the other subsystem. Our analysis continued as we used three conceptual models of both subsystems developed in the qualitative study.

Through these models, we were able to grasp how the subsystems evolved through different phases of systems development. The content of these conceptual models suggested to us that the other subsystem's requirements changed considerably during the process. This led us into investigating further why this subsystem's requirements changed so much, while the other subsystem's requirements remained stable.

To answer this question, we made focused interviews among the project participants to identify the reasons for these changes. Project participants were asked to reflect on the project's history by showing the analysis and implementation models of the system and to describe their understanding of what happened in the project between the requirement analysis and implementation phases. Four of the interviewed project participants were from the development side and three from the business side. Business side representatives were also asked about the competences and processes of development side during the time the project was running and how these competences and processes had evolved after that. The interviews were audio taped and fully transcribed to preserve all the details.

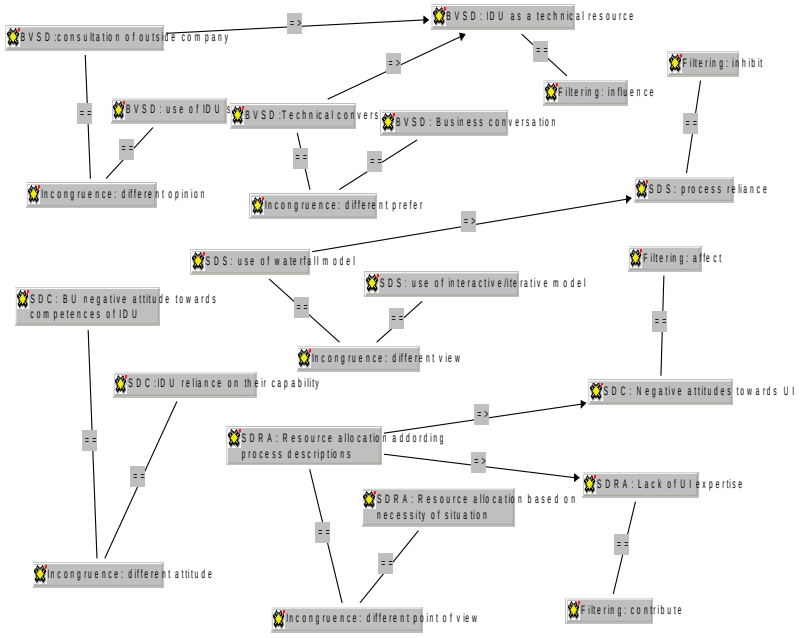
Based on the interviews, we observed that requirements did not change that much during the project, but the understanding of them changed. The open coding proceeded in parallel, treating each interview as confirmation or further development of results from earlier findings. During this process, the categories developed gradually. First, we identified quite concrete concerns of system development. In further analysis, we found more subtle contextual attitudes and expectations about how systems development should be performed. These attitudes and expectations were so strongly visible in the data that we could interpret them as technology frames (Orlikowski & Gash, 1994). In further analysis, we used technology frames as a priori construct or lenses to the data. This further analysis consisted of group analysis, cross-group analysis and re-examination of the categories. This iterative examination of the data yielded five categories of technology frames, which were used as a basis for the next phase, axial coding.

During the axial coding, we identified four processes of stereotypical "tensions" between these attitudes and expectations, which affected how project participants emphasized these frames of understanding in different phases of the project. These tensions and explained in another section.

In the selective coding phase, the causal relationships between categories were recorded with Atlas.ti's (Scientific Software, 2001) semantic network capability. Figure 3 shows an example of such a network diagram. In Figure 3, the boxes represent categories, the arrows the interpreted causalities between them, and the lines simple associations between the categories. The abbreviations business value of system development (BVSD), system development strategy (SPS), system development capability (SDC), and system development resource allocation (SDRA) correspond to the categories of frames of understanding.

Based on project material, interviews, and analysis, we formulated the project narrative to trace how the participants' attitudes and expectation influenced the systems development process. In the end of the research process, the project narrative was sent by e-mail to the project manager to get her opinion on the correspondence of the narrative with the reality. She adjusted some details, which did not affect the main findings.

Figure 3. An example of semantic network diagram using Atlas.ti (Scientific Software, 2001)



In the quantitative analysis, we formulated the metric describing the identified requirement evolution in the project. The metric was quite simple: it calculated the concepts found during the analysis and implementation models of the system. These analysis and implementation models were formed based on the project specifications. In the statistical analysis, we used the same simple prediction model as in phase one of the study. The other metrics needed were chosen based on simplicity and wide usage. Using this prediction model, we calculated the correlation between metrics chosen and the development effort. In the end of the process, we formulated the model errors to determinine the reliability of our prediction model and analyzed the results.

Phase Three

In phase three, we used cross-case analysis to interpret the final results in this thesis (Eisenhardt, 1989). We searched for cross-cased patterns to compare the multi-site and same-site development by listing their similarities and differences (Eisenhardt, 1989). We selected pairs of cases and listed similarities and differences between each pair. In this phase, the number of cases was actually three because one of the case projects consisted of two subprojects. The cross-category table produced in this process is shown in Table 4.

Table 4. Cross-category table between projects in the studies

Category	*Case Project 1* *subproject A*	*Case Project 1* *subproject B*	*Case Project 2*
Adaptation to method	yes	no	yes
Problems	minor technical inside project	inside project, coordination and architecture understanding	conflicts with client, requirement understanding
Timetable estimation	no major problems	problems	problems
Orientation and attitudes	Architect's technical orientation	Architect's business orientation	Designer's positive attitude towards UI and windows
Experience	Experienced project group	Unexperienced project group	Unexperienced project group
Communication	good	bad	in the beginning bad, later good
Understanding the system	fairly good	not very good	in the beginning no, learning to understand
Understanding the development situation	yes	no	in the beginning no later yes
Method purpose	communication and coordination	method use failed	communication and learning
The most meaning metric (relative to development effort)	Coupling (NIC)	-	Requirements Creep

Shaping the Hypothesis

From the within-case analysis, the cross-case analysis and overall impressions, tentative tenses and concepts and their relationships begin to emerge, which is called hypothesis shaping (Eisenhardt, 1989). The idea is that researchers constantly compare emergent theory and "raw" data—iterating towards a theory with closely fit data (Eisenhardt, 1989).

In the hypothesis shaping, we used the semantic network diagram capability of Atlas.ti software (Scientific Software, 2001). This semantic network is shown in Figure 2. This network shows the relationships between the core categories used to interpret the role of methods.

Finishing and Reporting the Studies

Eisenhardt (1989) distinguishes the phase "enfolding literature." By this phase, Eisenhardt means the comparison of the findings with similar and conflicting literature. The aim of this phase is to raise confidence, creative thinking, and the validity, generalizability, and conceptual level of the findings. Yin (Yin, 1994) refers to this as "analytic generalization" to distinguish it from the more typical statistical generalization that generalizes from a sample to a population. In phase one, the main comparisons were done with Malone and Crowston's (Malone & Crowston, 1990, 1994) coordination theory and cost estimation literature. The comparisons of phase two were made with traditional requirement engineering approaches and existing sociotechnical approaches to requirement elicitation, especially the concept of a technological frame. All these provided conflicting and similar concepts and patterns, which all provided an alternative and more creative view to our findings. In phase three, the findings were compared to a few empirical studies of the role of methods in systems development.

SUMMARY OF FINDINGS

Phase One: Multisite Coordination (Ovaska & Bern, 2004; Ovaska, Rossi, & Marttiin, 2004)

In phase one, we discovered six different coordination processes to explain most of the coordination problems related to software architecture in the case studies. These six processes can be summarised as follows:

1. *Interfaces*: Managing interfaces between system components
2. *Integration*: Managing the assembly of system components, enabling their integration into a full, working system architecture
3. *Interdependence*: Managing the interdependencies between system components
4. *Communication*: Managing the communication processes between development participants
5. *Responsibility*: Managing the allocation of responsibilities for key decisions related to the software architecture and subsystem designs
6. *Perspectives*: Managing the different perspectives development participants brought to the project and managing negotiations related to these perspectives

This phase suggest that technical dependencies among software components (interfaces, integration, and interdependence) create social dependencies (communication, responsibility, perspectives) among the software developers implementing those components. It also supports the Malone and Crowston's coordination theory by highlighting the importance of the coordination of interdependencies between activities instead of purely coordinating activities in systems development.

Phase Two: Requirement Understanding (Ovaska, 2004; Ovaska, Rossi, & Smolander, 2005)

In phase two of the research process four categories of technology frames were identified that could explain attitudes and expectations that affected people's understanding of requirements and which were directly related to the ISD process. These categories can be summarised as follows:

1. *Business value of systems development*: attitudes and expectations about the re-

lationship between business and system development.
2. *Systems development strategy*: attitudes and expectations about the system development lifecycle model and associated ISD processes.
3. *Systems development capability*: the assumptions, attitudes and expectations surrounding the various competencies of project participants in different areas of system development
4. *Systems development resource allocation*: the assumptions, attitudes and expectations as regards scheduling, budgeting, and prioritisation of systems development activities

Within these four frames it was possible to identify stereotypical "tensions" which emerged as the projects unfolded. These tensions and the processes associated with them had important effects on how the project participants emphasized various technology frames during the different phases of the project:

1. *Incongruence*: this tension emerged when understanding, attitudes, or expectations differed among the stakeholders, causing conflicts and misunderstanding
2. *Filtering*: this tension surfaced when a stakeholder of the development process left something out of scope because of his/her particular perspective, attitude, series of expectations, or experience of ISD
3. *Negotiation*: this refers to tensions associated with the various negotiations between project participants and took place in order to resolve incongruities
4. *Shifting*: this emerged when the understanding of a frame changed (called a "frame shift"). After a frame shift, the parties involved achieved an understanding of a frame that was more aligned and suitable for the current situation than before the shift.

In this phase, we observed that preconceptions, attitudes, and expectations about systems development among project participants filtered the understanding of software requirements, negotiating between project participants resolved the issues caused by filtering and shifts in these attitudes and expectations facilitated changes in the understanding of requirements. In spite of this observed filtering, shifting, and negotiation, the developed system exceeded the customer's needs and expectations even though it was delivered late. The results of this phase suggest that the current conceptions regarding requirement elicitation do not correspond with the needs of practice. The traditional requirement engineering research concentrates on detecting and representing requirements and ensuring that they are complete and consistent. It sees requirements mainly in a system context assuming that they already exist somewhere ready to be picked in the requirement elicitation phase. Our study has consequences beyond such a view. It suggests that requirement elicitation is in practice an ad-hoc and iterative process involving political, cognitive and social aspects that affect the interpretation of requirements during the whole project lifetime.

Phase Three: Working with Methods (Ovaska, 2005)

During the cross-case analysis in phase three of the research process, we discovered the five categories which summarised various ways in which methods were used in the target projects:

1. *Methods used as tools*: rather than following predefined phases and tasks practitioners used ISD methods as if they were a set of tools which could be variously deployed or discarded as the need arose
2. *Methods used to coordinate*: practitioners primarily used methods to coordinate systems development

3. *Methods used a communication language*: methods were used to provide common communication language between various project participants, and the methods thus helped to set the agenda for many discussions
4. *Methods as a means to aid understanding of the system and its requirements*: methods were used to help develop an understanding of the system in terms of requirements and expectations of users and align these with various technical and nontechnical issues which arose
5. *Understanding the development context*: methods were used by participants to make sense of the development situation as it shifted over time

This phase suggests that working with methods is complex social interaction and a mutual understanding process between project participants. The methods were used in our case study as a tool for communication and learning and as a resource in project planning among system development participants, not as a list of phases to be followed in detail. Method use requires communication between project participants to adopt the method to the development situation. Without communication, methods use failed. When participants gained common understanding

of the appropriate development strategy and method, the method use became a learning process. In this learning process, system development participants changed their understanding of the system, according to changing development situations. Without the common understanding of the system between project participants, the estimation of project timetable and resources did not succeed. Based on the observations of our study, we argue that methods are used more as support and guide the development than strict phases to be followed in detail.

DISCUSSION

In this section the multimethodological approach used in this study is discussed related to the multimethodological guidelines set out by Mingers (2001). The guidelines suggest that in designing research one should consider the following domains:

- *The context of the research*—in particular the relationships between the research situation and task, the methods and theories available, and the researchers' own competencies and commitments
- *The dimensions of the research situation*—in particular, the material, social, and personal aspects

In the following, these domains are explained in more detail along with the comparison our multimethodological approach to it. Finally, we summarize the general experiences of using the methodology and its limitations.

The Context of the Research

The multimethodology used in this study lies on the most close to dominant type of design in which one method is the main approach with contributions from the others. The qualitative methods were our main method in all phases contributed from qualitative statistical analysis. The reasons for such a design were formed from the nature of systems development as a social process.

The professional and scientific background of principal researcher provides some explanations for understanding the selection of the research method and also the role of the researcher in the interpretive research process. Because of her educational background in engineering, she also wanted to get some "hard evidence" of the projects, maybe to become more assured of the reliability of the research. She wanted to carry out some statistical, quantitative calculations that would

support the qualitative work. During the learning process in the research work, the quantitative calculations faded into the background and shifted more towards an interpretive approach.

Although the principal researcher did not work on the chosen projects, she acquired "deep familiarity" (Nandhakumar & Jones, 1997) with the research context and its actors during the 5 years working in that company. During the observation period, she was fully involved in the activities of the company. During the analysis period, she was not involved in the activities, but had full access to all project documents gaining access to information that would not otherwise have been divulged. The used data in the study was mainly documents gathered during the projects. Without her personal experience in the company, it would have been difficult to interpret the local meanings, dominant perceptions, tacit knowledge, and nonverbal communication (Nandhakumar & Jones, 1997) from the documentation. Without the deep familiarity with the research context and its actors, it would not be possible to gain additional insight in the actors' interpretations, their motivation, and perspectives (Nandhakumar, Jones, 1997) in the focused interviews carried out during the study. Her role as a researcher was somewhere between an outside observer and involved researcher (Walsham, 1995); it can be called an "involved observer" participating in the work of the company before the analysis period.

The Dimensions of the Research Situation

According to Mingers (2001), each research situation is the combination of three worlds: the material world, the social world, and the personal world. Each domain has different modes of existence and different epistemological possibilities as followed:

- *The material world* is outside and independent of human beings characterized as objective in the sense that it is independent of the observer although our observations and descriptions of it are not. Our relationship to this world is one of *observation.*
- *The personal world* is the world of our own individual interpretations, experiences, thoughts, and beliefs. We do not observe it, but *experience* it.
- *The social world* is the world that we share with others in a particular social system and *participate* in it.

In comparing our research situation and methodology to this framework, all these worlds were present to the following extent. The study covered material aspects, such as architecture as a predictor of system size or requirement creep as a measure of requirement evolution. The interpretive analysis of documentation and interviews explored the meaning of coordination and requirements understanding for particular individuals; and group interviews and their interpretations revealed the social aspects of coordination and requirements understanding.

General Experiences

In general, we experienced the methodology highly iterative learning process, in this process the research themes and questions changed during its phases.

The most challenging part of the research was the combination of quantitative and qualitative methods, mainly because of the lack of the empirical frameworks to guide the work; therefore the frameworks would be helpful in designing and developing this kind of research.

Limitations of the Methodology

A critical issue for researchers concerns the generalizability of the results of their work, and Yin (1994) notes that this issue is often rose with respect to case studies. Different arguments for

the generalizability of case study research have been given (Dutton & Dukerich, 1991; Eisenhardt, 1989; Walsham, 1995; Yin, 1994). It is argued that in case study research, the identified concepts and categories are compared to theoretical concepts and patterns, unlike in statistical generalization from a sample to a population. Still, due to the nature of this study in which the understanding of method use was interpreted on the basis of separate phenomena found in one organization, the generalization of the use of methods may be limited. Therefore, the understanding gained in these studies provides a basis for understanding similar phenomena in the same settings rather than enabling the understanding of phenomena in other contexts.

SUMMARY AND FUTURE WORK

This chapter has described the multimethodological approach to study coordination of systems development process. We used the principle of within-case and cross-case analysis to interpret the findings in different phases of the study. In all these three phases, we used both qualitative and quantitative methods to get richer and more reliable understanding from coordination phenomena.

The three phases of the study have provided a rich picture of different aspects of systems development in the software organization. In the first phase of the study, we examined the role of architecture in coordination and cost estimation in a multisite software development from quantitative and qualitative viewpoints. The second phase involved two studies, one qualitative and the other quantitative, on the evolving requirement understanding process and the measurement of this process. The third phase was a study based on the first two studies on the role of methods and how practitioners work with them using principle of cross-case analysis. The research process of these phases was explained.

We experienced the methodology highly iterative and adaptive learning process, in which the research themes and questions evolved during its phases. The study covered material aspects, such as architecture as a predictor of system size as well as personal and social aspects. Highly interpretive analysis of documentation and interviews covered aspects of particular individuals whereas the group interviews and their interpretation revealed the social aspects of coordination of systems development work.

The main contribution of this study is to demonstrate that contemporary systems development is very complex and more driven by opportunity than is currently acknowledged by researchers. We need more knowledge of the current and emerging business and organizational contexts in which and for which these systems are developed. This study suggests that further rich, multimethodological research is urgently needed in order to build a picture of how communities of practice make sense of systems development. Only in this way we can begin to find ways to understand and address the problems in systems development.

The most challenging part of the research process was the combination of qualitative and quantitative methods because of the lack of multimethodological work done so far; therefore the frameworks would be helpful in designing and developing multimethodological approaches.

In the future, our multimethodological framework needs deeper work towards a theory-building approach, especially related to analysis process in the phase three. Also, the replication of our study in other contexts and redefinition of the framework based on experiences is necessary.

REFERENCES

Allen, T. (1977). *Managing the flow of technology.* Cambridge, MA: MIT Press.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Dutton, J., & Dukerich, J. (1997). Keeping an eye on the mirror: Image and identity in organizational adaptation. *Academy of Management Journal, 34*, 517-554.

Eisenhardt, K.M. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532-550.

Frost, P.J., Morgan, G. (1983). Symbols of sense-making: The realization of the framework. L. R. Pondy, P.J. Frost, G. Morgan & R.T.C. Dandrike (Eds.), *Organizational Symbolism* (pp. 419-437). New York: Wiley.

Glaser, B., & Strauss, A.L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Chicago, Adline.

Glass, R., Vessey, I., & Ramesh, V. (2002). Research in software engineering: An analysis of the literature. *Information and Software Technology, 44*(8), 491-506.

Jayaratna, N. (1994). *Understanding and evaluating methodologies: NIMSAD a systematic framework*. New York, McGraw-Hill.

Hersleb, J.D., Mockus, A., Finholt, T.A., & Grinter, R.E. (2000). Distance, dependencies and delay in a global collaboration. *ACM Conference on Computer-Supported Cooperative Work*. Philadelphia (p. 319, 328), December 2-7.

Kraut, R.E., & Streeter, L.A. (1995). Coordination in software development. *Communications of the ACM, 38*(3), 69-81.

Lakoff, G., & Johson, M. (1980). *Metaphors we live by*. Chicago: The University of Chicago Press.

Lawson, C.L. (1995). *Solving least squares problems*. Society of Industrial and Applied Mathematics.

Lee, A (1997). Integrating positivist and interpretivist approaches to organizational research. *Organization Science, 2*, 342-264.

Locke, K. (2003). *Grounded theory in management research*. London: Sage.

Lyytinen, K. (1987). Different perspectives on information systems: Problems and solutions. *ACM Computing Surveys, 19*(1), 5-46.

Malone, T.W., & Crowston, K. (1990, October 7-10). What is coordination theory and how can it help design cooperative work systems? In *Conference on Computer Supported Cooperative Work (CSCW '90)*, Los Angeles, (pp. 357-370). ACM Press.

Malone, T.W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys, 26*(1), 87-110.

Markus, I. (1994). Electronic mail as the medium of managerial choice. *Organizational Science, 5*(4), 502-527.

Mathworks Inc. (1995). Matlab. Retrieved August 20, 2008 from http://www.mathworks.com

Miles, M.B., & Huberman, A.M. (1984). *Qualitative data analysis: A sourcebook of a new methods*. Beverly Hills, CA: Sage.

Mingers, J. (2001). Combining IS research methods: Towards a pluralist methodology. *Information Systems Research, 12*(3), 240-259.

Mingers, J. (2003). The paucity of multimethod research: A review of the information systems literature. *Information Systems Journal, 13*, 233-249.

Munro, I., & Mingers, J. (2002). The use of multimethodology in practice—Results of a survey of practitioners. *Journal of Operational Research Society, 53*, 369-378.

Nandhakumar, J., & Avison, D. (1999). The fiction of methodology development: A field study

of information systems development. *Information Technology & People, 12*(2), 176-191.

Nandhakumar, J., & Jones, M. (1997). Too close for comfort? Distance and engagement in interpretive information systems research. *Information Systems Journal, 7*(2), 109-131.

Ngwenyama, O., & Lee, A. (1997). Communication richness in electronic mail: Critical social theory and the contextuality of meaning. *MIS Quarterly, 21*, 145-167.

Orlikowski, W.J. (1993). Case tools as organizational change: Investigating incremental and radical changes in systems development. *MIS Quarterly, 17*(3).

Orlikowski, W.J., & Gash, D.C. (1994). Technological frames: Making sense of information technology in organizations. *ACM Transactions on Information Systems, 12*(2), 174-207.

Ormerod, R. (1995). Putting soft OR methods to work: Information systems strategy development at Sainsbury's. *Journal of the Operational Research Society, 46*(3), 277-293.

Ovaska, P. (2004, April 14-17). Measuring requirement evolution – a case study in the e-commerce domain. In *6th International Conference on Enterprise Information Systems (ICEIS(3))*, Porto, Portugal, INSTICC (pp. 669-673).

Ovaska, P. (2005). Working with methods: Observations on the role methods in systems development. O. Vasilegas, A. Capinskas, W. Wotjowski, W.G. Wotjowski, J. Zupneic, & S. Wrycza (Eds.), *Information systems development, advances in theory, practice and education, ISD 2004* (pp. 185-197). Springer.

Ovaska, P., & Bern, A. (2004, June 7-11). Architecture as a predictor of system size – a metaphor from construction projects. In *16th International Conference on Advanced Information Systems Engineering (CAISE '04 Forum)*, Riga, Latvia, Riga Technical University (pp. 193-203).

Ovaska, P., Rossi, M., & Marttiin, P. (2004). Architecture as a coordination tool in multi-site software development. *Software Process Improvement and Practice, 8*(4), 233-248.

Ovaska, P., Rossi, M., & Smolander, K. (2005). Filtering, negotiating and shifting in the understanding of information systems requirements. *Scandinavian Journal of Information Systems, 17*(1), 31-66.

Rosen, M. (1991). Coming to terms with the field: Understanding and doing organizational ethnography. *Journal of Management Studies, 28*, 7-24.

Schultze, U., & Orlikowski, W.J. (2001). Metaphors of virtuality: Shaping an emergent reality. *Information and Organization, 11*(1), 45-77.

Scientific Software. (2001). Atlas.ti-the knowledge workbench. Retrieved June 3, 2008, from http://www.atlasti.de/

Strauss, A., & Corbin, J. (1990). *Basics of qualitative research: Grounded theory procedures and applications.* Newbury Park, CA: Sage.

Trauth, E.M., & Jessup, L. (2000). Understanding computer-mediated discussions: positivist and interpretive analyses of group support system use [Special issue on intensive research]. *MIS Quarterly, 24*(1), 43-79.

Walsham, G. (1995). Interpretive case studies in IS research. *European Journal of Information Systems, 4*, 74-81.

Wynekoop, J., & Russo, N. (1997). Studying system development methodologies: An examination of research methods. *Information Systems Journal, 7*(1), 47-65.

Yin, R.K. (1994). *Case study research: Design and methods* (2nd ed.). Newbury Park: Sage.

ADDITIONAL READINGS

Andersson, S., & Felici, M. (2001, September 10-13). Requirements evolution: From process to product oriented management. In *Proceedings of the 3rd International Conference on Product Focused Software Process Improvement,* Kaiserslautern, Germany (pp. 27-41). Springer Verlag.

Andersson, S., & Felici, M. (2002, August 26-29). Quantitative aspects of requirement evolution. In *Proceedings of the 26th Annual International Conference on Computer Software and Applications Conference (COMPSAC 2002),* Oxford, England (pp. 27-32). IEEE Society

Baskerville, R., Travis, J., & Truex, D.P. (1992). Systems without method: The impact of new technologies on information systems development projects. In J.I. DeGross (Ed.), *Transactions on the impact of computer supported technologies in information systems development* (pp. 241-260).

Boehm, B. (1987). Improving software productivity. *IEEE Computer, 20*(8), 43-58.

Brodner, P. (2006). Behind the IT productivity paradox: The semiotic nature of IT artefacts. In F. Meyer (Ed.), *2006 International Federation of Automation and Control (IFAC) Conference on Automation Based on Human Skill* (ABoHS), Nancy, France.

Brooks, F.P.J. (1995). *The mythical man-month - 20th anniversary edition*. Boston, MA: Addison-Wesley.

Benbasat, I., D. Goldstein & Mead M. (1987) The case study research strategy in studies of information systems. *MIS Quarterly,* 11(3),369-386.

Benbasatam, I. & Weber, R. (1996) Rethinking 'diversity' in information systems research. *Information Systems Research.7*(4): 389-399.

Calloway, L.J., & Ariav, G. (1991). Developing and using qualitative methodology to study relationships among designers and tools. In E. Nissen, H. Klein, & R. Hirschheim (Eds.), *Information systems research: Contemporary approaches and emergent traditions* (pp. 175-193). Amsterdam, North-Holland.

Fitzgerald, B. (1998). An empirical investigation into the adoption of systems development methodologies. *Information & Management, 34*(6), 317-328.

Galliers, R. (1993). Research issues in information systems. *Journal of Information Technology, 8*(2), 92-98.

Gersick, C. (1988). Time and transition in work teams: Toward a new model of group development. *Acade*.

Iivari, J., & Maansaari, J. (1998). The usage of systems development methods: Are we stuck to old practices? Information and Software Technology, *40*(9), 501-510.

Jick, T.D. (1983). Mixing qualitative and quantitative methods: Triangulation in actions. In J. Van Maanen (Ed.), *Qualitative methodology* (pp. 135-148). Beverly Hills, CA: Sage.

Kidder, T. (1982). *Soul of a new machine*. New York: Avon.

Landry, M., & Banville, C. (1992). A disciplined methodological pluralism for MIS research. *Accounting, Management & Information Technology, 2*(2), 77-97.

Mintzberg, H. (1979). An emerging strategy of direct research. *Administrative Science Quarterly, 24*(4), 582-589.

Munro, I., & Mingers, J. (2002). The use of multimethodology in practice & results of a survey of practitioners. *J Opl Res Soc., 53,* 369-378.

Orlikowski, W.J. (1993). Case tools as organizational change: Investigating incremental and radical changes in systems development. *MIS Quarterly, 17*(3), 309-340.

Pinfield, L. (1986). A field evaluation of perspectives on organizational decision making. *Administrative Science Quarterly, 31*(3), 365-388.

Pohl, K. (1994). Three dimensions of requirements engineering: Framework and its application. *Information Systems, 19*(3), 243-258.

Robey, D. (1996). Diversity in information systems research: Threat, promise and responsibility. *Information Systems Research, 7*(4), 400-408.

Suchman, L. (1987). Plans and situated action. Cambridge: Cambridge University Press.

Truex, D.P., Baskerville, R., & Travis, J. (2001). Amethodical systems development: The deferred meaning of systems development methods. *Accounting, Management and Information Technologies, 10*(1), 53-79.

van Lamsweerde, A. (2000). Requirements engineering in the year 00: A research perspective. In *Proceedings of the International Conference on Software Engineering (ICSE 2000),* Limerick, Ireland (pp. 5-19).

Weill, P., & Broadbent, M. (1998). *Leveraging the new Infrastructure*. Boston, MA: Harvard Business School Press.

Wiegers, K.E. (1999). *Software requirements.* Microsoft Press.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition.* Norwood, NJ: Ablex.

Wynekoop, J., & Russo, N. (1995). Systems development methodologies: Unanswered questions. *Journal of Information Technology, 10*(2), 65-73.

This work was previously published in Information Systems Research Methods, Epistemology, and Applications, edited by A. Cater-Steel & L. Al-Hakim, pp. 162-182, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 5.29
Integrating Usability, Semiotic, and Software Engineering into a Method for Evaluating User Interfaces

Kenia Sousa
University of Fortaleza, Brazil

Albert Schilling
University of Fortaleza, Brazil

Elizabeth Furtado
University of Fortaleza, Brazil

ABSTRACT

We present artifacts and techniques used for user interface (UI) design and evaluation, performed by professionals from the human-computer interaction (HCI) area of study, covering usability engineering and semiotic engineering, which can assist software engineering (SE) to perform usability tests starting earlier in the process. Tests of various interaction alternatives, produced from these artifacts, are useful to verify if these alternatives are in accordance with users' preferences and constraints, and usability patterns, and can enhance the probability of achieving a more usable and reliable product.

INTRODUCTION

In a software development process (SDP), it is crucial for developers, customers, and users to interact in order to specify, generate, and evaluate the software. From software specification to its delivery, various kinds of tests must be performed, involving aspects such as: functionality, portability, performance, and usability. This work focuses on the context of usability, communicability, and functionality tests (e.g., appropriateness of a chosen interface design alternative to user preferences, consistency to a visual pattern, efficient execution of interactive tasks on interface objects, etc.).

Through our researches on tests in HCI and SE, and through our experiments on their integration in a SDP, we verified that HCI concepts facilitate the UI evaluation work performed by the test team of an interactive system under development. More specifically, by means of UI generation based on HCI models (e.g., task model), it is possible to evaluate the UI earlier (e.g., its functionality), independent of having the entire noninteractive specification ready. Prototypes, for instance, can represent UI design alternatives that may be tested early by HCI experts to verify if they are in accordance with user preferences, usability patterns, and so on.

This work presents a SDP to design and evaluate UIs, based on the integration of concepts, models, and activities of usability, semiotic, and software engineering.

This chapter is structured as follows: The "User-Interface Evaluation" section shows the contribution of each engineering area to UI evaluation; "The Process" section describes the UI design process; "The Evaluation Strategy" section describes the UI evaluation process, showing which concepts are used to perform tests and when they are performed; the "Case Study" section describes the case study in which we designed and evaluated UIs for the Brazilian System for the Digital Television (SBTVD); and, finally, the "Findings and Future Works" section describes findings and future works, and the "Conclusion" section concludes this work.

USER-INTERFACE EVALUATION

In this section, we present concepts and evaluation techniques from usability engineering, software engineering, and semiotic engineering.

Usability Engineering

Usability engineering is a set of activities that ideally take place throughout the lifecycle of the product, with significant activity at the early stages even before the UI has been designed. The need to have multiple usability engineering stages supplementing each other was recognized early in the field, though not always followed in development projects (Gould & Lewis, 1985).

In usability engineering, techniques and methods are defined aiming to assure a high usability level of the interactive UIs. Among them, we emphasize the application of ergonomic criteria in the UI design. Verification of these criteria in designed UIs is called heuristic evaluation, performed by usability experts without user participation. Evaluators examine the IS searching for problems that violate general principles of good UI design, diagnosing problems, obstacles or barriers that users will probably encounter during their interaction. In addition, methods to capture usability requirements attend to user preferences, restrictions, and use-context. A usability requirement can be derived from an interaction restriction; such as if part of the system needs to be implemented for palm-top devices.

The evaluation approaches from usability engineering suggests a structured sequence of evaluations based on "usability inspections methods" and on "usability tests".

Some inspection methods are: (1) heuristic evaluation, verification of usability heuristics (Nielsen, 1993); (2) review of guidelines, verification if the UI is according to a list of usability guidelines (Baranauskas & Rocha, 2003); (3) consistency inspection, verification of the consistency among the UIs related to terminology, color, layout, input and output format, and so on; and (4) cognitive walkthrough, simulation of the user "walking" through the UI to execute typical tasks.

Some usability test methods are: (1) thinking out loud, we request the user to verbalize everything he or she thinks while using the system, and we expect that their thoughts demonstrate how the user interprets each UI item (Lewis, 1982); and

(2) performance measures, quantification of some evaluated items to make future comparisons.

Software Engineering

Software engineering is composed of technologies and practices that are used in the development of software products, enhancing software productivity and quality, by providing a more systematic and controlled SDP (Sommerville, 2001).

In software engineering, there are various types of tests to be performed in each test stage, such as: usability tests to ensure that access and navigation through functionalities are appropriate for users; UI tests, to ensure a good functionality of the UI components and verify conformity to corporate patterns; and functionality tests, which are responsible for verifying if the generated software achieves all the proposed functionalities according to the customer's requests. Test cases (Myers, 2004) are normally generated and comprise procedures to be followed in test activities in order to deal with all possible situations when using the software, including basic flows, as well as error treatment and invalid data verification.

According to Pressman (1995), the main goal of test cases is to derive a set of tests that will probably reveal errors in the software. To achieve this goal, software engineering basically proposes two test categories: white-box and black-box tests.

For Pressman (1995), the white=box test must verify the internal part of the product, tests can be performed to guarantee that the components are integrated, and the internal operation achieves the performance level as specified in the requirements.

Functional tests, or black box tests, represent a test approach in which tests are derived from specifications of the system. In this kind of test, the evaluator is concerned with the functionality, not with the software implementation (Sommerville, 2001).

In these two test categories, software engineering defines four main types of tests: unit tests that are generally white-box tests; acceptance and regression tests that are usually black-box tests, and integration tests that blend this two categories.

Semiotic Engineering

Semiotic engineering is an HCI theory that emphasizes aspects related to the metacommunication designer user(s) via user-system communication, which passes through the UIs of interactive applications. The system is considered to be the "deputy" or a representative of the system designer (Souza, Barbosa, & Silva, 2001). The content of messages is the application usability model. Its expression is formed by the set of all interaction messages sent through the UI during the interaction process. The user plays a double role: interacting with the system and interpreting messages sent by the designer.

Semiotic engineering is essentially involved in test procedures with final users (empiric evaluation), aiming at system communicability analysis — based on qualitative evaluation, in which there are four phases: test preparation; labeling; interpretation, and formatting — and elaboration of the semiotic profile of the application to be evaluated. The techniques used in theses phases are: system-user observations, questionnaires, (somative-evaluation) inspections, interviews, filming,and so on.

Semiotic engineering is essentially present in tests with final users (e.g., empiric evaluation), aiming at analyzing the system communicability. A UI has a good level of communicability when it is able to successfully transmit the designer message to the user, allowing him or her to understand the system goal, the advantages of using it, how it works, and the basic UI interaction principles.

Evaluation

After studying about evaluation techniques, artifacts and approaches from software, usability, and

semiotic engineering we are able to conclude that an evaluation process can be seen under various perspectives.

Concerning software engineering, we noticed the importance of software quality concerning functionality, performance, portability, and other nonfunctional requirements. Its artifacts and techniques include the evaluation of these aspects in an objective manner.

Usability engineering focuses in providing more ease of use, ease of learning, and efficiency to interactive systems.

Semiotic engineering includes procedures that allow the evaluation of the quality of the interactivity of systems by observing the communication through messages of the user to the system.

Based on these perspectives, we believe that an approach for UI evaluation of interactive systems that integrates these approaches is able to guarantee a system with quality concerning functionality, usability, and interactivity, derived from software, usability, and semiotic engineering, respectively.

Next, we will describe a lightweight development process for interactive systems, called UPi (Sousa & Furtado, 2004), which integrates HCI and SE activities, artifacts, and professionals.

THE PROCESS

UPi can serve as a guide, providing useful steps and artifacts that can be tailored and customized when organizations intend to develop usable interactive systems. One of the best advantages of UPi is the idea to focus on activities, artifacts and guidelines that add value to the UI generation. With this approach, it can be integrated with any other process and inherit activities that are vital to the entire process, but that are better defined and solidified in other processes. For instance, project management, configuration and change management, implementation, and deployment activities are very well detailed in the RUP (Kruchten, Ahlqvist, & Bylund, 2001). Besides the RUP, UPi can also be applied in conjunction with ISO 13407 (ISO 13407, 1999), which already has other activities defined and validated, such as project planning, testing, and so on.

UPi is composed of activities that aim at designing UIs. These activities are based on RUP activities, but they follow different guidelines that take into consideration usability aspects.

In this work, we are integrating UPi with UPi-Test (to be presented in the next section) in order to guide professionals that are developing interactive systems to evaluate them throughout the entire development process.

Phase I: Inception

The main goal in this phase is to elicit requirements from users in order to develop an interactive system that best suits their needs through the execution of some activities (presented as follows). These requirements are documented through certain artifacts: use-case models, task models, usability requirements, and paper sketches.

Use-case models represent a well-established manner to define the system functionality, while *task models* can be used to detail use cases by breaking them down into tasks. *Usability requirements* represent users' preferences or constraints that can be part of a usable interactive system. *Paper sketches* focus on the interaction, UI components, and on the overall system structure, keeping the style guide secondary, without being too abstract.

The purpose of the *Elicit Stakeholder Needs* activity is to understand users, their personal characteristics, and information on the environment where they are located that have a direct influence on the system definition, and to collect special nonfunctional requirements that the system must fulfill, such as performance, cost, and device requests.

The purpose of the *Find Actors and Use Cases* and *Structure the Use-case Model* activities is

to define the actors (users or other systems) that will interact with the system and the functionality of the system that directly attend to users' needs and support the execution of their work productively.

The purpose of the *Detail a Use Case* activity is to describe the use case's tasks using the task model, to describe any usability requirements related to the use case, to define the system navigation based on the task model hierarchical structure, and to create paper sketches.

The purpose of the *Review Requirements* activity is to verify, with usability experts, if the paper sketches are in accordance to the task model and validate, with users, if the requirements are in conformance with their needs by showing them the elaborated paper sketches.

Phase II: Elaboration

The main goal in this phase is to transform the requirements in a representation that can be understood by UI designers and programmers. These representations are provided by the following artifacts: system architecture, UI Definition Plan, and drawing prototypes.

System Architecture is composed of smaller components that represent the main functionality of the entire system. The *UI Definition Plan* is a new artifact that aims to define which visual objects should be part of the UI. *Drawing prototypes* produce an accurate image of the system and they are useful to demonstrate patterns and style guides.

The purpose of the *Define and Refine the Architecture* activity is to (re)design the classes that represent the data that are handled by users while performing certain tasks.

The purpose of the *Define UI Plan* activity is to define which visual objects and which usability patterns can be part of the UI according to the nonfunctional requirements defined in the *Elicit Stakeholder Needs* activity.

The purpose of the UI Prototyping activity is to design a UI prototype in drawings following the description specified in the task models, in the UI definition plan and in the system architecture.

The purpose of the *Evaluate Prototype* activity is to verify if the UI prototypes are in accordance to usability principles and to validate with users if the UI prototypes are in conformance with their needs.

Phase III: Construction

The main goal of this phase is to implement and verify the accuracy of the components implemented and of the UI designed.

The purpose of the *Implement Components* activity is to develop the classes previously designed and implement the UI prototyped.

The purpose of the *Evaluate the Version of the System* activity is to verify if the functionality of the interactive system is in accordance with users' requirements.

Phase IV: Transition

The main goal of this phase is to deliver to the customer a system with high level of quality and usability.

The purpose of the *Deploy the System* activity is to make the system available for the customer.

The purpose of the *Evaluate the System* activity is to validate with users (by using the system in the deployment site) if the system conforms with their view of the system.

All of these phases are supported by processes concerning configuration and change management and project management, such as the RUP. This support is provided for the *Manage Change Requests* activity, which aims at evaluating the impact of change requests, deciding if they are to be included in the current iteration, and, if they are accepted, manage the changes in the appropriate artifacts.

Concerning the evaluation activities performed in each phase, they will be more thoroughly explained in the next section.

THE EVALUATION STRATEGY

The unified process for evaluating interactive systems "UPi-Test" (Schilling et al., 2005) has the same phases as the RUP (inception, elaboration, construction, and transition) (Kruchten, Ahlqvist, & Bylund, 2001). This process is based on the Unified Process for Interactive Systems, called UPi (Sousa & Furtado, 2004) and follows the approach to design UI prototypes dependent on the device (Coyette, Faulkner, Kolp, Limbourg, & Vanderdonckt, 2004).

Each phase is directly related to a specific area. This way, usability engineering supports the verification and validation in the inception and elaboration phases, software engineering supports verification in the construction phase, and semiotic engineering supports validation in the transition phase.

Figure 1 illustrates these four phases in the UPi-Test, each one with its flow of activities, artifacts, and techniques.

UPi-Test includes the verification and validation of usability and functionality of interactive systems UIs. Nonfunctional requirements are not in the scope of this work, such as aspects related to database (e.g., connection, integrity, etc.), security, and architecture. These aspects can be supported in future versions of this process.

Phase I: Inception

The inception phase is important in guaranteeing that the following phases achieve results to attend to users' and customers' usability goals. This phase has the constant participation of users in order to understand their requests, which are verified and validated according to usability engineering to allow the development of these requests. The description of the activities and artifacts in this phase is presented as follows.

Talk with Users and/or Customers

This activity consists of the first contact with users or customers, in which system analysts understand their profiles, objectives, and the scenario where they are included. In this activity, it is necessary to use a technique to elicit requirements.

We propose an initial informal talk in order to better understand users' environment. Then, we suggest the performance of interviews, with the use of a questionnaire that aids in the identification of users and/or customers and their intended goals, preferences, and possible constraints.

Obtain Users' and Customers' Preferences and Constraints

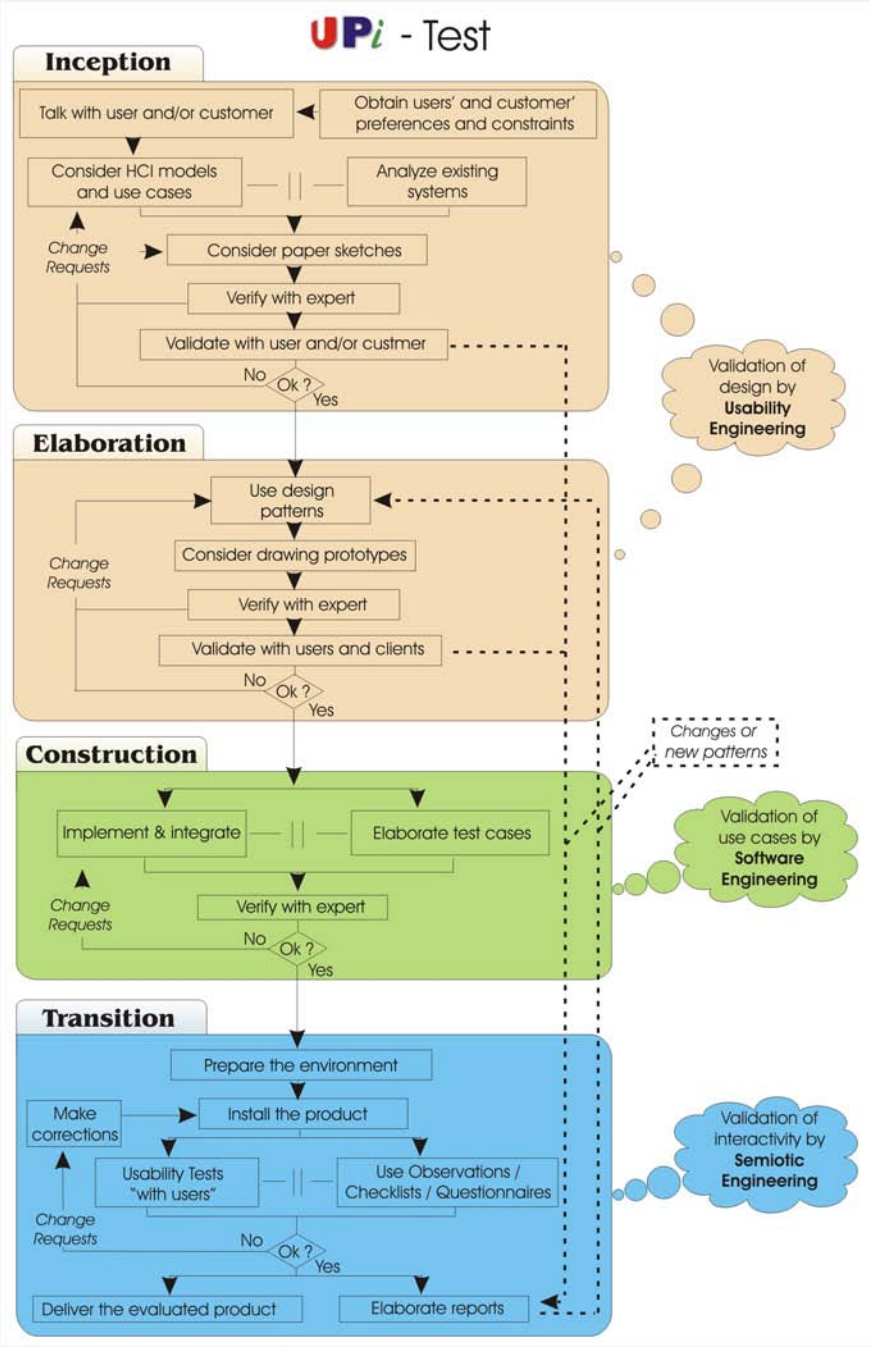
In this activity, system analysts aim at eliciting users' and customers' preferences and constraints in order to design UIs that attend their needs and also to help usability experts during evaluation and to help UI designers during prototyping. Some examples of preferences are: design colors, font styles, navigation schemes. Some examples of constraints are: technology constraints, such as platform, device, and so on.

Consider HCI Models and Use Cases

Use cases represent an artifact from software engineering that identifies the functionality of interactive systems; users' interactions, expressed through users' tasks; and the system's responses to perform these interactions (system's tasks).

Task models detail a use case or a group of related use cases by specifying users' tasks and system's tasks. It is useful to support UI designers in the elaboration of prototypes because it is easy to identify the necessary views and objects in the prototype from the task model hierarchical structure.

Figure 1. Evaluation process of UIs



Analyze Existing Systems

In this moment, it is important to analyze existing systems. This involves a comparative study of similar systems. These systems can be used as a reference in order for system analysts to propose new functionality or to choose design patterns, which are all useful for the design of UI prototypes.

Consider Paper Sketches

After talking with users and customers, the UI designer designs paper sketches. This prototype is verified and validated, as presented in the next activity.

Verify with Expert

This activity consists of the verification of paper sketches by the usability expert. The expert is concerned with verifying if users' goals, in terms of functionality, were included in the prototype, as well as if usability principles were used.

We propose that experts use the heuristic evaluation approach (Nielsen, 1993) as an inspection method. The usability heuristics will guide the usability expert in the process of verifying UIs' quality of use. We also suggest the use of the task model to verify if all the specified functionality was designed in the prototype.

When the expert notices that a functional requirement or any usability principle is missing, change requests can be made, which leads to changes in the models and new proposals of prototypes.

Validate with Users and Customers

After the verification with the expert, we propose a validation with users and customers so they can approve the generated artifacts. If the prototypes do not attend users' needs, change requests can be made, which leads to changes in the models (if new functionality is requested) and new proposals of prototypes (if changes in the navigation are requested). This process is repeated until the generated prototype attends users' preferences and needs.

This activity early in the process provides flexibility for users and customers to evaluate the evolution of the system, therefore, designers and users feel more confident with the UI design.

After the conclusion of the inception phase, the resulting artifacts are verified and validated as paper sketches.

Phase II: Elaboration

The elaboration phase is concerned with designing and evaluating drawing prototypes. In this phase, we use verification and validation techniques, such as heuristic evaluation and validations with users. After this phase, the resulting artifacts are drawing prototypes validated according to usability requirements and patterns. The description of the activities and artifacts in this phase is presented as follows.

Use Design Patterns

In order to guarantee the quality of the product and efficiency of the project, we suggest the use of design patterns for graphical UIs. These patterns will guarantee that we elaborate and develop UIs following already verified and validated parameters, which can be incremented by the reports generated in the end of the transition phase.

Consider Drawing Prototypes

UI designers are responsible for designing drawing prototypes based on paper sketches previously validated, and on usability patterns. These prototypes are verified and validated by the following two activities.

Verify with Expert

This activity consists of verifying the usability of drawing prototypes by experts. We propose that experts use the heuristic evaluation approach (Nielsen, 1993) as an inspection method to verify whether or not certain usability principles are present in the prototype.

When the expert notices that any usability principle is missing, change requests can be made, which leads to new proposals of prototypes. At this moment, it is not necessary to make changes in the models because this approach evaluates the quality of use, not functionality aspects, which were evaluated in the previous phase.

Validate with Users and Customers

After verification with the expert, a validation is proposed to users and customers so they can approve the generated prototypes. At this moment, users and customers evaluate the used usability patterns and the style guide. This process is repeated until the generated prototype attends users' preferences and constraints. If the prototypes do not attend users' needs, change requests can be made, which leads to new proposals of prototypes.

After the conclusion of the elaboration phase, the resulting artifacts are verified and validated drawing prototypes, which support development, tests, and deployment activities.

Phase III: Construction

In this phase, the UI is developed and the application is integrated with it. Considering software engineering, we propose functionality tests of an executable prototype (i.e., a product with some functionality) or the final product (i.e., a product with all the functionality), using functional test cases.

The activities and artifacts in this phase are presented as follows.

Implement and Integrate

These activities consist of developing the UI and integrating it with the application. The integrated product, either a prototype or the final system, can be useful for evaluating the navigation, interactivity, and functionality aspects.

Elaborate Test Cases

Test cases can be elaborated starting in the inception phase, using paper sketches, they can then be updated in the elaboration phase, using drawing prototypes, and finished in this activity. This artifact focuses on the system functionality, not on nonfunctional aspects.

The technique used to define test cases includes the following topics: association to a use case, specification of the item to be tested, preconditions to execute before testing, identification of valid and invalid inputs, and the expected outputs. The actual outputs are compared with the expected outputs described in the test cases and this comparison is used as validation of the use case.

Verify with Expert

This activity consists of the verification of the functionality of the product by usability experts and developers. Examples of aspects that are verified are: consistency of the outputs, navigation, existence of error messages, results after clicking on objects, as well as other aspects identified in the test cases.

After this verification, developers and experts can generate change requests to correct the errors; which leads to the repetition of the implementation and integration activities.

After the conclusion of the construction phase, the resulting artifact is the integrated product, which is tested with consideration to usability aspects in the transition phase.

Phase IV: Transition

This phase comprehends, in general terms, the preparation of the test environment, which can be a test laboratory or the environment where the system is used. With consideration to semiotic engineering, we use some validation techniques and artifacts. The description of the activities and artifacts in this phase is presented as follows.

Prepare the Environment

To prepare the environment, we suggest the installation of the system, software for capturing the system's use, and equipment, such as video cameras and necessary hardware devices. We also suggest the creation of questionnaires and checklists.

The test laboratory must be similar to the real user environment — with consideration to physical structure, climate, sound aspects, and equipment — in order to allow users to live the same conditions of the real environment. There should be a room where the test takes place and another one for observation.

Install the Product

In this activity, the product is installed, either a partial version or the final version of the system, in order to allow users to use the system in their real environment. This installation allows the tests to be performed.

Usability Tests "with Users"

This evaluation is performed with users. In it, evaluation techniques, proposed by semiotic engineering, are used, such as: recording, observation, questionnaires, and so on.

Before starting the tests, the usability expert talks with the user in order to: clarify that the system is under evaluation, not him/her; present the scenario used for the test; and make the user feel comfortable; which are aspects that influence the final results.

Observers that are in the observation room should fill out the questionnaires and checklists.

This activity can be divided in two moments, the first one, when the user interacts with the system to perform a task of his/her own interest; the second one, when the expert requires the user to perform a specific task.

In this activity, navigability, interactivity, and acceptability will be evaluated.

Use Observations/Checklist/ Questionnaires

Observations, questionnaires, and checklists are artifacts and techniques proposed by the semiotic engineering in order to verify the user-system interactivity and communicability. Experts and observers will use these artifacts during the tests, which result in the definition of the quality of the interactive system. These results can lead to change requests for developers to correct the detected mistakes.

Users' comments and the actual execution of the tests will be recorded to help in the analysis of the results of the questionnaires and of users' observations.

Make Corrections

In this activity, developers make corrections proposed by experts after the tests. After the changes are made, users validate the product.

Deliver the Evaluated Product

As a result of the process, we have the results of evaluations, which are useful for future versions; and we also have a verified and evaluated product according to a set of techniques proposed by usability, software, and semiotic engineering.

If the product is a final version of the system, it is ready to be delivered for use. If it is a partial version (e.g., executable prototype), the professionals need to perform the activities in the construction phase, then in the transition phase, until the product reaches its final version.

Elaborate Reports

The results obtained will be used as a basis for the elaboration of evaluation reports, which propose adaptations in the used patterns and in the creation of new patterns that can be used in future iterations.

CASE STUDY

In this chapter, we describe the case study of this research work, which is concerned with the evaluation of UIs for the SBTVD project, focusing on the applications: electronic portal, insertion of texts, and help.

Introduction

The digital TV represents digital and social inclusion for a great part of the Brazilian population, especially for people less privileged, who do not have access to computers, and therefore, cannot access the Internet.

The SBTVD must be adapted to the socioeconomic conditions of the country, as well as allow the use of conventional TV sets already in large use in the country in order to decrease risks and costs for the society.

The digital TV creates various possibilities of interaction between the TV and the user, such as: exchange of text or voice messages, virtual chats, searches for a favorite show, access to information about the government, and so on. These possibilities are different from the characteristics of the conventional TV, in which the user plays a passive role.

In the following section, we describe the performance of the activities proposed by UPi and UPi-Test. It is important to point out that the SBTVD is still under development. That is the reason why we cannot demonstrate all the activities of the process.

Phase I: Inception

Talk with Users and/or Customers

This activity was difficult to perform in this project because there is an almost unlimited number of users and/or customers. Fortunately, there were specific and clear specifications, established by the Brazilian government. Such specifications were used as requests from users and customers.

To define the scope of the application under our responsibility (access portal), we had meetings with representatives of the government and with other institutions that are participating in the project. These meetings were supported with brainstorming and the resulting decisions were analyzed and were shared with all the institutions through e-mails and discussion lists.

After these meetings, we decided that the portal will consist of a main application that allows the access to all other applications in the SBTVD, which can be: electronic mail, electronic commerce, EPG, help, electronic government, and so on.

Obtain Users' and Customers' Preferences and Constraints

The SBTVD project is concerned with various users' profiles, including the ones who are and those who are not used to technology, but not including the ones with any kind of disabilities.

In order to identify their possible preferences and constraints, we studied existing systems; we had many meetings and workshops. The opinions of all the participants in the project were taken into consideration because we can also be considered to be potential users.

Our main goal in the usability workshops was to choose the best usability pattern for each requirement based on the evaluation of positive and negative aspects of each proposed usability pattern, as specified in the UPi activity define UI plan.

These proposed usability patterns were selected from a list, such as the one available in Welie (2005), which are organized in the following format: problem to be solved, solution, context in which it can be used, and graphical illustration.

For the participants to evaluate the usability patterns, we provided a set of guidelines. After they read the guidelines, each group evaluated the positive and negative aspects of each usability pattern suggested for the requirement. Then, the participants discussed and reached the final decision as to what was the best usability pattern for the requirement under discussion.

The personalization group considered the guidelines as they evaluated the positive and negative implications of each usability pattern, aiming at achieving Nielsen's usability goals. For demonstration purposes, we focus on presenting the personalization of colors.

As a result of the evaluation, the personalization group decided to use color templates because of the greater impact of positive implications over the negative ones. The result of this work was the generation of a document associating usability patterns with their positive and negative implications. For instance, the selected pattern with more positive implications was "Provide predefined color templates to change font/background colors" instead of the pattern "Offer a list of colors from where the font/background colors can be chosen" because of the following positive implications: efficiency of use, compatibility of the system with the real world, and clarity of information. The second option had more negative implications than the first one, such as difficulty in use by beginners and the constant need to resize the space reserved to the presentation of information.

We defined that the applications in the SBTVD need to have a high level of usability. There was a list of technology constraints, especially the ones related to memory and processing capacity. Besides that, graphical representations need to be simple because conventional TV sets do not support images as computer monitors do, and the TV luminosity is very different from the one in monitors; consequently, colors and texts appear differently.

Consider HCI Models and Use Cases

In this activity, analysts and usability experts elaborated use case and task models, and then changed them, when change requests were made after the evaluation of users in the end of the Inception phase. The requests resulted in changes in the models because they were changes in the functionality, such as: do not consider access to multiple applications simultaneously (technical constraint) and include personalized help. These changes reflected in the models and will be considered in the prototypes.

Analyze Existing Systems

In this activity, we researched on the Internet in order to find digital TV systems already in use. It was very difficult to find them because there are few systems available for access in the Internet.

This analysis was used to identify some UI design patterns for the digital TV, such as: upper and bottom options bar, menu on the left, navigation icons, interaction buttons, and so on.

Consider Paper Sketches

After the analysis of existing systems, UI designers elaborated paper sketches (Figure 2). However, they needed to be redesigned after the evaluation of users in the end of the inception phase, which resulted in change requests.

Figure 2. Paper sketch: Personalization

Some change requests resulted in the following changes in the prototypes: transfer the bar from the top to the bottom of the screen, include the "TV" button in the bar when accessing an application or in the portal; include the "Portal" button in the bar when accessing an application, take off activated icons (because of technical restrictions), and give a preview of the requested options of personalization. However, other requests resulted in changes in the models as well as in the prototypes, such as the ones mentioned in the activity "Consider HCI models and use cases."

Verify with Expert

In this activity, we performed heuristic evaluations with three experts who considered the usability of the paper sketches. They observed aspects, such as: navigation between views and if usability principles were applied. Besides that, the experts verified if all users' goals, in terms of functionality (in the task models), were included in the prototypes.

Validate with Users and Customers

In meetings with the institutions participants of the project and with representatives of the Government users evaluated the elaborated prototypes.

Various change requests, as well as users' preferences were identified. These requests resulted in updates in the models and in redesign of the paper sketches.

As examples of preferences, we point out the following: Include the "TV" and "Portal" in the remote control; the bottom options bar should overpass the application; and make available a set of options for inserting text, such as the one used in computer keyboards, the one used in mobile phones, and in alphabetical order.

Phase II: Elaboration

Use Design Patterns

In this activity, the UI design patterns, identified while analyzing existing systems, were evaluated and the UI designers started to design drawing prototypes using the design patterns.

Consider Drawing Prototypes

The UI designers used image editors, such as Photoshopâ and Corel Drawâ, in order to design the drawing prototypes (Figure 3), which followed the selected patterns, the updated task models, and paper sketches. In this prototype, the small view on the right shows the preview of the required personalization about the color of the screen before applying it.

Verify with Expert

The drawing prototypes were evaluated by experts (Figure 4), who observed various aspects, such as: layout, colors and fonts, UI design patterns, and usability principles applied.

Validate with Users and Customers

This activity is yet to be fully performed in this project, currently; we have finished the document that specifies the SBTVD graphical UI, which is going to be evaluated by representatives of the Brazilian Government.

Figure 3. Drawing prototype: Personalization

Figure 4. Verification with expert

Figure 5. Validation with users

We scheduled a workshop with the other institutions participants of the project to evaluate the drawing prototypes (Figure 5). They requested us to elaborate other alternatives of UIs (including association of options in the UI with options on the remote control), increase the size of the options and fonts in the upper bar, and change the way to differentiate the selected option.

Phase III: Construction

Implement and Integrate

The implementation of the portal has not started yet. However, the developers have been studying and implementing simple applications for the digital TV, such as three possibilities of insertion of text (the one used in computer keyboards ('qwert'), the one used in mobile phones, and in alphabetical order).

Table 1. Test case: Personalization

Test Case	Personalization
Test Items	Changes in font and background color Changes in font size
Pre-conditions	The user must be accessing the portal
Inputs	The user must select a type of personalization: Font color The user selects green
Expected Result	The portal must present the personalized content with a green background color

Elaborate Test Cases

The elaboration of functional test cases has started since the inception phase, when use cases and paper sketches were elaborated. Functional requirements were selected and the associated test cases were elaborated: structure applications, access applications, access help and personalization (Table 1).

Verify with Expert

In this activity, the programmers and the experts verify the applications using the test cases for guidance. Some of these evaluations were done in the small applications developed for insertion of text.

Phase IV: Transition

The usability experts and the developers have prepared the test environment. For that, they have: prepared the physical structure (e.g., TV, video camera, couch, computer to simulate the set-top box, etc.) and installed the necessary software (i.e., software to capture the user interaction with the TV, Linux Operating System, Java Virtual Machine, and applications to be tested).

A group of three usability experts were responsible for performing the following activities before the tests started: First, they defined a questionnaire to apply with users in order to understand their characteristics and familiarity to the DTV technology. Second, they selected appropriate metrics (e.g., number of errors, number of access to the help, etc). Third, they created a checklist based on Nielsen's usability goals and on the metrics from the previous step. Fourth, they prepared the environment with a DTV, a couch, and a center table in order to make users feel at home. Fifth, they selected ten users with different profiles between the ages of 18 and 26.

When the users arrived, each one at a time was taken to the DTV room, where a usability expert explained the goals of the test; applied the questionnaire; and defined a specific goal to be achieved while interacting with the DTV. While the user interacted with the application, usability experts filled out the checklist in the visualization room, where we monitored the environment and the user with a camera and captured the interaction with the DTV using specific software.

FINDINGS AND FUTURE WORKS

After the tests, we evaluated the checklists and generated reports with solutions to the problems encountered during the tests. These reports contain comments about the icons, the navigation, and the help module of the portal application.

- No user realized the possibility of navigation through the numbers associated to each icon, which represents an interactive application (see Figure 6).
- When they were told about this, they used it and said it was very practical.
- For them, the icons were not a natural representation, but they did not want to give any opinion about possible solutions.
- Some users did not notice the application that was selected. They said it was better to have a big square around the icon instead of the current selection.
- When the users were confused and wanted some help, they did not look for it in the portal (the blue option in the bottom bar that activates the help module). Some users activated the menu button of the remote control and others waited for the evaluator to tell them what was necessary to do to obtain the help.
- As the bottom bar is used as a design pattern to put the navigation options, and as each option is different depending on the situation (the application being executed), the users got very confused. We realized they looked

Figure 6. The portal application

at this bar only once, and then they did not look at it any more. After memorizing the options, they wanted to use them all the time, but the color was different, so they made many navigation errors.

From the execution of these tests, we were able to quantify the results: 85% of the users found the system easy to use, easy to read, and easy to navigate, but on the other hand, 50% had difficulties in identifying that an option was selected and in getting out of an unexpected situation.

Our next step is to make the necessary changes that are related to improving the layout, color contrast, and icon selection. In addition, the next tests will be done with the elderly in order to investigate people who are not used to interaction devices.

CONCLUSION

In this chapter, we present a new method, which focuses on integrating different techniques of usability, semiotic and software engineering. The aims are to design usable UIs following a model-based UI design process and to facilitate the test process by using the evaluated HCI models. In this manner, we hope to contribute to the development of interactive systems that are easy for users to learn and use, and to help testers in performing their usability tests in an efficient manner.

As main contributions, we focus on evaluating the usability of UIs with the constant participation of users and customers. Besides that, the integration of various approaches results in positive outcomes for the prototypes, as well as for multidisciplinary team members, who are better integrated and can have their knowledge

enhanced, since they are continuously exchanging information and experiences.

REFERENCES

Baranauskas, M. C. C., & Rocha, H. V. da. (2003). Design e Avaliação de Interfaces Humano-Computador, NIED–Núcleo de Informática Aplicada à Educação, UNICAMP–Universidade Estadual de Campinas.

Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., & Vanderdonckt, J. (2004). SketchiXML: Towards a multi-agent design tool for sketching user interfaces based on UsiXML. In P. Palanque, P. Slavik, & M. Winckler (Eds.), *3rd Int. Workshop on Task Models and Diagrams for User Interface Design* (pp. 75-82). New York: ACM Press.

Gould, J. D., & Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM, 28*(3), 300-311.

ISO 13407. (1999). *Human-centred design processes for interactive system teams.*

Kruchten, P., Ahlqvist, S., & Bylund, S. (2001). User interface design in the rational unified process. In M. Van Harmelen (Ed.), *Object modeling and user interface design* (pp. 45-56). New York: Addison-Wesley.

Lewis, C. (1982). *Using the "thinking-aloud" method in cognitive interface design* (IBM Research Rep. No. RC9265, #40713). Yorktown Heights, NY: IBM Thomas J. Watson Research Center.

Myers, G. J. (2004). *The art of software testing.* New York: John Wiley & Sons.

Nielsen, J. (1993). *Usability engineering.* Boston: Academic Press.

Pressman, R. S. (1995). *Engenharia de Software*. São Paulo: Makron Books.

Schilling, A., Madeira, K., Donegan, P., Sousa, K., Furtado, E., & Furtado, V. (2005). An integrated method for designing user interfaces based on tests. In *Proceedings of the ICSE 2005 Workshop on Advances in Model-Based Software Testing,* (pp. 27-31). St. Louis, MO: ACM.

Sommerville, I. (2001). *Software engineering.* New York: Addison-Wesley.

Souza, C. S. de, Barbosa, S. D. J., & Silva, S. R. P da. (2001). Semiotic engineering principles for evaluating end-users programming environments. *Interacting with Computers*, *13*(4), 467-495.

Sousa, K. S., & Furtado, E. (2004). UPi — A unified process for designing multiple UIs. In *IEEE Computer Society, Proceedings of the International Conference on Software Engineering,* Edinburgh, Scotland (pp. 46-53). Edinburgh, Scotland: ACM.

Welie, M. van (2005). Retrieved September, 15, 2005, from http://www.welie.com

This work was previously published in Verification, Validation and Testing in Software Engineering, edited by A. Dasso, pp. 55-81, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 5.30
The Work of Art in the Age of Mechanical Production

Thomas B. Cavanaugh
Embry-Riddle Aeronautical University, USA

ABSTRACT

When Walter Benjamin wrote his famous essay *The Work of Art in the Age of Mechanical Reproduction*, he shone a light on the cultural changes inherent in technology's ability to infinitely reproduce and distribute art. One of the important consequences of this development was the democratization of art's availability, allowing the general population to experience artwork that they would otherwise be unable to access. Now technology has advanced to a point where not only is art's reproduction available to anyone who wants it, its very production is now accessible to almost everyone, even if the prospective artist is utterly devoid of training, expertise, or even talent. With software-based artistic assistance and low-threshold electronic distribution mechanisms, we have achieved the promise of Benjamin's blurred distinction between artist and audience. As a result, the process by which art is produced has now been democratized, resulting in legitimate questions regarding quality, taste, and the legitimacy of authorship in a human-technological artistic collaboration.

INTRODUCTION

When Walter Benjamin wrote his famous essay *The Work of Art in the Age of Mechanical Reproduction*, he shone a light on the cultural changes inherent in technology's ability to infinitely reproduce and distribute art. One of the important consequences of this development is the democratization of art's availability, allowing the general population to experience artwork that they would otherwise be unable to access—artwork that previously was only available to a privileged elite.

Now technology has advanced to a point where not only is art's reproduction available to anyone who wants it, the very production of art is now accessible to almost everyone, even if the prospective artist is utterly devoid of training, expertise, or even talent. This development is symptomatic of a wider self-service mindset that pervades western society, what I have dubbed the "kiosk culture." The term "kiosk culture" connotes the general acceptance and even encourage-

ment of self-service for the postmodern citizen. From touchtone telephone answering systems to Web-based electronic commerce to scanning our own groceries at the neighborhood supermarket, self-service is as much an expectation for the postmodern consumer as it is a business necessity for the provider. The kiosk culture represents nothing less than a cultural transition to a naturalized "do it yourself" attitude.

In the production of art, the do-it-yourself aesthetic of the kiosk culture has been actualized by the introduction of performance support technology into all manner of expression. Just as with consumer-oriented tasks, artistic performance support technology compensates for any skill-based shortcomings, allowing a novice to achieve the same level of performativity as an expert. As such, the very process by which art is produced has been democratized. But when these support technologies are applied to the creation of art, the cultural implications are quite different from those in the business milieu. It raises essential questions such as *What is good art*? *Is art developed with the assistance of performance support technology as valuable as that produced without it? Who really is the artist—the human or the machine?*

Art, of course, does not exist independently from a complicated infrastructure of production, distribution, and consumption, tied together by social relationships that maintain it. As described by Bourdieu (1993) and others, the consumption of art involves social, economic, and class implications. Recognizing this larger context, it's important to note that the subject of this essay focuses only on the production of art—specifically the production of art as enabled by the use of assistive performance support technology.

Software such as Dramatica and Scriptware assists the aspiring screenwriter to prepare a screenplay. In many cases, this support goes far beyond surface assistance such as formatting and spelling to include brainstorming services, story ideas, and other cognitive help normally considered the sole domain of the artist. Marketing literature for Dramatica® Pro touts the collaborative aspect of the software's artistic support, calling it "the 'Ultimate Creative Writing Partner.'" Another software product from the same company, Writer's DreamKit™, is promoted specifically as a tool for beginning screenwriters and even promises to co-write your script. "(A)s your mentor, the Writer's DreamKit will do something no other writing program can do—it predicts parts of your story based upon creative decisions you make!" (Write Brothers, 2004, p. 5-6)

With technological tools like these and low-threshold distribution mechanisms such as self-publishing, podcasting, and blogs, we have achieved the promise of Benjamin's blurred distinction between artist and audience. Benjamin (1968) predicted that the distinction between author and public is about to lose its basic character:

The difference becomes merely functional; it may vary from case to case. At any moment the reader is ready to turn into a writer. As expert, which he had to become willy-nilly in an extremely specialized work process, even if only in some minor respect, the reader gains access to authorship… Literary license is now founded on polytechnic rather than specialized training and thus becomes common property. (p. 232)

Just as we no longer need to be a trained cashier in order to use the complex point-of-sale (POS) technology of the retail industry, likewise we no longer need to possess any expertise in how to write a screenplay to accomplish the writing task. The embedded performance support technology compensates for our lack of competence. The "readerly" has literally become "writerly," where "the reader [is] no longer a consumer but a producer of the text" (Barthes, 1970, p. 4).

NOVICES AND EXPERTS

But what about other artistic endeavors, especially those that require the direct use of complex technology? The most obvious example might arguably be photography. Referring specifically to digitization, Willis claims that the "means of production of ... visual imagery is undergoing a mutation as significant as the invention of photography" itself (1990, p. 197). Unlike painting, for instance, where the artistic technology (e.g., the brush) doesn't easily permit compensatory performance support, photographic equipment utilizes a variety of compensatory performance support technologies, such as red-eye reducing flash bulbs, auto-focus, backlight adjustments, and digital enhancement tools. Even before the introduction of this technological assistance, photography had already secured a place as an equalizer between the amateur and professional.

From the middle of the nineteenth century, "with its low cost and availability, photography democratized the visual image…For the first time, it was possible for the ordinary person to record his or her life with certainty and to create personal archives for future generations" (Mirzoeff, 1999, p. 65). Although the technological elite of the time tried to categorize photography as a pursuit solely for society's upper classes, the low cost of materials and relative usability of the technology trumped any attempts to restrict its growth. "With the invention of the collodion-glass negative process in 1852, prints became affordable to all…working people. They availed themselves of this resource en masse to the despair of elite critics who disparaged the results as 'fried fish pasted onto metal plaques'" (Mirzoeff, 1999, p. 72). But the genie was already long out of the bottle. "Despite the attempts by [*photographic pioneers Louis Jacques Mande*] Daguerre and Fox Talbot to claim elite status for their devices, photography was quickly claimed as the people's medium" (Mirzoeff, 1999, p. 71). With the addition of compensatory performance support technology to photographic equipment, this even more true today.

One might argue that any literate and reasonably coordinated person can string two words together or point a camera and press a button and, as such, literary authorship or photographic competence are easier for an amateur to achieve than an artistic pursuit more dependent upon manual dexterity. For instance, what about music? At one time in our past, playing music was a common amateur activity and musical training was a requisite component of the bourgeois education. But with the advent of mechanized music in the late nineteenth century, such as that offered by the gramophone, homemade music rapidly declined. This phenomenon manifested in what Thompson (2002) calls "a wide chasm" between professionals and amateurs. An early attempt at musical performance support to reduce that chasm was the simplified sheet music produced by publishers for home use:

> *As amateurs gradually abandoned their own music making and listened increasingly to professional musicians, a wide chasm opened between the two groups… Sheet music publishers did their best to bridge the gap, by offering 'Brilliant but not Difficult' versions of the most popular showpieces, but the effect of the discrepancy was gradually but effectively to silence many amateur performers of music. By the end of the century, countless parlor pianos had been replaced by automatic 'reproducing' pianos or other mechanical devices that recreated the performances of great concert pianists. …The result of these trends was a new dissatisfaction with amateur music and, perhaps more significantly, a heightened engagement by amateurs with the experience of listening to professionals.* (pp. 49-50)

Thompson quotes an 1894 article from the *Atlantic* magazine which describes the term

"amateur" pejoratively, equating amateur efforts with "bad work" (p. 49-50).

Benjamin's colleague Adorno would likely agree with the *Atlantic*'s assessment, excoriating any attempts to produce "Brilliant but not Difficult" sheet music of popular tunes for the untrained masses. He refers to such products designed to bridge the gap between amateur and professional as a:

sort of musical children's language…it differs from the real thing in that its vocabulary exists exclusively of fragments and distortions of the artistic language of music. In the piano score of hit songs, there are strange diagrams. They relate to guitar, ukulele and banjo, as well as the accordion—infantile instruments in comparison with the piano—and are intended for players who cannot read the notes. They depict graphically the fingering for the chords of the plucking instruments. The rationally comprehensible notes are replaced by visual directives, to some extent by musical traffic signals. These signs, of course, confine themselves to the three tonic major chords and exclude any meaningful harmonic progression. The regulated music traffic is worthy of them. (1982, pp. 290-291)

Adorno would agree that playing a musical instrument is difficult (clearly privileging the piano) and rightfully so, or it might have no more value than any other common activity, as unique as walking across the room. Mastering a difficult activity such as playing a musical instrument is what makes the artist admired. Even that great contemporary prophet of technological usability, Donald Norman, accepts the usability challenges of musical instruments, stating that they "take years of dedicated practice to be used properly, and even then, errors and poor performance are common among nonprofessionals. The relative unusability of musical instruments is accepted, in part because we know of no other alternative, in part because the results are so worthwhile" (2004, p. 77-78).

But is this really the case? With today's technology, do instruments really *need* to be so difficult in order to retain their value? Just as you might be able to envision the design of an airplane with embedded transparent performance support that enables an untrained novice to pilot it successfully, why can't you picture a musical instrument equipped with similar performance support? What if a piano's keys could be internally illuminated in time to a musical score, allowing a novice to perform within an acceptable margin of error even if he or she had never sat on a piano bench before? What if a saxophone's interface (its keys) could be equipped with both visual support (such as lights or shades of color) and haptic feedback (such as localized vibrations or mild electrical sensations)? What if other, more sophisticated and transparent performance support technology were embedded within musical instruments? Instruments are, after all, technology. By definition, performance support technology is technology that helps humans use other technology. When a novice user is unable to accomplish the artistic/musical performance, he/she activates the support (directly or indirectly) and the resulting performance is one far more polished than the novice could otherwise achieve. A popular example is the video game Guitar Hero, which allows a novice musician to play music on a simulated guitar with embedded support and performance feedback.

If, as Thompson suggests, the widening gap between musical novices and professionals is a symptom of modernity—something surely celebrated by a high modernist such as Adorno—then, perhaps the obliteration of that gap by transparent performance support technology is indicative of a postmodern evolution. In this context, the only relevant criterion is the musical performance. How that performance is achieved is irrelevant. This is no different than the goal of performance support in an industrial environment. When viewed via this extreme postmodern perspective, terms such as "novice" and "expert" become meaningless.

There are no novices or experts. There is only *in-situ* "performance."

Taylor (2001) addresses this transition directly, connecting it to the "modern" chasm between amateurs and professionals, but recognizing what may be a postmodern shift. "Before the advent of recording technology and radio, people made their own music most of the time, but what is radically different today is that it is now possible to create entire worlds of sound all by yourself with your computer" (p. 139). The days are now gone when you need formal musical training in order to produce music. The *performance now resides in the situation.* "Digital technology…makes home music making possible as never before. One can create complex, polyphonic music at home with a computer and other digital equipment without having had years of piano lessons" (Taylor, 2001, p. 13). We have reached a musical point where Benjamin's reader (listener) has become writer (composer). And this is not limited solely to music. "The relatively low cost of new media…and the internet allows many more people to be both producers and consumers of visual and other media" (Mirzoeff, 1999, p. 256-257). The postmodern atomization of both artistic production and distribution have contributed to this new producer/consumer phenomenon, where at any given time, any given individual can be found on either side (or both) of the creation-consumption continuum.

THE QUESTION OF TASTE

Although performance may be the most important metric for success in both business and art, unlike many businesses, the art world cares very much about the human being responsible for the performance. With transparent performance support, the difference between the artist and the technological helper is hazy at best. Lanham (1993) describes the challenge in attributing artistic responsibility and assessing talent:

> *Programs available widely and cheaply…allow novices to compose pleasant sounding music by enlisting the computer as co-composer. Far from…thinking that the computer diminishes human originality and skill, the authors of such programs often regard the physical skills needed for performance, and the theoretical knowledge needed for notation, as elitist prejudices…What "musical talent" is thought to be may itself change…In Laurie Spiegel's Music Mouse, you move the Macintosh mouse around on its pad and the linear motions are translated by the computer into musical sounds. Time and space, drawing and music, are made one by digitization. And if the music sounds good, as often it does, what does "good" mean here? "Who" has created the goodness?* (p. 12)

Does it matter "who" is responsible, as long as the result—the performance—is the desired outcome? What, after all, is "taste?" As Lanham said, what is "good?" Is there any objective definition of good taste that can be applied across society? As noted by Freedman (2003), "even the term art has traditionally signified a judgment of quality" (p. 53).

This notion of quantifying aesthetic taste was an issue addressed by pioneering acoustical physicist Wallice Sabine as early as 1902. Sabine first became known for the development of a scientific acoustical formula that allowed him to predict how sound would behave in any interior environment, based upon a set of variables. After the initially favorable reviews of the musical acoustics of the newly constructed Boston Symphony Hall, on which Sabine consulted, dissenting critical voices began to complain about those very same acoustics. So Sabine convened a committee of faculty members from the New England Conservatory of Music and conducted a series of acoustical tests to determine the optimal amount of acoustical reverberation (Thompson, 2002, p. 55-56).

What Sabine attempted to do, in true scientific fashion, was determine an algorithm for taste.

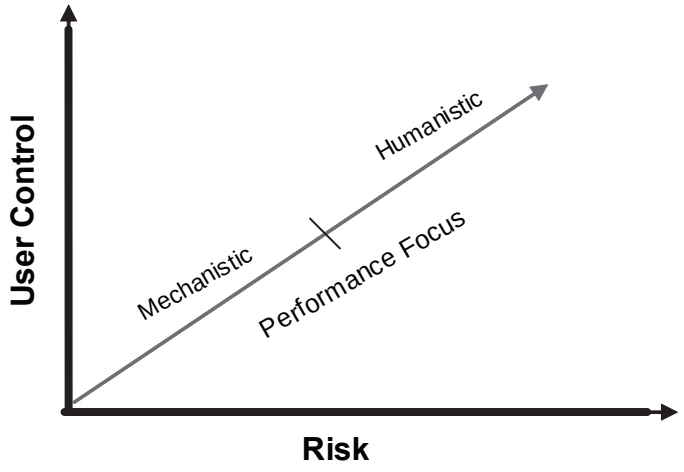
Whether he or his acoustical successors achieved this goal is debatable. Throughout the history of sound, the amount of ideal reverberation continued to be adjusted based upon the individual preferences of the audience and the fashion of the time. Sabine, of course, was trying to objectify the *external* quality of the sound. He wanted to answer the question: *what is good sound*? He did not address the *internal* quality of sound, specifically its content—*what sounds good?* This, naturally, begs the question: can an algorithm be developed that describes exactly what constitutes a "good" song? Can we derive a formula for taste?

In essence, that's exactly what screenwriting software, entry-level musical composition software, and photographic auto-focus attempt to do. In order for a novice to possibly write a screenplay or compose a song, the system must make some transparent "intuitive" performance support decisions about how that artistic expression will be created. Those decisions—the variety of choices offered, the amount of user control—constitute a *de facto* taste algorithm. By changing the values of the few variables in the formula we are permitted to adjust, we create an original piece of art within prescribed limitations.

The result, however, seems to be (to echo Barthes in *Image-Music-Text*) a general "flattening" of the qualitative differences between pieces (1977, p. 189). For example, with musical composition software, on the bottom end, the system prevents you from producing something so horrible as to be universally unrecognizable as music. However, by so doing, it also prevents you from really excelling at the top end. The very performance support decisions that enable a novice to compose a song actually preclude someone with real talent and experience from composing something truly wonderful. The system is only capable of producing music in a general "fair" range.

It seems that an equation could be derived that states that the user's ability to leverage his/her own talent and experience is directly proportional to the amount of control that a piece of software affords that user (Figure 1). That amount of

Figure 1. Control vs. risk with artistic performance support technology



control, however, is also proportional to the risk that a novice has of failure. The lower the risk of failure, the lower the ability to excel.

Greater User Control =	Greater Risk of Failure Increased Ability to Excel
Lesser User Control =	Lesser Risk of Failure Decreased Ability to Excel

This equation can very likely be applied to any example of artistic performance support technology, including the earlier examples of screenwriting software.

As the amount of user control lessens, the greater the level of mechanical influence. Conversely, as the amount of user control increases, so do the humanistic/organic characteristics. Control is an important issue relative to performance support across all areas of society—whether in business, the arts, medicine—since it is intrinsically linked to risk and success (financial, artistic, physiological). Norman (1988) addresses this issue directly when he writes:

All tasks have several layers of control. The lowest level is the details of the operation, the nimble finger work of sewing or playing the piano, the nimble mental work of arithmetic. Higher levels of control affect the overall task, the direction in which the work is going. Here we determine, supervise, and control the overall structure and goals. Automation can work at any level. Sometimes we really want to maintain control at the lower level. For some of us, it is the nimble execution of the finger or mind that matters. Some of us want to play music with skill. Or we like the feel of tools against wood. Or we enjoy wielding a paintbrush. In cases like these, we would not want automation to interfere. At other times we want to concentrate on higher level things. Perhaps our goal is to listen to music, and we find the radio more effective for us than the piano; perhaps our artistic skill can't get us as far as can a computer program. (p. 197)

The risk of creative failure is mitigated, of course, through the use of artistic formulas. An artistic formula, like a mathematical formula or algorithm, is simply a structure in which variables can be adjusted to produce a predicted result. However, unlike mathematics, where a formula is generally perceived as a useful tool, artistic formulas have traditionally been denigrated as clichés and the work of hacks. "The more stuffed the poem with 'formulas,' the more it passes for successful," wrote Barthes (1979, p. 112). Unfortunately, in the world of arts education, where the professional artists of tomorrow are being trained, "the value of originality is imparted at the same time as the value of following rules" (Freedman, 2003, p. 8). This mixed message—originality vs. following rules—is fundamental to the challenge faced when trying to balance issues of control. As formula-based performance support technology embeds itself more intrinsically into the tools of artistic production, questions will arise regarding the value of artistic education. If the performance is resident in the instrument (or camera, or word processor), then why do we need master artists or teachers? This is a dangerous question, of course, and one that will likely be rejected by those within the established artistic community.

Adorno doesn't spare the rod when addressing the formulaic structure of popular music. He laments the "standardization" of this music and foreshadows the very "flattening" evidenced by entry-level musical compositional software:

The harmonic cornerstones of each hit—the beginning and end of each part—must beat out the standard scheme. This scheme emphasizes the most primitive harmonic facts no matter what has harmonically intervened. Complications have no consequences. This inexorable device guarantees that regardless of what aberrations occur, the hit will lead back to the same familiar experience, and nothing fundamentally novel will be introduced. (1941, p. 17)

Of course, what constitutes a formula may only be in the eye of the beholder. Just because Adorno or the musical elite might denigrate the music of a currently popular but critically panned artist such as Ashlee Simpson doesn't mean that her music has no value. She brings pleasure and enjoyment to her fans, who would likely roll their eyes if Adorno suddenly appeared and told them they were being duped by the great manipulative apparatus of corporate music. The critical lambasting of Ashlee Simpson and her music is consistent with Mirzoeff's observation of mass culture (specifically visual culture) that "the generalized antipathy of intellectuals to popular visual representations may be displaced hostility to those who participate in and enjoy mass culture" (11). But are the critics even relevant? The Beatles and Elvis were panned at first, too. Besides, is there anything wrong with someone listening to what they like and actually enjoying it? Does everyone have to enjoy the same music?

This is likely the primary reason why achieving a "taste" algorithm is so difficult. If you prefer Ashlee Simpson and I prefer Beethoven, then how could our tastes possibly agree in order to be captured in any type of meaningful equation? The answer may lie in those very differences. What if you could choose the type of music you liked best and then the embedded performance support technology helped you create music in that style? Inventor Ray Kurzweil has written at length about this issue and his "Poet's Assistant" software is an excellent example of this concept:

The Poet's Assistant provides a simple word processor ("poem processor") in which you write your poem or song lyrics. Besides providing standard text editing features, the Poet's Assistant monitors your "poem-in-progress," and provides a wide range of helpful suggestions as you type…Each suggestion is based on a "poet personality" that you select…The standard version…includes 50 poet personalities…For example, the Poet's Assistant can provide rhymes (words that rhyme with one of your words) or half-rhymes that were used by Robert Frost; alliterations used by Yeats; a set of "next words" used by Shelley; and entire lines (or stanzas) of poetry that are based on (but which were not actually written by) Emily Dickinson (or based on a combination of contemporary poets). From these lines and/or stanzas, you can find ideas for words and turns of phrase. (2005a)

Kurzweil also offers the next step in mechanistic art, what he terms a completely "cybernetic poet" in which a human being isn't even necessary. Called the Ray Kurzweil Cybernetic Poet (RKCP), the system auto-generates poems using an algorithm based upon pre-selected poet styles. This "unique recursive poetry generation algorithm" is rooted in an analysis of the selected poet's technique (2005b). When tested, it was difficult for a combined general audience of juvenile and adult readers to determine which poems were written by the actual poet and which were written by the computer (Kurzweil, 1992).

However, Kurzweil's cybernetic poet strays a bit far from the domain of performance support technology, which, by definition, requires a human to be supported. The cybernetic poet is more indicative of automation (a term invoked by Norman in the earlier quote regarding issues of control). But I include it here to illustrate recent attempts to develop taste algorithms. Are they successful? To many, especially those who couldn't tell the cybernetic poetry from that of Eliot or Frost, the answer is certainly "yes"; although, to others, the jury is still decidedly out.

When Kurzweil discusses "language modeling techniques" and a "recursive poetry generation algorithm," we can't help but speculate on a post-human stylistic recursion. Will we reach a point where the very poetry generated by the computer is being used as a stylistic model for future cybernetic poetry, in a sort of technological Ouroboros, a great mechanical serpent continually swallowing its own tail, reducing humanity to an uninvolved bystander? As Kahn (1999) says,

"The relationship of techniques to technologies is…complicated since it is clear that technologies can derive from techniques…and techniques can derive from technologies" (p. 15). Musicians today, especially those working in sampled sounds, are creating music that can only be produced by very specific technology. The technology is molding the technique, which, in turn, continues to influence the technology of music production.

Yet, if performance support technology employs formulas to assist in the creation of art, even if those formulas are based on T.S. Eliot or J.S. Bach, perhaps all we're left with is a computerized Minou Drouet (to reference Barthes), where the formulaic banality of the work is masked by the novelty of who (or what) produced it. It's possible that, as Adorno stated, "the golden age of taste has dawned at the very moment in which taste no longer exists" (1982, 280-281). Adorno, of course, implored us to resist what he called "the cult of the machine…For the machine is an end in itself only under given social conditions—where men are appendages of the machines on which they work. The adaptation to machine music necessarily implies a renunciation of one's own human feelings and at the same time a fetishism of the machine such that its instrumental character becomes obscured thereby" (1941 p. 17-48). To many, the simple fact that a particular piece of music was created by a computer is what makes it interesting, regardless of any artistic merit. Such listeners have fetishized the machine to the detriment of the art.

"A TRIUMPH OF AMATEURISM"

However, some artists today do resist the overmechanization of the creation process. This goes back to the essential issue of control. The more experienced and/or talented the artist, the less he/she wants to abdicate control of the creative process to a machine. Talent is a rare commodity and those who have it are unlikely to dull its edge with assistive technology. Trained artists with legitimate talent may naturally resist the unrestricted democratization of artistic production. Perhaps extending the kiosk culture into the arts eventually leads to an overall cheapening of the value of artistic talent. How many of the scripts generated by the screenwriting software would be movies you would want to see? How much of the computer-mediated music is worth listening to? Will we lose something intrinsically—exclusively—human by layering in compensatory artistic performance support technology?

The answer to this question may very well be different for each person depending upon your own perception of personal artistic talent. Hayes (1991) feels that such human-computer collaborations could result in "a triumph of amateurism. Computer-based tools may compensate for the amateur's lack of skill, but they cannot make up for a failure of taste or judgment" (p. 15). Do we end up with, as the critics of early amateur photography protested, "fried fish pasted onto metal plaques" (Mirzoeff, 1999, p. 72)? Barthes astutely observed that the difficulty in mastering Beethoven is related more to the artistry of the music—the code—than in any technical skill required to perform it (1977, p. 152).

Benjamin himself remarks on the rarity of talent in a long note at the end of *The Work of Art in the Age of Mechanical Reproduction*. He quotes at length from Huxley (1949), who declares that "artistic talent is a very rare phenomenon," in both the visual and auditory arts. With the increase in the sheer volume of artistic material produced, the result is that:

the gramophone and the radio have created an audience of hearers who consume an amount of hearing-matter that has increased out of all proportion to the increase of population and the consequent natural increase of talented musicians. It follows from all this that in all the arts the output of trash is both absolutely and relatively greater than it was in the past; and that it must remain

greater for just so long as the world continues to consume the present inordinate quantities of reading-matter, seeing-matter, and hearing-matter. (pp. 217-251)

The production of art of dubious merit is only exacerbated by the rise of performance support technology. A quick internet search will reveal literally millions of homemade songs, podcasts, and other creative expressions available for the general public.

Nevertheless, for the novice or untalented, the machine offers access to previously unavailable artistic expression. Perhaps self-service art is the ultimate extreme in egalitarianism, opening the closed, rarified world of art to anyone who wants to join in. Any layman who wants to write or paint or compose or play an instrument can do so successfully. If that results in more people enjoying the creation of and participation in art, is that so bad?

Where technology once created distance between amateurs and professionals (Thompson's "wide chasm"), technology is now pulling both sides back together into a homogenous group where at any given moment someone can be either creator or audience (or both). "With digital technology, there is some hope that people—at least those who can afford computers—will begin to make music for themselves again using their computers and cheap, easily available software" (Taylor, 2001, p. 5). Taylor quotes from a publication aimed at amateur composers of electronica music: "'Forget about talent,' says 'How to Become a Techno God,' a guide to making your own techno music; 'talent just gets in the way'" (p. 164).

Whether in music or the visual arts, this attitude is reflective of "a young generation that seems increasingly resistant to the seductions of production values and actively engaged in a 'do-it-yourself' aesthetic…the production of visual imagery is by now so diversified that it collectively signifys [sic] a shift away from the passive consumer trying to invest consumption with meanings towards a consumer-producer" (Mirzoeff, 1999, p. 257). Recall, too, Lanham's remark that the programmers of composition software regard the physical skills of musicianship as "elitist prejudices." Who needs talent when it's already programmed into the software?

Although he would probably cringe at artistic performance support technology, even Adorno might smirk at some of its cultural consequences. As more and more art is produced by people who were previously unable to do so, traditional distribution channels are being torn asunder. There is simply too much content now. As a result, the seams of the traditional pipelines are popping open under the pressure of volume and artistic expression is finding new, unorthodox ways to reach the public.

USA Today recently observed that "new sounds and new bands are emerging from Web cabals, a blow to radio and labels used to setting the agenda… That's causing a democratic shift, as more phenomena filter up and fans elect their own stars rather than accept the dictates of radio or MTV" (Gunderson, 2005). The vast corporate musical apparatus is losing the power to dictate what kinds of popular music the mass consumer should listen to. Media can now be "distributed both locally and globally by activist groups, charities and other small-scale organizations" (Mirzoeff, 1999, p. 257). This sentiment is resonating across the media spectrum with the emergence of outlets such as the Current television network, which is founded on the principle that its programming will be produced largely, if not entirely, by amateurs. Such direct distribution channels, with podcasting being the most acute example, are undermining the role of Bourdieu's mediators (1984), which until now have been serving as "cultural intermediaries," controlling the presentation of commoditized artistic goods and services.

As a catalyst for this shift, enabling amateurs to produce art and other media, performance support technology helps to answer some of Adorno's

Marxist critiques of the music industry while at the same time validating Benjamin's vision of artistic democracy. If, as Freedman (2003) has written, "all art is political" (p. 52), then the politics of artistic performance support and postmodern technological distribution channels are certainly "democratic." The kiosk culture is "of the people, by the people, and for the people."

Perhaps the role of artistic performance support technology isn't to fundamentally change the process of creating art. It seems unlikely that it will turn talentless hacks into artist poseurs. By and large, especially over time, audiences can tell the difference. Audiences grow increasingly sophisticated and, despite Adorno's assertions, Mirzoeff (1999) appears to be on the right track when he declares that "it is no longer possible to suggest that the mass audience will gullibly consume any product that is offered to it containing a simple formula of entertainment" (p. 255). The example of *American Idol* notwithstanding (although the show's eventual winners do generally seem to have legitimate vocal ability and much of the show's appeal lies in its spectacle of poor talent), this is why copycat films and trend-chasing television programs rarely succeed. The true promise of artistic performance support technology lies elsewhere.

If a musical composition software program were used to introduce a child to the concept of reading music, combined with the positive reinforcement of a technologically-enabled early compositional success, it could quite easily serve as a catalyzing inspiration for a lifelong love of music. And if, as a result, such a child happened to discover a previously unrealized natural musical talent and blossomed from that early nurturing into tomorrow's Ludwig van Beethoven or George Gershwin or Brian Wilson, that wouldn't be such a bad thing. With artistic performance support technology, anyone can now go from reader to writer, listener to performer. Benjamin's vision is realized, where the cultural democracy of artistic production grows increasingly more democratic every day.

REFERENCES

Adorno, T. W. (1982). On the fetish-character in music and the regression of listening. In A. Arato, & E. Gebhardt (Eds.), *The essential Frankfurt school reader* (pp. 270-299). New York: Continuum.

Adorno, T. W. (1941). On popular music. *Studies in philosophy and social science*. New York: Institute of Social Research, 17-48.

Barthes, R. (1979). Literature according to Minou Drouet. *The Eiffel Tower & Other Mythologies*. New York: Hill and Wang, 110-118.

Barthes, R. (1977). The grain of the voice. *Image-Music-Text*. Translated by Stephen Heath. New York: Hill and Wang, 179-189.

Barthes, R. (1977). Musica Practica. *Image-Music-Text*. Translated by Stephen Heath. New York: Hill and Wang, 149-154.

Barthes, R. (1970). *S/Z*. Translated by Richard Miller. New York: Hill and Wang.

Benjamin, W. (1968). The work of art in the age of mechanical reproduction. In H. Arendt (Ed.), *Illuminations* (pp. 217-251). New York: Shocken.

Bourdieu, P. (1984). *Distinction: A social critique of the judgment of taste*. Ed. R. Nice. London: Routledge

Bourdieu, P. (1993). *The field of cultural production: Essays in art and literature*. Ed. R. Johnson. New York: Columbia University Press.

Freedman, K. (2003). *Teaching visual culture*. New York and London: Teachers College Press.

Gunderson, E. (2005). *Music fans reach for the stars*. USA Today 9 Mar 2005. Retrieved

March 15, 2005, from http://www.usatoday.com/tech/webguide/music/2005-03-09-internet-jukebox_x.htm

Hayes, B. (1991). The information age. *Sciences,* 2, 13-15, March/April 31.

Huxley, A. (1949). *Beyond the Mexique Bay. A traveller's journal.* London: Heron Books, 274. First published in 1934. (Quoted from an endnote within: Benjamin, W. (1968). The Work of Art in the Age of Mechanical Reproduction. *Illuminations.* Ed. Hannah Arendt. New York: Shocken, 217-251.

Kahn, D. (1999). *Noise, water, meat: A history of voice, sound, and aurality in the arts.* Cambridge: MIT Press.

Kurzweil, R. (2005). *Ray Kurzweil's Cybernetic Poet: Features.* Kurzweil Cyber Art Technologies. Retrieved March 15, 2005, from http://www.kurzweilcyberart.com/poetry/rkcp_features.php3

Kurzweil, R. (2005). *Ray Kurzweil's Cybernetic Poet: How It Works.* Kurzweil Cyber Art Technologies. Retrieved March 15, 2005, from http://www.kurzweilcyberart.com/poetry/rkcp_features.php3

Kurzweil, R. (1992). A kind of turing test. In R. Kurzweil (ed.), *The age of intelligent machines* (pp, 374-379). Cambridge: MIT Press.

Lanham, R. A. (1993). *The electronic word: Democracy, technology, and the arts.* Chicago & London: The University of Chicago Press.

Mirzoeff, N. (1999). *An introduction to visual culture*. London & New York: Routledge.

Norman, D. A. (2004). *Emotional design: Why we love (or hate) everyday things.* New York: Basic Books.

Norman, D. A. (1988). *The design of everyday things.* New York: Basic Books.

Taylor, T. D. (2001). *Strange sounds: Music, technology and culture.* London: Routledge.

Thompson, E. (2002). *The soundscape of modernity* (1st ed.). Cambridge: The MIT Press.

Willis, A. M. (1990). Digitisation and the living death of photography. In H., Philip (Ed.), *Culture technology and creativity in the late twentieth century.* London: John Libby.

Write Brothers. (2005). *Writer's dreamkit product overview.* Screenplay.com. Write Brothers, Inc. Retrieved March 15, 2005, from http://www.screenplay.com/products/dreamkit/popup03.html

Write Brothers. (2004). *Software secrets of highly successful authors.* Write Brothers Marketing Literature. Glendale: Write Brothers, Inc.

This work was previously published in the International Journal of Technology and Human Interaction, edited by B. Stahl, Volume 4, Issue 3, pp. 27-42, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Section VI
Managerial Impact

This section presents contemporary coverage of the managerial implications of software applications. Particular contributions address agile practices in project management, virtual software teams, and computer-aided management of software development. The managerial research provided in this section allows executives, practitioners, and researchers to gain a better sense of how software applications can inform their practices and behavior.

Chapter 6.1
Open Source Software and the Corporate World

Sigrid Kelsey
Louisiana State University, USA

ABSTRACT

This chapter discusses various ways that open source software (OSS) methods of software development interact with the corporate world. The success achieved by many OSS products has produced a range of effects on the corporate world, and likewise, the corporate world influences the success of OSS. Many times, OSS products provide a quality product with strong support, providing competition to the corporate model of proprietary software. OSS has presented the corporate world with opportunities and ideas, prompting some companies to implement components from the OSS business model. Others have formed companies to support and distribute OSS products. The corporate world, in turn, affects OSS, from funding labs where OSS is developed to engaging in intellectual property disputes with OSS entities. The consumer of software is sometimes baffled by the differences in the two, often lacking understanding about the two models and how they interact. This chapter clarifies common misconceptions about the relationship between OSS and the corporate world and explains facets of the business models of software design to better inform potential consumers.

INTRODUCTION

Open source software (OSS) is impacting the corporate world in numerous ways, from providing software and competing with its proprietary software companies to changing the direction of the software industry. While some corporate giants are embracing the OSS business model, launching OSS projects of their own, and supporting existing OSS projects, others are vigorously competing with the OSS movement and its products. Still others are capitalizing on successful OSS products by packaging, distributing, and providing support for them. Sharma et al. (2002) assert that the success of OSS is turning the software industry from a manufacturing to a service industry in which customers are paying more for support and service than for the product itself. In addition, the OSS model of production

has gained recognition as an "important organizational innovation" (Lerner & Tirole, 2002, p. 1). Without a doubt, the OSS movement has had a substantial influence on the software industry and the corporate world.

BACKGROUND

Both the OSS and proprietary models of software productions have existed since the early days of software development. Unix, for example, was developed at Bell Laboratories in the late 1960s and early 1970s and distributed freely to universities during the 1970s. Unlike the altruistic motivations of many OSS products, the reason for Bell Laboratories' free distribution was to keep the "consent decree" that resulted from a 1956 antitrust litigation that prevented AT&T from marketing computing products (Vahalia, 1996). In fact, AT&T's 1979 announcement that it would commercialize UNIX prompted the University of California Berkeley to develop its own version, BSD UNIX (Lerner & Tirole, 2002). AT&T's move to make the cooperatively developed UNIX into a proprietary product came four years before Stallman's decision to develop GNU and General Public License.

By 1980, a business model for software had emerged, restricting the copying and redistribution of software by copyright. Bill Gates had already established himself as a supporter of this proprietary model, stating in his February 3, 1976, "An Open Letter to Hobbyists":

As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid? … Is this fair? … One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? (Gates, 1979)

Gates' letter indicates the differences in philosophy between proprietary and free software proponents that have existed since the early days of software development.

In 1984, computer scientist Richard Stallman, frustrated that all available operating systems were proprietary, quit his job at MIT to develop the GNU (pronounced guh-noo, a recursive acronym for GNU's Not Unix) system. His goal, in addition to developing a new operating system, was to change the way software was created and shared, giving users freedom to modify or add to programs, redistribute the programs with their changes, cooperate with each other, and form communities. Stallman also developed the concept of "copyleft" and the GNU General Public License (GPL) in 1989, publishing all of his work under that license. Copyleft gives software a copyright and users permission to change the software, add to it, and redistribute it, as long as it remains under the GPL terms. By preventing the software from entering the public domain, the GPL prevents users from turning free software into a proprietary derivative. Thus, the beginnings of the OSS movement were a reaction to the proprietary corporate model. In 1990, University of Helsinki student Linus Torvalds wrote the Linux kernel, releasing it under GPL, and filling the gap for a piece of Stallman's system still under development. Soon after, the Apache Web server was developed, providing an OSS application for Linux. This combination of software offered a new option to Internet service providers and e-commerce companies, which, until then, had only proprietary options.

Stallmans's Free Software Foundation Web page, reminding readers that free software means "free" as in "free speech," not as in "free beer" (Free Software Foundation, 2005), echoes a concept brought forth perhaps more eloquently by Thomas Jefferson and widely-quoted by OSS advocates that "ideas should freely spread from

one to another over the globe, for the moral and mutual instruction of man, and improvement of his condition ... ." With the growth of the OSS movement, some of the values of the OSS culture have diversified, but freedom and sharing remain integral to its success and completely dissimilar to the proprietary model of development and distribution.

The corporate and OSS models and philosophies continued to influence one another and develop throughout the 1990s. The "open source" label came out of a 1998 meeting, and shortly thereafter, the Open Source Initiative was created. Also in 1998, the Digital Millennium Copyright Act (DMCA), criminalizing the production of software for the purpose of evading copyright, and the Sonny Bono Copyright Term Extension Act, extending U.S. copyright terms by 10 years, both passed. Despite the divergent directions the two movements were taking, the difference between free software and proprietary software has never been reducible to a battle between anti-corporate OSS proponents and the profiteering corporate world, as many people perceive. Corporate companies with a stake in the software industry have, in fact, navigated various approaches to succeed in an industry in which the motivations for developing software go beyond the commercial value of the product or ownership of intellectual property. While the two models of software production and distribution are competitive in many ways, it was also in the 1990s that it became common for commercial companies to interact with the OSS community (Lerner & Tirole, 2002).

During 1998, Torvalds appeared on the cover of *Forbes*, Netscape announced a decision to make the next version of its Web browser an OSS product, and IBM adopted the Apache Web server as the core of its Websphere line of products (O'Reilly, 1999). Like IBM, some corporate giants have chosen to use and support OSS voluntarily. Others have found it necessary to contribute in order to market products to Linux users; still others have fought intellectual property battles with OSS constituents. The relationship between the OSS and corporate cultures is complex, but it is clear that the OSS culture is making an impression on the corporate world, and vice versa.

As the OSS community has grown to include professionals, students, hobbyists, corporate giants, universities, and others, the freedom of ideas and sharing knowledge remains the crux of the OSS ideology. To integrate into the OSS culture, therefore, the corporate world must be willing to share its developments. This chapter summarizes some ways in which the OSS movement has motivated change in the software industry and corporate world, citing some specific examples of corporations reacting to OSS software and strategies in different ways, which serves to illustrate the larger picture.

MAIN FOCUS OF THE CHAPTER

Corporate Culture and Motivations of OSS Developers

A misconception often associated with OSS developers is that they are volunteer programmers, willing to "dedicate their time, skills, and knowledge to the OSS systems with no monetary benefits" (Ye & Kishida, 2003, p. 1). In fact, there are many money-making opportunities for open source developers, from providing software support to programming for companies or institutions using the software. While it is true that many OSS developers are paid to make the developments, Eric Raymond is quick to point out that while OSS developers may be paid for their contributions to the software, their salaries rarely depend on the sales value of their software (Raymond, 2001). OSS contributors may work for support companies, universities, and other organizations with motivations not attached to selling the software. This is a key difference between the OSS and proprietary software business models. Stallman's

1985 GNU Manifesto not only outlines his reasons for creating GNU but also offers some suggestions for how programmers can make money in an OSS environment.

While too often OSS advocacy is reduced to an anti-Microsoft position, the challenge that the OSS community has posed to the software giant does provide an illustration of the extent of the movement's success. For example, throughout the Microsoft Corporation antitrust case, Linux was a named threat to Windows domination, with Microsoft CEO Steve Ballmer referring to open source as a "cancer" (Microsoft Exec, 2006). In the midst of the antitrust trial, Eric Raymond became the recipient of two leaked internal Microsoft memos, posting them on a Web site and naming them the "Halloween Documents." The documents, acknowledged by Microsoft to be authentic but according to Raymond dismissed as an engineering study not defining company policy, discuss the success of Linux, acknowledging the achievements of the OSS movement and outlining strategies to "beat" Linux (Raymond, n.d.). Microsoft is not the only corporation combating the success of Linux and other OSS products.

The SCO Group is a software company currently involved in a number of disputes regarding intellectual property, including lawsuits with IBM, Red Hat, and Novell. SCO filed a complaint against IBM in March 2003 claming that IBM has misappropriated SCO's proprietary knowledge by contributing to the GNU/Linux systems with code SCO claims to own, alleging damages of at least $1 billion. The result of the ongoing litigation will set a precedent for future cases.

In 2005, Columbia University law professor Eben Moglin formed the Software Freedom Law Center to help protect OSS development from similar litigation. The center provides pro bono legal services to FOSS projects and developers; its mission to help provide FOSS developers with "an environment in which liability and other legal issues do not impede their important public service work. The Software Freedom Law Center (SFLC) provides legal representation and other law-related services to protect and advance FOSS." His foundation is one of several helping to defray legal costs for litigation against FOSS developments. The Open Source Development Labs (OSDL) Linux Legal Defense Fund has raised more than $10 million to provide legal support for Linus Torvalds and others subject to SCO litigation (Goth, 2005, p. 3).

While Microsoft and SCO are resisting the OSS model of business, others in the corporate world have come to see the benefit of working with OSS producers and products. Silicon Valley's NetApp, for example, became involved in Linux because its Linux-using customers were experiencing difficulty moving files between their computers and NetApp filers. Although it was a Linux problem, customers complained to NetApp, and with a vested interest in fixing it, NetApp cofounder and chief of engineering talked to Linus Torvalds. Mistrustful of companies like NetApp, Torvalds declined NetApp's offer to fix the problem, naming instead his choice programmer for the job, Trond Myklebust. NetApp, along with Linux developers worldwide, could submit suggestions to Myklebust in hopes that he would accept them. Therefore, if NetApp was to market its product to Linux users, it was obliged to join the OSS culture (Lyons, 2004). The NetApp circumstances demonstrate that any company wishing to make its product compatible on a Linux platform has a stake in the OSS world. Yet while the OSS culture is able to influence the actions of the corporate world, so the corporate world is able to do likewise.

In 1999, NetApp began funding the University of Michigan's Center for Information Technology Integration (CITI), home to a lot of Linux NFS development. By 2002, NetApp was paying Myklebust a stipend and providing him office space in the lab and a company-paid apartment in Ann Arbor. Peter Honeyman, scientific director of the lab where Myklebust works who receives $192,000 a year from NetApp, notes, "What's in it for [NetApp] is sales; it can sell into the Linux

market. This is not about philanthropy. There is plenty of mutual benefit going on here" (quoted in Lyons, 2004). Torvalds, who was mistrustful of NetApps' offer to help, works at a Beaverton, Oregon, lab funded in part by Hewlett-Packard (Lyons, 2004). In response to the apparent conflict between picking up a salary from a revenue-hungry corporation and developing OSS, Torvalds compares himself to an athlete with a corporate sponsor (Lyons, 2004).

The OSS culture, therefore, is not separate from the proprietary model of development; rather, the two models interact with and influence each other in many ways. Indeed, the OSS, corporate, and academic worlds have a complex relationship, each able to control, to some extent, the others' directions.

A 2004 *Forbes* article notes that many top technical firms hire Linux programmers in hopes of manipulating the direction of Linux development (Lyons, 2004). Hewlett Packard Vice President Martin Fink acknowledges that the closer he can get to Torvalds, the more influence he can have on Linux, saying "I try to keep it under two hops. ... The way to get stuff done in the Linux community is to hire the right people." In 2003, Hewlett Packard generated $2.5 billion in Linux-related revenue; IBM $2 billion; and Red Hat, which distributes a version of Linux, $125 million in revenues. Linux runs in datacenters of places like Charles Schwab & Co. and Sabre Holdings (Lyons, 2004). These corporations have recognized the benefits of the OSS culture, and many of them have become sponsors of its research.

IBM has been a powerful corporate advocate of OSS development for years. In 2005, IBM promised free use of 500 of its U.S. patents to open source developers, stating, "The open source community has been at the forefront of innovation and we are taking this action to encourage additional innovation of open platforms" (IBM, 2005). IBM's Bob Sutor, vice president of standards, says that this move was made in hopes of starting a "patent commons" for companies to contribute intellectual property for open source developers to use freely without fear of litigation (Goth, 2005, p.4) from companies like SCO. OSS supporters generally believe that software patents hinder advancements in software research; in Europe, efforts are underway to prevent laws that would allow the patenting of software (Carver, 2005).

With IBM and other such corporations designing their products with OSS platforms in mind, and contributing to the furthering of OSS research, it is plain that the two cultures have learned to work together.

Reliable Code, Reliable Support

The success of OSS projects like Linux, Apache, and Perl evince the success of the bazaar model on the code itself. The traditional paradigm of collaborative development follows Brooks' Law, which ascertains that only a select circle of experts should be allowed to collaborate, with little or no feedback, to improve a product before it is finished. Brooks' Law states, "Conceptual integrity in turn dictates that the design must proceed from one mind, or a very small number of agreeing resonant minds" (Neus & Scherf, 2005, p. 216). Eric Raymond dubs the bazaar approach Linus' Law in which software is released early and often, evolving as users around the world use it and contribute to it. Making the code freely available and open to review by one's peers makes the quality better (Bergquist & Ljungberg, 2001). Open source has proved itself to be a formidable model for creating quality software, and as OSS projects become even more widely adopted, the culture and communities grow larger.

Customers often question the availability or longevity of support available for OSS. Without a revenue-generating company supporting it, it is difficult for OSS newcomers to imagine that any support will exist. But the success of OSS projects like Linux, GIMP, and Apache provide examples of the bazaar model's success. In fact, with proprietary software, the support is propri-

etary as well, where anyone who is able to provide support for OSS is free to do so. O'Reilly Media, Inc., a strong supporter and early advocate of OSS, points out on its Web site that its success came in part because it was not "afraid to say in print that a vendor's technology didn't work as advertised" (O'Reilly, n.d.). Besides publishing numerous support books, O'Reilly provides online services and hosts OSS summits and conferences. O'Reilly is not the only company providing support for OSS. The OSS culture of sharing and helping gives assurance that with any successful OSS product, adequate support is available.

FUTURE TRENDS

A July 2005 article reports that 70% of Web servers on the Internet use Apache compared to roughly 25% using Microsoft's Internet Information Server (Bradbury, 2005). Already, European governments have adopted OSS for their computing needs, and California has started a U.S. trend toward the same, making a 2004 recommendation for the use of OSS in its performance reviews. Products like OpenOffice, named by Developer.com as a 2006 Open Source Product of the year, offer products that are able to compete with Microsoft's Office Suite. The state of Massachusetts is currently deciding whether to go forward with a decision made by the former CIO to use OpenOffice, with Harvard Law School Professor John Palfrey predicting, "If Massachusetts gets this right, others will follow" (McMillan, 2005).

OpenOffice is already common in Israel, in part because OpenOffice works well with the Hebrew language and because Microsoft software is expensive. China, where software theft has discouraged proprietary companies from marketing, has embraced OSS, creating the China Standard Software Company (CSSC) and the China Open Source Software Promotion Alliance (Bradbury, 2005). The growing trend to adopt OSS has spread worldwide. With such support, the OSS culture and movement will continue to grow. Market researcher IDC predicts that by 2008, Linux server sales could approach $10 billion.

The GPL has yet to be ruled enforceable in a U.S. court of law; until now, it has only been enforced in private negotiation or settlement agreements (Carver, 2005). In Germany, however, a Munich district court has ruled it valid and enforceable (Carver, 2005). The result of ongoing litigation between SCO and IBM will set a precedent for how the GPL is interpreted in the United States.

Lerner and Tirole acknowledge that the future of the open source development process is difficult to predict with existing economic models and that further research is needed from an economic perspective.

CONCLUSION

The predominating shared norms, values, attitudes, and behavior that characterize OSS culture are deeply rooted in valuing freedom and sharing. As OSS has grown to offer software options for large entities like governments, companies, and universities, reasons for joining the OSS movement diversify. While the movement has grown and the culture has shifted, the basic values have remained in tact. Its success has impacted other cultures and traditions worldwide, from academic publishing and research to government to the corporate world. Clearly, the initial ideas and philosophies set forth by Jefferson and echoed by Stallman are affecting the culture of research worldwide, with the OSS movement proof that a culture of sharing is beneficial to everyone involved.

REFERENCES

Association of College and Research Libraries (ACRL), Association of Research Libraries (ARL), SPARC, & SPARC Europe. 2003. *Create: New systems of scholarly communications; change: old systems of scholarly communication.* Retrieved from http://www.createchange.org/resources/CreateChange2003.pdf

Bergquist, M., & Ljungberg, J. (2001). The power of gifts: Organizing social relationships in open source communities. *Information Systems Journal, 11*(4), 305.

Bradbury, D. (2005). *The future is open source: Should Microsoft be watching its back?* Retrieved January 18, 2006, from http://www.silicon.com/research/specialreports/opensource/0,3800004943,39150625,00.htm

Carver, B. W. (2005). Share and share alike: Understanding and enforcing open source and free software licenses. *Berkeley Technology Law Journal, 20*(1), 443.

Free Software Foundation. (2005). *The free software definition.* Retrieved December 1, 2005, from http://www.fsf.org/licensing/essays/freesw.html

Gates, W. H., III. (1979). *An open letter to hobbyists.* Retrieved July 7, 2006, from http://www.digibarn.com/collections/newsletters/homebrew/V2_01/homebrew_V2_01_p2.jpg

Goth, G. (2005). Open source infrastructure solidifying quickly. *IEEE Distributed Systems Online, 6*(3), 1.

Hars, A., & Shaosong, O. (2001). *Working for free? Motivations of participating in open source projects.*

IBM. (2005). *IBM statement of non-assertion of named patents against OSS.* Retrieved January 29, 2006, from http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf

Lattemann, C., & Stieglitz, S. (2005). Framework for governance in open source communities. In *Proceedings of the 38th Hawaii International Conference on System Science.*

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics, 50*(2), 197.

Lyons, D. (2004). Peace, love and paychecks. *Forbes, 174*(5), 180. Retrieved July 11, 2006, from http://www.msnbc.msn.com/id/5907194/

McMillan, R. (2005). CIO who brought OpenOffice to Massachusetts resigns: Whether Peter Quinn's departure helps or hinders state's move away from Microsoft remains to be seen. *InfoWorld.* Retrieved January 29, 2006, from http://ww6.infoworld.com/products/print_friendly.jsp?link=/article/05/12/28/HNmasscio_1.html

Microsoft exec leaves suddenly: Windows live official helped company fight Linux threat (2006, June 21). *The Seattle Post-Intelligencer,* p. E1.

Moore, J. (2003). *Revolution OS: Hackers, programmers & rebels unite.* In W. Productions (Producer): Seventh Art Releasing.

Neus, A., & Scherf, P. (2005). Opening minds: Cultural change with the introduction of open source collaboration methods. *IBM Systems Journal, 44*(2), 215.

Open Source Initiative. (n.d.). *The open source definition.* Retrieved December 1, 2005, from http://www.opensource.org/docs/definition.php

O'Reilly, T. (n.d.). *O'Reilly media: History.* Retrieved July 11, 2006, from http://www.oreilly.com/history.html

O'Reilly, T. (1999). Lessons from open source software development. *Communications of the ACM, 42*(4), 32-37.

Raymond, E. S. (n.d.). *The halloween documents.* Retrieved January 30, 2006, from http://www.catb.org/~esr/halloween/index.html

Raymond, E. S. (2001a). *The cathedral & the bazaar.* Beijing: O'Reilly.

Raymond, E. S. (2001b). *How to become a hacker.* Retrieved January 24, 2006, from http://www.catb.org/~esr/faqs/hacker-howto.html#what_is

Rheingold, H. (1994). *Virtual community.* London: Minerva.

Sharma, S., Sugumaran, V., & Rajagopalan, R. (2002). A framework for creating hybrid-open source software communities. *Information Systems Journal*, *12*, 7.

Stallman, R. (1985). *The GNU manifesto.* Retrieved from http://www.gnu.org/gnu/manifesto.html

Unsworth, J. M. (2004). The next wave: Liberation technology. *The Chronicle of Higher Education*, *50*(21), B16-B20.

Vahalia, U. (1996). *UNIX internals: The new frontiers.* Upper Saddle River, NJ: Prentice-Hall, Inc.

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)* (pp. 419-429).

KEY TERMS

Free Software (FS): Software that users have the freedom to alter, use, and redistribute, usually under the terms of the General Public License. Closely related to Open Source Software, the two terms are sometimes used interchangeably. "Free" is not associated with cost but with the freedom associated with it. However, free software is often cost-free as well.

General Public License (GPL): A license created by Richard Stallman that protects free software from being turned into proprietary software.

Open Source Software (OSS): Software that allows the user to see and alter the source code; closely related to free software.

Proprietary Software (PS): Software that does not allow the user to see or alter the source code.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant; B. Still, pp. 570-577, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 6.2
Combining Tailoring and Evolutionary Software Development for Rapidly Changing Business Systems

Jeanette Eriksson
Blekinge Institute of Technology, Sweden

Yvonne Dittrich
IT-University of Copenhagen, Denmark

ABSTRACT

This article reports on a case study performed in cooperation with a telecommunication provider. The telecom business changes rapidly as new services are continuously introduced. The rapidly changing business environment demands that the company has supportive flexible software. The company's continuous evolution of the IT-infrastructure makes it necessary to tailor the interaction between different applications. The objective of the case study was to explore what is required to allow end-users to tailor the interaction between flexible applications in an evolving IT-infrastructure. The case study followed a design research paradigm where a prototype was created and evaluated from a use perspective. The overall result shows that allowing end-users to tailor the interaction between flexible applications in an evolving IT infrastructure relies on, among other things, an organization that allows cooperation between users and developers that supports both evolution and tailoring.

INTRODUCTION

End-user development (EUD) is one way to provide a flexibility that allows companies to compete in rapidly changing business environments. Telecommunication provision is one such example of a rapidly changing business area. Telecommunication providers compete by, among other things, providing their customers with new

types of services, and as the business changes, the business systems supporting it must also change. One way of conducting EUD is end-user tailoring. End-user tailoring is an activity allowing end-users to modify the software while it is already in use, as opposed to modifying it during the development process (Henderson & Kyng, 1991). End-user tailoring ranges from setting the values of parameters to adding code to the software. Since evolution of software is inevitable (Lehman, 1980) and since tailoring is recognized as a way of reducing the efforts when keeping the system up to date through further development (Mørch, 2002), tailoring could be an alternative to increase the sustainability of software in a rapidly changing business environment.

Tailoring research so far has focused on flexible stand-alone systems. In earlier projects, we too focused on the design of flexible and end-user tailorable applications (Lindeberg, Eriksson, & Dittrich, 2002). However, interaction with other systems turned out to be a bottleneck, since business systems in telecommunication are part of an IT-infrastructure consisting of heterogeneous data sources. Other research also indicates that software and IT-infrastructures pose new challenges for software engineering (Bleek, 2004). Normally, the data exchange between different systems is the realm of the software developers, but in this article we use the evaluation of a prototype to answer the question: What is necessary to allow *end-users* to tailor the interaction between flexible applications in an evolving IT-infrastructure? Our results support the claim that end-users can even tailor the interaction between business applications. The analysis of a user evaluation of a case-based prototype results in a number of issues to be addressed regarding the technical design, the know-how demanded of the users, and the organizational setting, particularly the cooperation between users and developers. These issues both confirm and extend existing research on end-user development and tailoring.

We start by briefly describing the relevant work practices and business systems of our industrial partner. We then present how our research relates to others' work. In the following section, we describe our research approach in detail and the design of the prototype is presented to provide a basis for the evaluations and discussions. Thereafter, we present the outcome of the evaluation, which points out three different categories of issues that are important when providing end-users with the possibility to manage interactions between applications in an evolving IT-infrastructure. The discussion relates these results to the state of the art.

HISTORY AND BACKGROUND

The research reported here is part of a long-term cooperation between the university and a major Swedish telecommunication provider, exploring the applicability of end-user tailoring in industrial contexts (Dittrich & Lindeberg, 2002). The subject of the prototype is part of the telecommunication provider's back office support infrastructure for administering a set of contracts and computing payments according to these contracts. To compute payments, the system must be supplied with data from other parts of the IT-infrastructure. When creating new contract types based on different data, flexibility is constrained by the hard-coded interface to other systems. As a work-around, ASCII files can be created providing the necessary data sets – or events – to compute the payments. The data for these extra payments is handled and computed manually. To compute the data for an extra payment, members of the administrative department first run one or more SQL queries against the data warehouse. The result is stored in ASCII files. Next, the user copies the data from the ASCII files and pastes it into a prepared spreadsheet. When the user has thus accumulated the data, the user works through the spreadsheet in order to remove irregularities. The contents of the sheet are eventually converted again to an ASCII

file that is imported into the payment management system. The manual procedure to compute the data for the extra payments has worked well until recently, although it is time consuming. The competitiveness of the telecom business is however continually forcing the company to come up with new services; more and more types of extra payments will be needed. This situation necessitates a tool to define and handle the new data sets or events. To make such event tool as flexible as possible, it must allow the collection and assembly of data from different kinds of systems. Experience suggests that it is impossible to anticipate the structure of future extra payments or which details will be needed. As a result, the tool must be able to communicate with any system in the IT-infrastructure. It is also essential that the tool allow for expansion of the tailoring capabilities, meaning that new data sources can be added. The addition of a new source should be as seamless as possible. Since different system owners and developers are responsible for these systems, it is their responsibility to make new data sources available. Such changes are part of the maintenance of the other systems, and here the limits of end-user tailoring are reached.

RELATED WORK

The research on end-user tailoring addresses mainly the design of tailorable applications, tailoring as a work practice, and cooperation between users and tailors. Examples of research on the design of tailorable systems are (Eriksson, Warren, & Lindeberg, 2003; Lindeberg et al., 2002; Mørch, 1997; Stiemerling, 2000; Stiemerling & Cremers, 1998). Of these, only two address tailoring in the context of distributed systems. In Eriksson et al. (2003) a prototype that dynamically connects different physical devices (video cameras, monitors, tag readers, etc.) is presented. The tool can be regarded as tailoring the interaction between different intelligent devices. Stiemerling (2000) and his colleague (Stiemerling & Cremers, 1998) show how to build a search tool by using customized Java Beans. The users customize search and visualization criteria. The tailorable search tool is used within a distributed environment provided by a groupware system. Neither of the distributed tailoring approaches is evaluated by users to explore beyond technical issues of how end-users can manage interaction between applications.

Several researchers have studied how tailoring activities are carried out in work practice, for example (Gantt & Nardi, 1992; Trigg & Bødker, 1994). In a study involving tailoring spreadsheets, Nardi and Miller (1991) identify collaboration between three kinds of users of CAD (Computer Aided Design) systems (1) users who do not program (2) users who acquired the skill to program small macros, and (3) local developers: users having a more or less formalized responsibility for supporting other users and maintaining the macro selection of a group or department.

Carter and Henderson (1999) invented the expression *tailoring culture* to express the need for organizational support for tailoring. Kahler (2001) also points out that, in order to make tailoring successful, an organizational culture must evolve that supports the development and sharing of tailoring knowledge. Kahler also emphasizes three often coexisting levels of tailoring culture, identified and addressed by different researchers:. First there is a level with equal users; people help each other to tailor the software (Gantt & Nardi, 1992) or there is a network of whom to ask when encountering trouble when tailoring the software (Trigg & Bødker, 1994). Second, there is a level with different competencies (Gantt & Nardi, 1992). The third level is a level of organizational embedment of tailoring efforts and official recognition of tailoring activities (MacLean, Carter, Lövstrand, & Morgan , 1990). We will return to this classification in the discussion of our results, as our findings propose the consideration of a fourth level of tailoring culture when implementing and deploying tailoring possibilities in an IT-infrastructure environment.

THE CASE STUDY

Research Approach

Our research approach can be described as a single case study (Yin, 2003) following a design research paradigm (Nunamaker, Chen, & Purdin, 1991). The question "What is necessary to allow end-users to tailor the interaction between flexible applications in an evolving IT-infrastructure?", addresses the design and deployment of a previously inexistent functionality. In design research, the design and development of a (prototypical) information system can be used both to answer technical questions and as a probe to explore requirements posed by the deployment of the technical possibilities. Hevner, March, Park, and Ram (2004) especially emphasize the need for combining design research and behavioral science. The technical design of the prototype is discussed in (Eriksson, 2004).

The practical work was conducted during a period of slightly more than one and a half years. Prior research indicates that the collection of data to process the so-called extra payments was a bottleneck both for the users' work as well as for deploying the flexibility implemented in the existing systems. During the initial field studies focusing on the work practice of the business department, we visited our industrial partner once or twice a week to observe and interview both users and developers. These field studies informed the development of the overall research question and also the design of the prototype.

In the beginning of the design phase, workshops were arranged involving researchers, users and developers. When designing the prototype, one of the researchers was stationed at the company two or three days a week to ensure that the prototype conformed to existing company systems. Field notes were taken, and meetings and interviews were audio taped, during all phases of the case study.

The prototype was evaluated by all three employees involved in the collection of data and computation of the extra payments and by one developer involved in the maintenance of the payment system. These evaluations were video taped. The analysis in the section of this article entitled "Outcome of Evaluation" is mainly based on the latter tapes, but uses the other field material as a background. For secrecy reasons, videotaping is not allowed on the telecommunication provider's premises. We therefore installed the system on a stand-alone computer outside the actual work place. To allow the users to evaluate the prototype realistically, we reconstructed part of the IT-infrastructure in a local environment and populated it with business data, developing our prototype into a case-based prototype (Blomberg, Suchman, & Trigg, 1996). The users were given two tasks. One task was to construct the collection and assembly of data for an extra payment that they implemented regularly (in the manual fashion described above) as part of their normal work. For the second task they had to construct a totally new but realistic payment. The users were asked to *talk-aloud* while performing the task. This method is common when evaluating software in a use context (Ericsson & Simon, 1993; Robson, 2002). The researcher performing the evaluation observed and asked exploratory and open-ended questions to provoke reactions that differed from our expectations. The developer who worked with maintenance of the regular system evaluated the prototype in a workshop, and discussed advantages and drawbacks concerning use, tailoring, and expansion of the tailoring capabilities.

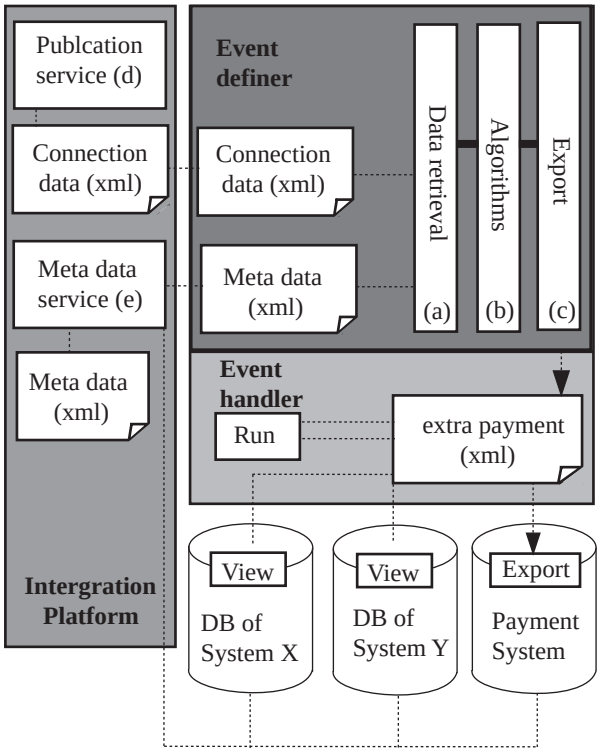
We analyzed the data in a manner that was inspired by grounded theory. A coding scheme was developed with its starting point in the transcripts of the evaluation sessions. The researchers coded the interviews independently from one other and then compared their results. The resulting categories were finally merged into three core categories, that is, design issues, user knowledge, and organizational and cooperative issues. The categorization can be found in the evaluation section.

The Prototype

The prototype is divided into two parts, the Event Definer and the Event Handler (Figure 1). By using the Event Definer, the end-user can tailor communication and data interchange between systems, that is, the end-user defines the event types for the computation of the above-described extra payments. It allows the user to: define the assembly of data from different sources (Figure 1a), set up rules for aggregation and algorithms that will be performed on the data when aggregating the data (Figure 1b) and define how to map data sets to the format required by the receiving application. (Figure 1c). The Event Definer needs to be used only when defining new types of extra payments. The Event Handler handles the execution of extra payments or events and is to be used once a month to run the different extra payments.

Various solutions exist that provide the functionality needed to manage the connections between applications. These are found in tools for system integration that connect systems, in network management for monitoring the IT-infrastructure, in component management (if you choose to regard the different systems as components) and in report generation for assembling data. These tools are designed exclusively for system experts, not for end-users. A possible exception is report generation, which sometimes supports end-users but often needs support from developers to adapt it to fit new data sources. We found that none of these approaches was suitable for fulfilling the requirements for a tool for interapplication communication that can be adapted by users. Neither were the approaches suitable for the purpose of exploring what is necessary to allow end-users to tailor the interaction between flexible applications in an evolving IT-infrastructure. For our prototype we used an existing platform that supports integration between the telecommunication provider's back office applications. The

Figure 1. The connection between the prototype and the surrounding systems



integration platform makes it possible to publish events that other applications can subscribe to. We had a somewhat different intention when using the platform. We wanted to collect the information when needed, and we used the platform to provide the prototype with information about how to get in touch with desired resources and what data were accessible at these resources. We created a service (Figure 1d) on the integration platform that allowed the developers of the different systems to publish information about available data and showed how to connect to the respective database. To do so, the developers must set up a database *view* containing data that could be accessible to other systems (such as the prototype). The service produced an XML file containing connection data for all published data sources. When the Event Definer starts, the XML file is fetched from the integration platform. Yet another service (Figure 1e) provided the prototype with metadata from the data sources, for example, which fields (attributes) could be accessed in a specific database, and the types of the fields.

Tailoring

The graphical tailoring interface of the Event Definer was constructed to consist of seven different steps. These steps are intended to guide the user through the process, but could also be used in an arbitrary order as the end-user chooses.

Step 1: Naming the extra payment.
Step 2: Choosing which databases to connect to.
Step 3: Choosing which fields to use from the selected databases.
Step 4: Setting up criteria for what data to collect from the different databases, that is, by drag and drop, the end-user chooses which field should be used and the end-user can also specify how the different views should be linked together, for example, fieldX in SystemX must be equal to fieldY in SystemY.
Step 5: Showing the specified criteria from Step 4 as SQL queries, that is, here the user can edit the SQL queries to set up more complicated (and unusual) conditions for data retrieval than can be accommodated by the graphical interface.
Step 6: Setting up algorithms for what to do with the collected data. (partially implemented.)
Step 7: Mapping the input table structure to the output table structure, that is, the end-user can map the assembled and computed data to a receiving system by dragging the fields from the assembled data table and dropping them in a table representing the receiving database.

All these choices, criteria, algorithms, mapping and so forth, were finally brought together and arranged into an XML file (extra payment in Figure 1).

Use

The XML files produced by the Event Definer are then used whenever the end-user decides to execute the extra payment. The Event Handler contacts the chosen systems one by one and collects the data specified in the XML file. When the data is collected and assembled in a single table it is displayed to the user to allow for checking and correcting the result where necessary. By clicking on a button, it is possible for the end-user to export the result to the system handling the payment data, in accordance with the mapping specification (Step 7).

Expansion of Tailoring Capabilities

There will inevitably be situations where end-users wish to define extra payments based on data that is currently unavailable. If the data and metadata are unavailable, the end-users are unable to perform new tasks. They have neither the

authority nor the ability to alter or add views in surrounding systems. In this case the surrounding systems, as well as the tailorable system, have to evolve to meet the additional requirements from the end-users. The developer responsible for the respective system must then (a) alter the system by creating a new view or changing an existing view, so that it contains the required data, and (b) make the changes available through the integration platform. To support the latter, the publication of a new source was supported by a web interface where the developer (also system owner) could fill in the necessary data.

Outcome of Evaluation

The evaluation presented here focuses on issues beyond the technical design and the appearance of the graphical interface of this specific application. It addresses overall design issues for this kind of application, the end-user knowledge necessary to handle such complex tailoring tasks, and organizational issues to deploy such systems in a sustainable way. We have also evaluated the prototype against functional requirements, but the results are not reported here. Individual opinions held by only one or two of the subjects are disregarded in the following presentation.

Design Issues

In terms of technical support we focused on the different interfaces provided by the prototype: the tailoring interface, the deployment interface and the development interface.

The Tailoring Interface

Functionality for Controlling and Testing

All users appreciated the freedom to alternate between the seven steps. They found that the steps provided not only guidance and an overview but also the freedom to alter something performed in previous steps, without losing the overall view. To be able to overview all choices and trace them backwards was one way of providing control. But there was also a need for error control and limitation. The users, especially the beginners, wanted some kind of guidance in order to feel secure.

It became very obvious that the design must enable the end-user to test and control the correctness of the specification of extra payments. Control facilities must be provided to ensure security for the users in their work. Although control and test functionality was important for all users, the attitude towards test and control varied between the users. The better the knowledge of the task, the surrounding systems and possible errors, the less important explicit test and control seemed to be. Following statements exemplifies different attitudes towards control and test functionality: *When you make an extra payment for the first time you would probably like to make a test run to see that it really works correctly.* (user comment, evaluation session, February 24, 2004)

and

there isn't the same protection as in SystemZ … but to make a more flexible solution, then you can't expect it to be strictly user friendly (user comment, evaluation session, February 24, 2004).

Clear Division Between Definition, Execution and the Tailoring Process

When tailoring, the user rises from one level of abstraction to another, higher level. From thinking only in terms of the execution of an extra payment the user had to think in more general terms of what characterizes this extra payment, what kind of data were fetched, what variables there were, and so forth. The users had to think in terms of levels which is not an easy step to take. We found that a clear separation between execution and the tailoring process helped the users to make this step successfully.

The users also started to discuss the division of labor enabled by a system resembling the prototype. For example, one of the users said:

I think it is very good because then someone is very familiar with how to make a new extra payment and then all employees in the group can run the extra payment. (user comment, evaluation session, February 24, 2004)

Unanticipated Use Revealed to the Tailor

Systems that continuously evolve through tailoring aim to support unanticipated use. The possibilities for unexpected use are inevitably limited by the technical design. To support unanticipated ways of tailoring, the system has to provide additional information of what is possible to do and what the limitations are. In the prototype this was achieved by providing information for the user that is not directly applicable to the type of extra payments that exist today. As one of the users expressed it when seeing the opportunity for one of the export systems to also act as input source:

this is interesting! It opens up new opportunities. It might be like one extra payment uses another payment as a base (user comment, evaluation session, February 24, 2004).

Complexity

We found that the users preferred more information, rather than a less complex tailoring interface, resulting in more tailoring possibilities. Their opinion was that, as tailoring is not routine work, performed several times a day, it is allowed to take extra time. Then it is better to have a more complex interface providing more opportunities to tailor the system.

The Deployment Interface

Simplicity

One thing that was revealed and worth mentioning is that it seems that the deployment interface should be even simpler than an ordinary user interface. The users expressed the opinion that the tailoring interface and the tailoring process may be rather wide-ranging if that allows for a simpler deployment interface.

The Development Interface

One Point of Interaction

The development interface in the prototype was a graphical Web interface where the developer could fill in the data that was to be published about the respective source system. During the evaluation of the development interface, the software engineer emphasized the importance of having one point where changes to the data sources are published. The developer should not be forced to make changes in several places in the application in order to extend the tailoring capabilities.

End-User Knowledge Required for Tailoring

Even previous to the evaluation session we had experienced the high expertise of the users not only regarding their tasks but also regarding the data available in the different databases that are part of the IT-infrastructure. The users acquired the knowledge in order to perform the assembly manually. The communication between different systems is normally hidden from the user in a data communication layer for the separate systems. Our prototype is designed to make exactly this communication tailorable. Its deployment depends on the respective expertise of the users.

Task Knowledge

Business knowledge about contracts and payments provides the base on which the users decided what data to collect. Extensive business knowledge was a prominent feature of the results of the evaluation. The users' reflections on which data to collect always concerned different aspects of the business tasks.

System Knowledge

To map requirements regarding the task at hand and the available data, demands expertise regarding the available data in the different systems. And the users knew where to find the data needed for defining a specific extra payment. The prototype just helped with the exact location of the data, for example it guided the user to which fields to use, by listing the fields with examples of the data they contained. However, the user had to understand the sometimes quite cryptic names and know where to look for specific data.

Error Knowledge

All users were extremely aware of which errors could occur,that is, errors concerning the use of the prototype, the IT-infrastructure and the task. Task-specific errors are particularly important for the end-user to overview since they may cause serious consequences for the company if the errors are not prevented. On several occasions during the user tests the users expressed concern about making errors. They made statements like:

when you work as we do you must know a little about database management, you have to understand how the tables are constructed and how to find the information. And also in some way understand the consequences of or the value of the payment. In other words how you can formulate conditions and what that leads to. (user comment, evaluation session, February 24, 2004)

Organizational and Cooperative Issues

The system for which the prototype was a test would depend on data published by many different surrounding programs. Each one of these systems is itself the subject of both tailoring and evolution. Both the users, and the software engineer who evaluated the prototype, addressed the necessary interaction with other system owners and the assignment of responsibilities regarding the publication and updating of the connection information and the kinds of data available.

Publication and Update Responsibilities

During the workshops it became apparent that there is already friction in the coordination between the payment system and the changes in the surrounding systems. When one system in the IT-infrastructure is changed, the changes are orally communicated to the owners of other systems that may or may not be affected by the change. For the prototype to function as designed, it was important that the systems that the prototype was expected to communicate with were visible and accessible. The design of the prototype solved this problem by requiring every change relevant to the prototype to be reflected in the published information. In other words, it was designed so that the respective system owners were responsible for keeping their system visible and showing its current status. As the prototype was dependent on accurate just-in-time information, the evaluation revealed a need for coordination concerning publication and updates of surrounding systems and tailoring activities in the prototype.

Collaboration Between Developer and End-Users

The fieldwork revealed, and the evaluation confirmed, that it is impossible to know what future contracts will look like. Therefore there will

always come a time when the end-user wants to retrieve data that is not published in any available view. In this case the system that can provide the data has to be identified and the respective system owner or developer has to be persuaded to implement a new view of the system or update existing ones, and publish the relevant information.

Another issue related to communication and cooperation between users and developers concerned the decision of how much information to make available for the users to do a good job of tailoring. The users wanted to see as much information as possible, provided it was within reasonable limits. In order to have better control over the execution of the system and to decouple maintenance that would not necessarily impact the communication with the payment system, the developers would rather prefer to restrict the user's options. These two perspectives have to be negotiated.

In this company, cooperation between business units and the IT unit works very well. The users evaluating the system were quite aware of the limit of their own competences and knew when to consult the responsible developers. All users frequently referred to developers when they experienced that something was beyond them. None of them considered the necessary coordination and cooperation to be a serious problem.

Summary of Outcome of Evaluation

The evaluation revealed many issues to consider when making a system that continuously evolves through tailoring work in a rapidly changing business environment. The issues could be divided into three categories regarding design issues, user knowledge, and organizational and cooperative issues. Below, the issues are summarized and listed under the respective category.

DESIGN ISSUES

1. Functionality for controlling and testing changes has to be integrated into the tailoring interface and there must be sufficient technical support for the end-user to estimate and check the correctness of the computation.
2. A tailorable system has to define a mental model that makes a clear division between definition, execution and tailoring. This mental model must be adopted in the tailoring interface and be shared by users, tailors and developers.
3. The tailoring interface also has to reveal potential for unanticipated use to the tailor. This means, that the information flow must, to a certain extent, exceed what is currently necessary.
4. The tailoring interface can be more complex, provided the tailoring process makes the deployment easier. The tailoring interface is not used as often as the deployment interface and additionally the tailoring itself often involves careful thought.
5. The deployment interface should be simpler than ordinary user interfaces.
6. The developer expanding the tailoring capability should only interact with one clearly defined point in the tailorable system, that is, changes are made at one point in the system.

End-User Knowledge

7. End-users must have sufficient knowledge of how the systems are structured and what the systems can contribute.
8. End-users must have solid knowledge of the nature of the task and what data is required to perform it.
9. End-users must have knowledge of which errors can occur and what the consequences of these may be.

Organizational and Cooperative Issues

10. System owners or developers must be responsible for making their systems publicly

available within the company. System owners or developers must also be responsible for updating the systems according to external requirements.

11. The necessity to extend the possibilities for end-users to manage the interaction in an evolving IT-infrastructure requires effective collaboration between the developer and end-users.

DISCUSSION

Our results are applicable in other areas that are similar to telecommunications, and that depend on an IT-infrastructure for a major part of their business and where the development of new products requires changes in this IT-infrastructure. On one hand, our results confirm existing research: Users ask for additional functionality to guide the tailoring and test the outcome (Burnett, Rothernel, & Cook, 2003). We found that users wished to incorporate control of the tailoring process in the form of an outline, preferably in a step-by-step fashion. They also asked for visualization and test facilities in order to check the impact of the separate steps on the end results. The evaluation of the interface allowing software engineers to expand the tailoring possibilities confirms and expands previous research results addressing the developer responsible for the evolution of tailorable systems as an additional stakeholder whose requirements also have to be considered (Eriksson et al., 2003; Lindeberg et al., 2002).

On the other hand, the results indicate that tailoring in an IT-infrastructure of networked applications provides additional challenges for the design of the software, the competence of the users and tailors, and the cooperation between users and developers. Changes—independently of whether they are implemented by tailoring or by evolving the software—can depend on and affect changes in other applications of the IT-infrastructure and the interaction between applications. This requires coordination between tailoring and development, and cooperation between the persons responsible for tailoring and developing the different applications. And this, in turn, requires a different set of competences from users and developers. The use of an application such as the prototype discussed here, for example, required knowledge of the surrounding systems and their data structures. Developers as well as users have to understand not only the system they are responsible for but also the dependencies between different systems and tasks. Several researchers have discussed collaboration between users and tailors, but not between users, tailors, and professional developers. For example Nardi and Miller's approach (1991) differs from ours in that they see local developers and programmers as being skilled users, while we take the concepts a step further and state that there is also a need for collaboration between users and professional developers who can perform programming tasks to extend the tailorable software beyond the script level.

What we claim is that in order to make tailoring really successful,it must be made possible for the tailorable system to evolve beyond the initial intention when building the tailorable system. Kahler's three levels (2001) of tailoring culture— cooperation between tailoring end-users, cooperation between tailors and users, and the organizational recognition and coordination of tailoring efforts—have to be extended with a fourth level, of organizational support for coordinating tailoring and development activities *involving the cooperation not only between users and tailors but also between tailors and software developers.*

CONCLUSION

Allowing end-users to tailor the interaction between flexible applications in an evolving IT-infrastructure requires that the tailoring activities are supported by the design of the system, for

example by providing a clear division between execution and tailoring, by revealing potential for unanticipated use, and by supporting single interfaces for changes to the software. It is also essential that the competence of the end-users is sufficient in terms of knowledge of how the systems are structured and what the systems can contribute. End-users must also have substantial knowledge of the task and which errors can occur and what the consequences of these may be. To allow end-users to tailor the interaction between applications in an evolving IT-infrastructure, the organization has to allow for cooperation between users and developers.

The main conclusion of the research described here is that it is possible to provide end-users with the possibility to tailor not only the applications, but if necessary also the interaction between different applications that are part of an IT-infrastructure. The evaluation clearly showed the dependencies between tailoring and the further development of the tailoring capabilities. The evaluation also made it apparent how the different actors were aware of their colleagues' skills and of what each individual could contribute. To ensure a sustainable tailorable system when deploying a system intended to evolve continuously through tailoring, it is necessary to take into account resources concerning various skills and collaboration between users and developers. Without smooth collaboration between the parties an extended fourth level of tailoring culture will not be provided for, and therefore the system will soon become partially obsolete and the competitive advantages provided by the system will decrease dramatically. The results challenge the clear division between software use and evolution on one side and software development on the other side, when developing and maintaining an IT-infrastructure. Collaboration between the end-user and the developer must work satisfactorily in order to achieve tailorable, sustainable software. In other words, in a rapidly changing business environment with continuously changing requirements, such as the one presented in this paper, the tailoring activities have to be coordinated with the software evolution activities.

ACKNOWLEDGMENT

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

REFERENCES

Bleek, W.G. (2004). *Software infrastruktur. Von analystischer Perspective zu konstruktiver Orientierung.* Hamburg: Hamburg University Press.

Blomberg, J., Suchman, L., & Trigg, R.H. (1996). Reflections on a work-oriented design project. *Human-Computer Interaction, 11*(3), 237-265.

Burnett, M., Rothermel, G., & Cook, C. (2003). Software engineering for end-user programmers. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'03)* (pp. 12-15).

Carter, K., & Henderson, A. (1999). Tailoring culture. In *Proceedings of the 13th Information Systems Research Seminar (IRIS'13)* (pp. 103-116).

Dittrich, Y., & Lindeberg, O. (2002). Designing for changing work and business practices. In N. Patel (Ed.), *Evolutionary and adaptive information systems* (pp. 152-171). Hershey, PA: Idea Group Publishing.

Ericsson, K.A., & Simon, H.A. (1993). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Eriksson, J., Warren, P., & Lindeberg, O. (2003). An adaptable architecture for continuous development - User perspectives reflected in the archi-

tecture. In *Proceedings of the 26th Information Systems Research Seminar (IRIS'26)*, Finland.

Eriksson, J. (2004). Can end-users manage system infrastructure? User-adaptable inter-application communication in a changing business environment. *WSEAS Transactions on Computers, 6*(3), 2021-2026.

Gantt, M., & Nardi, B.A. (1992). Gardeners and gurus: Patterns of cooperation among CAD users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 107-117).

Henderson, A., & Kyng, M. (1991). There's no place like home: Continuing design in use. In J. Greenbaum & M. Kyng (Eds.), *Design at work* (pp. 219-240). Hillsdale, NJ: Lawrence Erlbaum.

Hevner, A.R., March, S.T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75-105.

Kahler, H. (2001). *Supporting collaborative tailoring.* Roskilde: Roskilde University.

Lehman, M.M. (1980). Programs, life cycles, and laws of software evolution. In *Proceedings of the IEEE, 68*(9), 1060-1076.

Lindeberg, O., Eriksson, J. & Dittrich, Y. (2002). Using metaobject protocol to implement tailoring; Possibilities and problems. In *Proceedings of the 6th World Conference on Integrated Design & Process Technology (IDPT '02)*, Pasadena, California.

MacLean, A., Carter, K., Lövstrand, L., & Morgan, T. (1990). User-tailorable systems: Pressing the issues with buttons. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'90)* (pp. 175-182).

Mørch, A. (1997). Evolving a generic application into domain-oriented design environment. *Scandinavian Journal of Information System, 8*(2), 63-89.

Mørch, A. (2002). Evolutionary growth and control in user tailorable systems. In N. Patel (Ed.), *Evolutionary and adaptive information systems* (pp. 30-58). Hershey, PA: Idea Group Publishing.

Nardi, B.A., & Miller, J.R. (1991). Twinkling lights and nested loops: Distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies, 34*(1), 161-184.

Nunamaker, J., Chen, M., & Purdin, T. (1991). System development in information systems research. *Journal of Management Information Systems, 7*(3), 89-106.

Robson, C. (2002). *Real world research*. Oxford: Blackwell Publishers Ltd.

Stiemerling, O. (2000). *Component-based tailorability.* Bonn: Bonn University.

Stiemerling, O., & Cremers, A.B. (1998). Tailorable component architectures for CSCW-systems. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Programming* (pp. 302-308).

Trigg, R., & Bødker, S. (1994). From implementation to design: Tailoring and the emergence of systematization in CSCW. In *Proceedings of the Conference of Computer Supported Cooperative Work (CSCW 94)* (pp. 45-54).

Yin, R.K. (2003). *Case study research - Design and methods.* Thousand Oaks, CA: SAGE Publications.

This work was previously published in Journal of Organizational and End User Computing, Vol. 19, Issue 2 , edited by M. Mahmood, pp. 47-64, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 6.3
Differentiated Process Support for Large Software Projects

Alf Inge Wang
Norwegian University of Science and Technology, Norway

Carl-Fredrik Sørensen
Norwegian University of Science and Technology, Norway

ABSTRACT

This chapter presents a framework for differentiated process support in large software projects. Process support can be differentiated in different levels based on the size of the development organization and the need for coordination across different levels of the organization. We have defined four main perspectives: individual, group, team, and project level, where the framework consider essential issues when planning and executing the software development processes in organizations with different levels of management. Further, a guideline is provided that suggests what is required of process support in the various organizational levels.

INTRODUCTION

Development of large and complex software systems involves large organisations. In such working environments, it is essential to plan and coordinate the process, feed the involved developers with necessary documents, tools and files, track the process and effort, and learn from and improve the process.

Software process modeling is aimed at understanding, guiding, coordinating, automating, and improving the software process to thus improve the software quality and reduce the effort of developing software products (Wang, 2001). Many ***process models*** and ***process-centred support environments*** (PSEs) have been created with the assumption that the same process support should be provided at every level in an organization (Conradi, Fuggetta, & Jaccheri, 1998; Derniame, Baba, & Wastell, 1998; Finkelstein, 2000; Fuggetta, 2000; Nitto & Fuggetta, 1998).

If we consider development of large software systems, the organisations in such projects usually involve several levels of management. Depending on the level of an organisation a person is working in, the perspective and goal of the work will vary.

For a programmer, the main concern would be to have access to all necessary files, documents, and tools to carry out efficient programming. Personnel working at higher levels in the organisation would typically have other concerns like coordinating people, scheduling of the process, quality assurance, planning of activities and so forth. Thus, it is essential that the process support in such organisations reflects the levels being supported. It is also important that the way the processes are modeled is tailored for the organisational level and the characteristics of this level.

This chapter presents a differentiated process support framework that describes the elements required to model the software process, the required external resources (like tools and documents), and the required process support provided by a process-centred environment. Our framework describes the required process support from four perspectives: At the individual level, at the group level, at the team level, and at the project level. Thus, the objectives of this chapter is to give insights into essential issues to be considered when planning and executing a software development process for large software projects consisting of several levels of management. The chapter also provides a guideline for what is required of process support for the various levels of an organisation. This guideline can be used as input when evaluating tools to be used to support the development and management processes of large software projects.

BACKGROUND

This section gives an introduction to the background and the important terms used in our framework, and describes related work.

Software Process and Software Process Modeling

At a NATO Conference in Garmisch-Partenkirchen in 1968, the term ***software engineering*** was first introduced (Naur & Randell, 1969). The conference discussed what was called the "software crisis" introduced by third generation computer hardware. The work within the software engineering domain has been concerned with methods, techniques, tools, programming languages, and more to face the problems in software projects like delayed products, cost overruns, and bad reliability and performance in software products. Despite the many efforts to try to solve the software crisis, we are still struggling with the same problems as they did in the sixties. Brooks (1986) argues that there is no "silver bullet" that can solve all problems in software engineering. The only way to limit the negative effects identified as the software crisis is to use best practices in the many areas of software engineering. There are no best practices that solve all problems, but in combination many issues can be eliminated. One best practice is to improve the software process itself involving all the activities necessary in developing a software product. This chapter is intended as a guideline for improving the software process at different levels in a software development organisation.

Before we take a closer look at the software process support for different levels of an organisation, it is necessary to agree on the central terminology used in this chapter. As mentioned before, the term *software engineering* covers most aspects involved when developing large software systems. According to Sommerville (1995):

Software engineering is concerned with software systems which are built by teams rather than individual programmers, uses engineering principles in the development of these systems, and is made up of both technical and non-technical aspects.

As Sommerville states, software development involves more than the technical aspects like dealing with the source code of the system. Important nontechnical aspects of developing software involve scheduling, budgeting, resource

management, and so forth. In addition, the term *software process* (Lehman & Belady, 1985) is used to denote all activities performed within a software organisation.

In this perspective, *software process modeling* describes the activities of understanding and describing a given software process (Wang, 2001). Several elements have been proposed that need to be identified to describe a software process, but the most common model involves the elements *products* (a set of artefacts related to the software product), *resources* (assets, like tools or human resources needed to carry out the process), *activities* (steps of the process) and *directions* (policies, rules, and procedures that govern the activities) (Derniame et al., 1998). To be able to provide process support, the software process must be modeled. Software process modeling is, according to Høydalsvik (1997), defined as "*The activities performed, and language and tools applied, in order to create models of software development processes within a software development organisation.*"

Another term related to software process is ***workflow***. Workflow is by the Workflow Management Coalition (1999) defined as "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*"

The main difference between software process and workflow is that workflow focuses on business processes. However, this does not rule out the possibility to use workflow tools to support the software process.

When talking about software processes, we also often mention the term ***software lifecycle*** (Royce, 1987) that describes the life of a product from the initial stage to the final production and maintenance. A software lifecycle consists of several phases that a software product usually must go through: requirement specification, system design, implementation, testing and operation, and maintenance. During these phases, the required process support will vary. For the two first phases, the main focus is on analysis and documentation. For the three last phases, the main focus is on the source code and computer system. For all these phases, various tools exist to make the software development easier.

In addition, there are other parts of the software process that are phase independent, for example, software configuration management, software cost estimation, software quality management, software process improvement, and software process technology (Wang, 2001).

The lifecycle of a software product can be organised in various ways. The classical approach is the *Waterfall model* (Royce, 1987) that describes the lifecycle as a predefined sequence of phases, where each phase has a set of defined activities. Each phase has an input and output, and the output from one phase is the input to the next. The Waterfall process is in principle linear and a succeeding phase should not start until the previous phase has been finished.

The *evolutionary development model* is another way of viewing the software development process. This approach interleaves the phases; specification, implementation and validation, and is based on rapid prototyping. A prototype is developed early in the project in cooperation with the customer and is refined until the customer is satisfied. An example of an evolutionary development model is the *Spiral Model* (Boehm, 1988) that can incorporate different development processes through separate iterations.

Agile software development models like *eXtreme Programming* (Beck, 1999) focus on being lightweight and flexible. Such model embraces short development iterations to be able to cope with upcoming changes in requirements, minimum project documentation apart from comments in source code, user involvement during development, and other best-practices. Agile methodologies fit very well for high-risk projects with unstable requirements.

A software process model gives an overview of all the required activities and how they relate to each other, documents, tools, actors, methods, techniques, standards and so forth. This chapter is not limited to process support of a specific lifecycle model, but focuses rather on the required process support independent of how the software process is modeled and carried out in detail. Thus, this chapter describes guidelines that are important to incorporate when planning and supporting a software process. This may make it easier to improve the software process, thus improving the software product.

Related Work

The framework described in this chapter divides the software process into four distinct levels. The Personal Software Process (PSP) (Humphrey, 1997) defines the software process at the individual level in a software development process. The intention of PSP was to help software engineers to do their daily work well by applying practices that work well based on experiences from the software engineering field. Further, PSP describes detailed methods for making estimates and plans, shows how software engineers can track their performance, and describes how defined processes can guide their work. The process model in PSP is described only at a high-level. The model consists of the activities Planning, Design, Code, Compile, Test, and Post-Mortem. The PSP fits very well with the individual process level described in this chapter. However, note that PSP does not say anything explicitly about coordination of developers and project management on a higher organisational level.

The Software Process Engineering Metamodel (SPEM) defines software process models and their components using UML notation (OMG, 2002). In SPEM, a software development process is usually described by a *Process role* performing an *Activity* on a *Work product*. In addition, SPEM provides modeling elements to express phases, iterations, lifecycles, steps and so forth. This means that SPEM can visualise different organisational levels of a software process. Thus, SPEM can be used in combination with our framework to describe the software processes at different levels. However, SPEM was not made with software process enactment in mind. This means that to provide process support for a process specified using SPEM, the model has to be translated to an enactable process model language (PML).

UML4SPM is executable software PML based on UML 2.0 (Bendraou, Gervais, & Blanc, 2006). In UML4SPM PML, a software process can be represented at different hierarchical levels by using the main building blocks Process, Activity, and Action. The organisational structure can also be modeled in a hierarchical manner using the model building blocks Team and Agents. Although UML4SPM support modeling of a software process at different levels, it uses the same PML to represent all levels of an organisation. As the PML represents processes as activity-networks, collaborative activities beyond coordination are hard to represent in such PMLs (Wang, 2002).

Serendipity-II is a process management environment that supports distributed process modeling and enactment (Grundy, Apperley, Hosking, & Mugridge, 1998). The PML allows hierarchical modeling of software processes represented as process stages and subprocesses. Moreover, Serendipity-II is based on a decentralized distributed architecture that enables users to collaboratively edit process models synchronously and asynchronously. This approach enables support for autonomous suborganisations that can tailor their specific part of the process to their needs, which is essential to support different levels of a software organisation.

Kivisto (1999) describes the roles of developers as a part of a software process model for development of client-server process. Kivisto proposes a process model in three dimensions: Organisational, Technological, and Process. The organisational dimension focuses on what roles

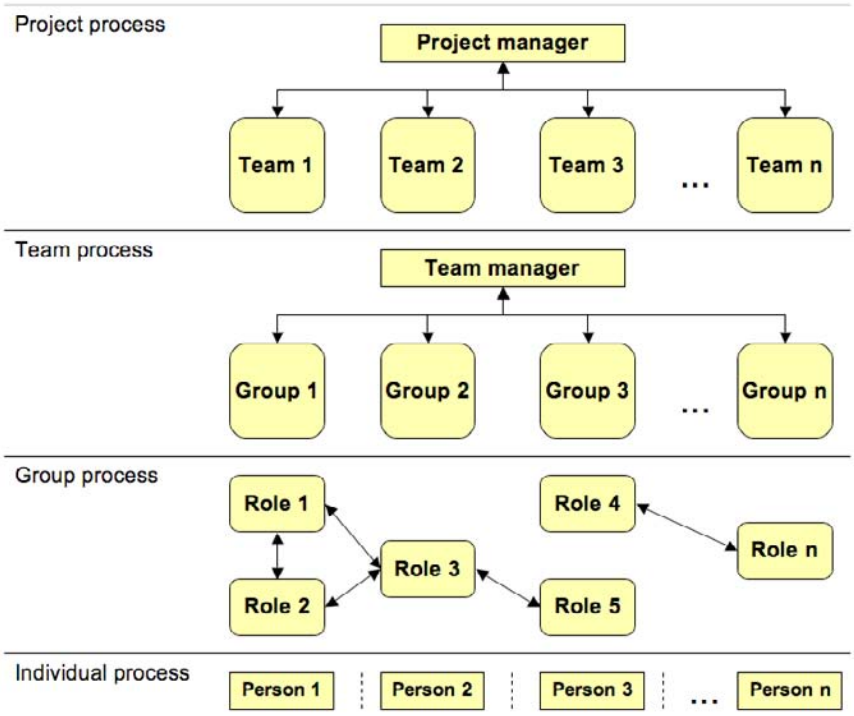
are involved in the development process and how these roles are organised. The technological dimension focuses on issues related to client-server software architecture and technology choices for such a system. The process dimension gives an overview of how the development process can be organised in phases and tasks, and the assignment of roles to specific tasks.

A taxonomy for characterising metaprocess categories with the main focus on process changes is presented in Nguyen and Conradi (1994). The taxonomy reflects the following questions to be asked concerning the metaprocess: *What* aspects of the software process are allowed to be changed, due to what reasons (*Why), How* can the changes be implemented/installed and be propagated (*When)* by which roles *(Whom)*? The *Why* aspect identifies four sources for process changes: the external world, an organisation, a project environment, and individuals. The framework described in this chapter can be mapped to the individual (individual level), project environment (group and team level), and organisation (project level). Changes in the process support will affect the processes at all levels in our framework, while modifications of the production process (the way the software is developed) would typically be mapped to the team level and below. Modifications of the metaprocess (changes and evolvement of the process model) would typically be mapped into the project level. The taxonomy described in Nguyen and Conradi (1994) is useful for analysing process changes at all four levels in our framework.

PROCESS SUPPORT IN DIFFERENT LEVELS IN A SOFTWARE DEVELOPMENT ORGANISATION

This section describes issues that must be considered when planning the process support for large software projects being organised in groups and teams (typically 20 or more people). In such software development organisations, the process

Figure 1. Overview of process support in different levels



support required is very dependent on where you are in the hierarchy in the organisation. For example, the software process support needs for an individual developer is totally different from the needs of a project manager.

Our framework as detailed below, describes the software process support from four perspectives based on the organisational level, as shown in Figure 1. The *first level* in the framework focuses on the *individual process* reflecting the personal software process in the daily work. The *second level* describes group processes where two or more individual collaborate and interact. The *third level* is concerned with management of groups within a team. Finally, the *fourth level* is concerned with project management where several teams are involved. Typically for smaller projects, the team and group levels will merge.

Table 1 shows the four levels of process support required in a big software organisation. For each level, the model describes:

- **Process elements:** The required process elements needed to model the process,
- **Process relations:** The required relationships between process elements,
- **Required external resources:** The resources used by the process-centred support environment (PSE), and
- **Required process support:** The process support required at a specific organisational level.

The framework specifies interfaces that describe the input and output between levels. The uppermost interface specifies the relationships to the company and customer(s). Some activities are carried out at all levels of an organisation, for example, quality assurance and configuration management.

The rest of this section will describe each level of the framework in more detail.

Individual Process

The lowest process level in an organisation is the *individual* process level focusing on the individual tasks of a software developer. An example of a role related to this level is a programmer with typical tasks like reading documents, writing code, writing documentation, debugging code, compiling code, building code, and so forth.

Table 1. Process support in an organisation

Org. level	Process elements	Process relations	Required external resources	Required process support
	Interface: Product, Experiences and knowledge, Resources, Project management rules and practice			
Project process	Teams, Team processes, Milestones	Team coordination, Assigned team(s)	Project management tools and Artefacts, Experience-/ knowledgebase, Estimation models and tools, Resource tools	Project planning, Project management, Experience exchange, Project estimation, Resource management
	Interface: Input and output artefacts, Team and resources assigned, Team process state			
Team process	Groups, Group processes	Group coordination, Assigned group(s)	Team management tools and Artefacts	Team management
	Interface: Input and output artefacts, Group assigned, Group process state			
Group process	Roles, Process fragments, Cooperative activities	Cooperation and coordination rules, Assigned role(s)	Cooperative tools, Configuration management and Artefacts	Coordination, Negotiation, Collaboration
	Interface: Input and output artefacts, Role assigned, Process fragment state			
Individual process	Activities	Assigned activities	Tools and Artefacts	Process guidance, automation, calendar

Generally, the process of an individual actor typically consists of a set of activities related to the roles that the actor plays (an actor can play more than one role). It is very important that the process support is adaptable and configurable, making it possible for the actor to fit the work to personal preferences. The required process support is very dependent on the experience level of the individual actor. An inexperienced person would typically need process guidance to show the tasks to do next, what procedures to follow, what tools to use, where to find documentation and so forth. However, for an experienced person that knows all the basics, extensive process guidance would be a hindrance to work effectively. For the latter, it would be more useful to provide automation of repetitive tasks. Independently of the experience level of the actors, the process support should provide quick and easy access to the required tools and artefacts. The activities of the individual process should be integrated with personal digital calendars to enable activity states and deadlines to be accessible on personal computers, personal data assistants (PDAs), and mobile phones. This means that the execution environment for the process support also must be capable of communicating and utilising mobile devices and networks.

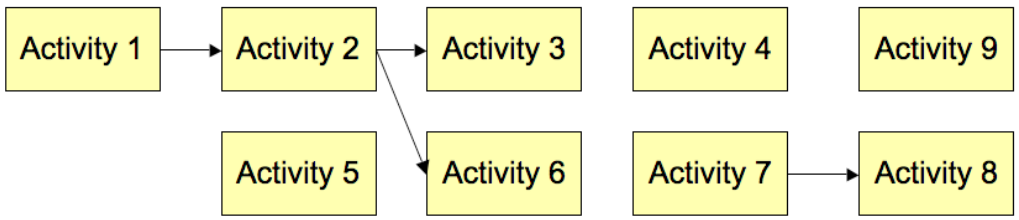
The framework presented in Table 1 identified activity as the main building block for individual process models. Another approach could be to use roles with responsibilities as the main building block. However, the most common approach is that individuals represent the daily work as a set of activities. Also, the most used process tools for individual processes also focus on activities, like calendar software. In addition, activities or tasks are the most common output of many project-planning tools, where individuals are assigned a set of activities. This means that using activities as process model elements makes it easier to integrate with higher-level management processes and tools. In addition, because people are used to the representation of work as a set of activities and tools that represent the work as activities, it is also easier for the individuals to tailor their own processes (reschedule activities, etc.). In some cases, it can be useful to let an actor model her or his own processes. In Wang (2002), an experiment indicates that it is easier to use activity-based than role-based modeling for individual processes. This is especially true for inexperienced process modelers.

The process relation with respect to individual processes is the assign relation. A team or project manager typically assigns activities to an individual actor. In some cases, some activities must be carried out in a specific sequence, putting constraints on how the work should be carried out and how the individual process can be modified. A representation of an individual process is illustrated in Figure 2.

The process model in Figure 2 shows a collection of activities. Some of the activities have pre-order relations (the activities 1, 2, 3, 6, 7, and 8). The remaining activities have no constraints and can be carried out regardless of the state of the process.

In this chapter, we define a collection of related activities (e.g., activities related to the same

Figure 2. An individual process model



phase in the process and the same project) to be a *process fragment* (PF). The name process fragment indicates that this collection of activities is part of a larger process at a higher level of the organisation. The process of assigning activities to an individual actor in the organisation is carried out by delegating one or more process fragments to the actor. The state of the process fragment is determined by looking at the state of the activities. A process fragment is completed when all actvities in the fragment are completed.

The goal of the individual process is to produce various artefacts like requirement and design documents, source files, build files, test files and so forth. To provide a sufficient software process support at this level of the organisation, it is essential that the PSE can be integrated with production tools such as office applications, integrated development environments (IDEs), and other similar tools. The process support environment should make it easy to access all necessary tools and files, and handle configuration management.

When the individual actor initialises her/his process, the process models could either be fully specified by, for example, a team or project manager or be provided as a process template where all necessary activities are defined. For the latter, the actor must detail her/his own process to meet her/his needs and preferences. During the process enactment, the individual actor might also add, change, or delete activities depending on how the process proceeds (e.g., change of project plan, changed priorities, or changed process goals). Some PSEs might provide support for changing the process automatically or semi-automatically. It is important that the PSE is flexible enough to managing instantiation of template process fragments as well as allowing changes of the process during process enactment (Grundy et al., 1998).

Group Process

The next level in the framework is the *group* process level focusing at the cooperative aspects of the software process. Traditionally, software process modeling and enactment of software processes have been focusing on ”forcing” and guiding people to work according to a specified model, where interaction between people has been coordinated through a strictly defined control/data flow. Such approaches are usually based on modeling processes as activity-networks consisting of subprocesses, activities, tasks, or operations. Cooperative aspects of the software development process have often been either eliminated or ignored because it has been hard to model cooperative activities in existing systems, or there has not been an interest for doing so. The software development processes are also human-centred processes. Cugola and Ghezzi (1998) state that ”Human-centred processes are characterised by two crucial aspects that were largely ignored by most software process research: They must support cooperation among people, and they must be highly flexible.”

The cooperative process involves coordination of activities carried out by several actors or roles, cooperative activities where two or more persons must participate to complete the activity, and coordination and negotiation of artefacts and resources.

At this level, ***configuration management*** (CM) is very important because two or more actors often will share the same resources (including files). The sharing of resources might cause conflicts where two or more actors would like to update the same data at the same time. Thus, the CM environment must be integrated with the PSE to provide support for initiating negotiation processes in case of conflicts regarding resources (e.g., files), and support for synchronising output artefacts from the process (Wang, 2000; Wang, Larsen, Conradi, & Munch, 1998).

The group process defines the synchronisation points for individual processes involving several actors. These synchronisation points represent parts of the software process where files or other resources need to be exchanged and synchronised

(mainly coordination of artefact and process). In addition, the group process involves cooperative activities where two or more actors are involved. An example of such activities can be distributed brainstorming, electronic voting, collaborative authoring, and conflict management. Cooperative activities have very different characteristics compared to individual activities. While the main emphasis of the individual activities is on the activities themselves, cooperative activities are all about interaction and coordination between roles. This means that the process support for cooperative activities must provide an infrastructure to enable efficient interaction of the involved roles and to enable flexible exchange of artefacts. Note that most people working in a large software project will be involved in both individual and group processes.

The cooperative processes can be represented in many ways. However, to represent cooperative processes as a set of related activities, as described in a previous section, does not make sense because the focus is on interaction between roles. We propose two alternative approaches to represent cooperative processes at the group level.

Cooperative Software Agents

Cooperative software agents can be used to represent actors involved in cooperative activities. The cooperative agents will act on behalf of the involved actors to provide the required infrastructure for collaboration (coordination, negotiation, etc.). In Wang, Conradi, and Liu (2000), a cooperative agent framework developed to provide process support is presented. In this framework, all actors have their own workspace where they can interact and configure their agents. Agents interact on behalf of various actors in agent meeting places that represent neutral workspaces where services and artefacts can be exchanged. The collaborative aspects are supported through coordination agents, negotiation agents, and mediation agents. When software agents are used, the agent environment usually provides a set of ready-to-use agents that provide support for various collaborative tasks and that can be configured for specific user needs. If the required functionality is not supported by the predefined agents, new agents can be implemented using a high-level API. A similar approach to support group processes using software agents is described in Glaser and Derniame (1998). This approach uses agents to look for competence profiles that match the tasks to be carried out.

Role-Based Process Environments

Role-based process environments can also be used very efficiently to model and provide support for cooperative activities, for example, like the workflow system ProcessWeb (Yeomans, 1996). In ProcessWeb, all involved in the process are modeled as *roles*, and *interactions* are used to provide communication channels between roles. A role is defined by its *actions* (methods) and its *resources* (attributes). Preconditions called *"when guards"* are used to select action for enactment, and they are expressed as **if** statements. Interactions are unidirectional, asynchronously typed communication channels provided through a takeport and a giveport. A *takeport* receives data or control flow from another role, and a *giveport* sends data or control flow to another role. Similar approaches can be found in Fadlia, Said, and Nora (2005). Most role-based process modeling approaches are object-oriented and the modeling process will be similar to programming in an object-oriented programming language. This gives a very flexible and expressive PML, but it requires a certain level of expertise to handle.

A comparison of how well cooperative activities can be modeled and supported in software agents, activity networks, and role-based process environments is described in Wang (2002). The results show that activity networks are not well suited, but software agents and role-based process environments are able to represent cooperative activities very well.

In the previous section, the term process fragment was used to denote a collection of activities that represents the process for an individual actor. To model group processes, it is necessary to model the relationships between the process fragments of the individual actors as well as cooperative activities and the roles involved. *Cooperative rules* (Wang et al., 2000) can be used to define the relationships between process fragments and cooperative activities. The cooperative rules specify conditions for cooperative activities that should be initiated depending on the condition of the individual process fragments. The cooperative rules also specify how the individual processes should proceed depending on the outcome of a cooperative activity. Thus, the cooperative rules specify a loose coupling between the individual process fragments and the cooperative activities enabling federation of different PSEs for individual and group processes.

Figure 3 illustrates a group process model consisting of roles, process fragments, a cooperative activity, and cooperative rules (C1-C6). The model describes how cooperative rules describe the relationships between process fragments (individual processes) for each role and the cooperative activity. The cooperative rules are modeled as result-reaction pairs that specify the *reaction* that should be taken based on the *result* (outcome) of a cooperative activity or a process fragment. Typical reactions can be to execute, regroup, halt, change, or add a process fragment. In addition, a reaction can initiate or change cooperative activities (e.g., initiate a specified agent) or change the cooperative rules (reflective).

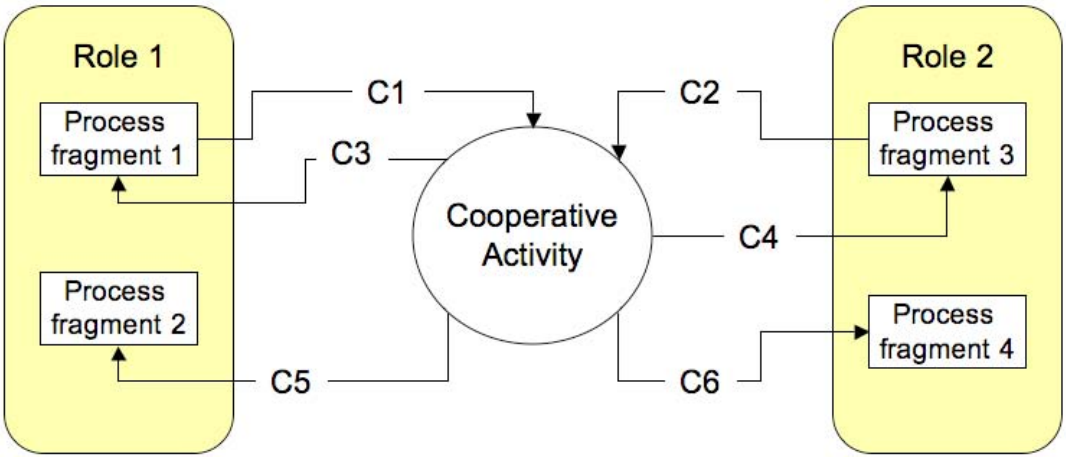
The cooperative rules in the process model in Figure 3 specifies that *Process fragment 1* and *3* must be completed before the cooperative activity will be executed (C1 and C2). Depending on the outcome of the cooperative activity, either *Process fragment 1* and *3* will be re-enacted (C3 and C4), or *Process fragment 2* and *4* will be initiated (C5 and C6).

In addition to provide support for cooperative activities in a PSE, it is also important for the PSE to provide access to other collaborative tools like chat-applications, collaborative editors, Wikis, discussion forums and so forth. The output of group processes is artefacts produced by the group and the individuals.

Team Process

The *team* process level is related to middle management and specific roles in a software project like a team manager. The team manager manages

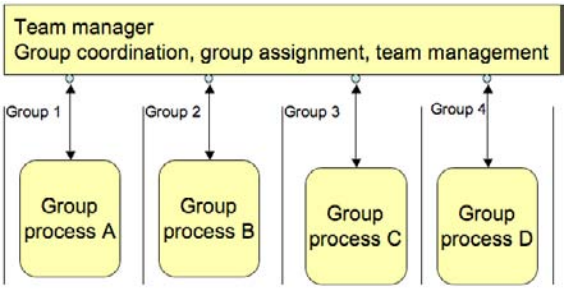
Figure 3. Group process model



the development of one or more main parts of a software system within the competence of the team (e.g., security, business logic, etc.). Typical tasks at this level can be planning of group processes, assignment of group process to groups, tracking progress of the team, monitoring resource usage, and low-level quality assurance. The process support required at this level is monitoring of the ongoing processes, computation of resource usage, prediction and estimation of progress and resource usage, and creation and editing of process models. It is also essential that the team process model can integrate the group and individual process models and provide a process environment that will exchange artefacts and state changes between the individual and group processes. The main difference between the coordination and cooperation at the group and team levels is the granularity. At the group level, the cooperation between group members is tight and frequent. At the team level, the groups are independent but the coordination is still important to, for example, manage available resources and exchange project artefacts (document, source code, programs, tools, etc.). It is the team manager who is responsible for the coordination of groups. Thus, it is important that the process support provides the infrastructure and tool to support and ease this job.

Figure 4 illustrates a process model at the team level that consists of team process coordination and assigning group processes in addition to other team management activities. The group processes are coordinated through defined interfaces provided by the PSE (illustrated by the small circles). Another alternative could be to allow direct coordination between groups. This approach might cause problems because two levels of coordination must then take place at the group level: coordination of individuals and coordination of groups. Another advantage by coordination groups through a team manager is that it is easier to make correct decisions about changes of the process, as this level gives a better overview of the process and the state of the different parts of the process.

Figure 4. Team process model



The output from the team process level is the artefacts from all groups as well as the state of the team process.

Project Process

The *project* process level represents the software development process at the highest abstraction level, usually managed by a project manager. For this role, the required process support involves tools and procedures to create a project plan (project process model) for the whole project, establish and change teams and groups, assign personnel to teams and groups, estimate project resource usage, track progress of the whole project, monitor resource usage, perform company quality assurance procedures, assess and improve the development process, and so forth.

At this level it is not unusual to have external subcontractors to carry out parts of the project that require special expertise or expertise that is not present in the company. From a process support point of view, it is also likely that external subcontractors use their own process models and support environment. Thus, it is essential for the process-centred support environment to enable federation of various process model languages and tools to enable monitoring and process enactment of the whole project.

Another essential activity at the project level is to package artefacts, process models, and knowledge and experiences from completed projects that can be used as input when planning,

estimating, modeling, and managing new projects. One method for harvesting experiences from completed projects is to use postmortem analysis methods (Birk, Dingsoyr, & Stalhane, 2002) like Affinity Diagram (KJ diagrams) (Scupin, 1997) and Root Cause Analysis (Straker, 1995). Affinity diagrams provide a method for carrying out a structured brainstorming focusing on successes and problems in the project. The root cause analysis investigates one particular issue found in the brainstorming process (positive or negative) to discover the causes and subcauses of the success or problem. In addition, reusable code and documents need to be identified and described with meta-information to enable search on historical information. A knowledgebase or experience-base is the most common tool to store and manage project experiences and reusable artefacts (Dingsøyr & Røyrvik, 2003).

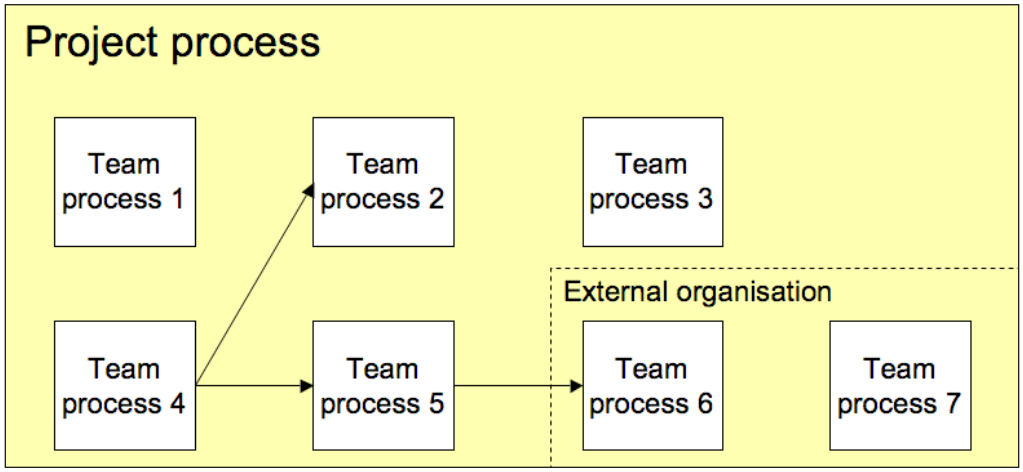
Figure 5 illustrates a process model at the project level. A project process consists of several team processes. Some of these team processes must be carried out in a specific order (e.g., Team process 4, 2, 5, and 6). The pre-order constraints are most often caused by one team process that requires a finished product from another team process before it can start. Note also that parts of the process might be carried out by external organisations. This might implicate that this part of the process model will be expressed in a different process modeling language compared to the in-company models. Also, this part of the process model might need integration with another PSE.

The output of the project process is the final software product along with experiences and knowledge acquired during the project. The completion of a project will also release resources that can be used in new projects. The input to the project process should also be experiences and knowledge from previous projects. Further, a new project is allocated resources like people, tools, equipment, and money. Other important inputs to the project process are company policies and project management rules and practices that must be followed. These constraints will influence the way the process is organised and how the software process will be modeled.

Organisational Issues

This section has so far only briefly mentioned the organisational issues that have to be taken into consideration when planning multilevel process support. This section will briefly look at organisational issues that must be taken into account.

Figure 5. Project process model



At all levels of an organisation, people involved play different roles related to the activities they are involved in. What roles people play varies from project to project. The role also depends on the domain of the project, the company culture, and the culture of the nationalities of the people employed in the company. Kivisto (1999) describes the roles involved in the development of client-server applications to be: application developer, business object developer, database developer, end-user, team leader, project manager, quality assurancer, tester, and expert. Further, these roles are mapped into a hierarchy where the project manager is responsible for managing team leaders, and team leaders are responsible for managing the rest of the roles. This organisational decomposition fits well with the two top levels and the bottom level of our framework. For a specific project, the roles involved must be described and people that fit these roles must be found. An employee profile database can be used for this purpose to find the correct person for a specified role.

McGrath (1998) has proposed a software process modeling framework that incorporates the behavioural issues of software development. The framework has three levels: the universe of discourse (level 1), the conceptual model (level 2) and the external models (level 3). The framework includes behaviour aspects such as attitudes, opinions, beliefs, knowledge, organisational power, politics, culture, and structure. These important issues are not covered in the approach presented in this chapter.

CREATION OF A MULTILEVEL PROCESS MODEL

We propose two main approaches to create a multilevel process model for a software project: top-down and bottom-up. These two approaches are presented in this section.

Top-Down Approach

In the top-down approach, a project manager will typically first create a high level process model based on experiences from previous projects found in the knowledgebase, and the present information (constraints and resources) about the new project. This process model should at least identify the processes at the team process level. Further, the model might contain representations of group and individual processes at a coarse level. It is important that the individual and group processes are not constrained in such a way that they cannot be tailored by the process participants carrying out the processes. If the group and individual processes are modeled by a project manager (at the project level), then the process model should be described by a process template rather than an instantiated process model. If the project manager has not modeled any group or individual processes, the team manager must identify and model individual and group processes. It is also essential that the individual activities are not modeled in such a way that they cannot be altered or tailored by the process participants at the individual process level. The team manager should include sufficient content to the individual process level like activity description, necessary documents and tools, and a proposed sequence of the activities. In some cases the sequence of activities must be frozen because of coordination and synchronisation between developers. The top-down approach is suitable when the top management has a good idea of how the software is being developed in the company or the company management wants to enforce a specific way of developing software motivated by software improvement initiatives (e.g. CMM). For the latter, it is important that the project manager has a good dialogue with the people working in the teams to avoid sabotage of the process. If a process is enforced on the individual actors, they need to be motivated and know why they have to

do their work in a particular way (e.g., because of quality or economical reasons).

Bottom-Up Approach

The button-up approach aims to harvest the process models by starting at the individual processes and moving upwards in the hierarchy of the software organisation. This means that the team process models will be created based on the underlying group and individual process models, and that the project process model will be created based on the underlying team process models. If no processes or process models are described or defined in the company, the bottom-up approach can be applied to create a process model that represents how the organisation is currently working. This process model can be used as a starting point to improve the process by identifying parts of the process that are cumbersome or uses unnecessary resources. However, it is likely that the process needs to be simplified to be able to model it and provide process support for it.

What Approach to Choose?

There is no clear answer if a software organisation should use the top-down or the bottom-up approach. The choice depends on the strategic goals of the organisation and on the working routines used. If the current work processes are causing several problems, a top-down approach inspired by established software development practices could be used to establish a new and more efficient way of working. If the current work processes are efficient and the employees are pleased with the way they work, a bottom-up approach should be chosen. Many organisations are in between these two extremes. If this is the case, a combination of top-down and bottom-up can be used. To succeed in combining the two approaches, it is necessary for the different levels of the organisation to interact often and synchronise the process models to make sure they fit. This chapter can be used as a checklist of what should be considered at each level of the organisation. In addition, the chapter identifies the interfaces between the organisational levels, making it possible to provide one process model for the whole project consisting of autonomous parts.

CONCLUSION

This chapter has presented a framework that describes process support in four levels in a large software development organisation.

The *first level* is the individual level where the main focus is on the activities of individual developers and the process support they require. At this level it is important that the process support can be integrated with calendar applications running on personal computers, personal data assistants, or mobile phones. In addition, it is essential that the process support provides easy access to the required files and tools.

The *second level* is the group level where actors involved in the development process need to collaborate, and coordinate and negotiate shared resources. The process model at this level focuses on cooperative activities, cooperative rules, and the process fragments of the individual participants. The cooperative activities can be modeled and supported using software cooperative agents or role models. Because files are likely to be shared at this level, configuration management is essential. The configuration management system should be integrated to the process-centred support environment to provide process support for negotiation for shared resources.

The *third level* is the team level where the team manager coordinates the various groups in the team. The main focus at this level is team management that involves assigning work to groups, and provides the necessary infrastructure and support for intergroup coordination.

The *forth level* is the project level where the project manager initiates, manages, and finalises

projects. At the end of a project, it is important that experiences, knowledge and reusable artefacts are packaged in a knowledgebase for later usage. In the initialisation phase of a new project, the knowledgebase plays a vital role when making resource estimates, modeling the process, assigning resources, and finding the reusable software components. It is also important at this level to make a heterogeneous PSE able to interact with PSEs in external organisations, as some team processes might be carried out by external resources. Finally, support for monitoring the process consisting of several teams that also might be external to the company is essential.

The main contribution of this chapter is a description of the process model elements required in a process model at each level of a software development organisation, and a description and discussion about the required process support at these levels.

FUTURE RESEARCH DIRECTIONS

Professional industrial software development organisations are always looking for ways to improve their development and project processes. One way of improving the software process is to provide the appropriate support for the process at different levels of the organisation. Most research within software process modeling assumes software development made by one company with a stable organisation and where the requirements are stable. This is not always the case.

The success of large open source projects have made open source development practices interesting to also apply within larger software organisations that perform global software development.

The ITEA COSI project (Codevelopment using inner & Open source in Software Intensive products, http://itea-cosi.org/) has observed a shift in software development, to more cooperation, new coalitions, and more collaboration with open source communities. These forms of network-enabled collaborations create new challenges for software developers (Lacotte, 2004). These new forms of development make the normal development processes less scalable and flexible because more and more of the software is harvested through reuse of internal components, through COTS, or through open source projects. In addition, a lot of the actual development efforts are outsourced, making it harder to ensure quality, productivity, and control of the software project(s).

The percentage of code actually produced locally in an organisation is decreasing, while the complexity and size of the products are still increasing. Large software companies like IBM and Sun Microsystems have made parts of their product portfolio as open source, creating precedence for how to make and publish software.

Participants in open source projects are increasingly employed by industrial companies. Many of these participants are very flexible with respect to how software is developed, and will often adapt the development practices used in the open source project.

When considering software process support for open source processes, the most important issues that must be considered are collaboration and coordination between many distributed developers and flexible and open PSEs that can be integrated with a large set of tools.

Agile methods (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003) have gained much attention in recent years by moving the attention from documents and specifications to customer participation and incremental and iterative development of the products. Smaller projects are more likely to adopt agile methods, while large projects still will employ project management practices that are more heavyweight. To use agile methods in larger organisation, the main problem is to manage multiple cooperating teams that facilitate various overlapping, informal cross-team communication. Kahkonen (2004) describes an approach for introducing agile methods in large

organisations. This approach was tested by Nokia with success, where the informal cross-team communication problem was solved by using communities of practices theory.

Our proposed framework is embracing these new trends by differentiating the process model and support for different levels of organisation.

REFERENCES

Abrahamsson, P., Warsta , J., Siponen , M. T., & Ronkainen, J. (2003), New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03),* Portland, OR, USA.

Beck, K. (1999). *Extreme programming explained: Embrace change*. Reading, MA: Addison-Wesley.

Bendraou, R., Gervais, M.-P., & Blanc, X. (2006). UML4SPM: An executable software process modeling language providing high-level abstracts. In *Proceedings of the10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06),* Hong Kong, China.

Birk, A., Dingsøyr, T., & Stålhane, T. (2002, May-June). Postmortem: Never leave a project without it. *IEEE Software, 19*(3), 43-45.

Boehm , B. W. (1988, May). A spiral model of software development and enhancement. *IEEE Computer.*

Brooks, F. P. (1986). No silver bullet: Essence and accidents of software engineering. In *Proceedings of Information Processing'86.* North-Holland, IFIP.

Conradi, R., Fuggetta, A., & Jaccheri, M. L. (1998). Six theses on software process research. In *Proceedings of Software Process Technology: 6th European Workshop (EWSPT'98*), Weybridge. Springer-Verlag. LNCS 1487.

Cugola, G., & Ghezzi, C. (1998). Software processes: A retrospective and a path to the future. *SOFTWARE PROCESS—Improvement and Practice, 4*(2), 101-123.

Derniame, J.-C., Baba, B.A., & Wastell, D. (Eds.). (1998). *Software process: Principles, methodology, and technology.* Berlin: Springer-Verlag LNCS 1500.

Dingsøyr, T., & Røyrvik, E. (2003). An empirical study of an informal knowledge repository in a medium-sized software consulting company. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03),* Portland, OR, USA.

Fadila, A., Said, G., & Nora, B. (2005). Software process modeling using role and coordination. *Journal of Computer Science, 2*(4).

Finkelstein , A. (Ed.). (2000). The future of software engineering. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'2000),* Limerick, Ireland.

Fuggetta, A. (2000). *Software process: A roadmap.* In *Proceedings of the Conference on the Future of Software Engineering (ICSE 2000)* (pp. 25-34). Limerick, Ireland.

Glaser, N., & Derniame , J.-C. (1998). Software agents: Process models and user profiles in distributed software development. In *Proceedings of the 7th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises* (WETICE'98), Palo Alto, CA, USA.

Grundy, J. C., Apperley, M. D., Hosking, J. G., & Mugridge, W. B. (1998, September-October). A decentralized architecture for software process modeling and enactment. *IEEE Internet Computing, 2*(5), 53-62.

Høydalsvik, G. M. (1997). *Experiences in software process modeling and enactment.* Doctoral thesis, Department of Computer and Information

Science, Norwegian University of Science and Technology, Trondheim, Norway.

Humphrey, W. S. (1997). *Introduction to the personal software process.* Information Technology for European Advancement. Addison-Wesley.

Kahkonen, T. (2004). Agile methods for large organizations—building communities of practice. *Agile Development Conference (ADC'04*), Salt Lake City, Utah, USA.

Kivisto, K. (1999). Roles of developers as part of a software process model. In *Proceedings of the 32nd Hawaii International Conference on System Sciences.*

Lacotte, J.-P. (2004). *ITEA report on open source software* (Tech. Rep.). ITEA -

Lehman, M. M., & Belady, L. A. (1985). *Program evolution—processes of software change*. Academic Press.

McGrath, G. M. (1998, January). Behavioural issues in software engineering process modelling: A multi-paradigm approach. *Hawaii International Conference on System Sciences (HICSS).*

Naur, P., & Randell, B. (Eds.). (1969). Software engineering. In *Proceedings of the NATO Conference in Garmisch-Partenkirchen, 1968*. NATO Science Committee, Scientific Affairs Division, NATO, Brussels.

Nguyen, M. N., & Conradi, R. (1994). Classification of meta-processes and their models. In *Proceedings of the Third International Conference on Software Process*, Washington, USA.

Nitto, E. D., & Fuggetta, A. (Eds.). (1998). Process technology. *Journal on Automated Software Engineering, 5*(Special Issue).

OMG. (2002). *Software process engineering metamodel specification*. Formal/2002-11-14.

Royce, W. W. (1987). Managing the development of large software systems: Concept and techniques. In *Proceedings of WesCon, 1970*. Reprinted in Proceedings of the International Conference on Software Engineering. IEEE Computer Society Press.

Scupin, R. (1997). The KJ method: A technique for analyzing data derived from Japanese ethnology. *Human Organization, 56*(2), 33-237.

Sommerville, I. (1995). *Software engineering.* Addison-Wesley. ISBN 0-2014-2765-6.

Straker, D. (1995). *A toolbook for quality improvement and problem solving.* Prentice Hall International (UK).

Wang, A. I. (2000). Using software agents to support evolution of distributed workflow models. In *Proceedings of the International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000) at International ICSC Congress on Intelligent Systems and Applications (ISA'2000),* Wollongong, Australia.

Wang, A. I. (2001). *Using a mobile, agent-based environment to support cooperative software processes.* Doctoral thesis, Norwegian University of Science and Technology. ISBN 82-7984-172-5.

Wang, A. I. (2002). An evaluation of a cooperative process support environment. In *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA2002),* Cambridge, MA, USA.

Wang , A. I., Conradi, R., & Liu, C. (2000). Integrating workflow with interacting agents to support cooperative software engineering. In *Proceedings of the 4th IASTED International Conference on Software Engineering and Applications (SEA'2000),* Las Vegas, NV, USA.

Wang, A. I., Larsen, J.-O., Conradi, R., & Munch, B. (1998). Improving cooperation support in the EPOS CM System. In *Proceedings of the 6th European Workshop on Software Process Technology (EWSPT'98),* Weybridge (London), UK.

WfMC. (1999, February). *Workflow management coalition—terminology & glossary* (Tech. Rep. No. WFMC-TC-1011). The Workflow Management Coalition. Retrieved March 6, 2008, from http://www.wfmc.org/standards/docs/TC1011_term_glossary_v3.pdf

Yeomans, B. (1996). *Enhancing the World Wide Web* (Tech. Rep.). Computer Science Department, University of Manchester.

ADDITIONAL READING

The following is a list of references recommended for further reading.

Ambriola, V., Conradi, R., & Fuggetta, A. (1997, July). Assessing process-centered software engineering environments. *ACM Transactions on Software Engineering and Methodology (TOSEM), 6*(3), 283-328.

Andersson, C., Karlsson, L., Nedstam, J., Höst, M., & Nilsson, B. (2002). Understanding software processes through system dynamics simulation: A case study. In *Proceedings of the 9th IEEE Conference and Workshop on the Engineering of Computer-based Systems*, Lund, SWEDEN.

Barros, M. O., Werner, C. M. L., & Travassos, G. H. (2000, October). Using process modeling and dynamic simulation to support software process quality management. *XIV Simpósio Brasileiro de Engenharia de Software, Workshop de Qualidade de Software*, João Pessoa, Brazil.

Chatters, B. W., Lehman, M. M., Ramil, J. F., & Wernick, P. (1999, June 28-30). Modelling a software evolution process. In *Proceedings of the Software Process Modelling and Simulation Workshop*, Silver Falls, OR.

Collofello, J., Yang, Z., Merrill, D., Rus, I., & Tvedt, J. D. (1996). Modeling software testing processes. In *Proceedings of the International Phoenix Conference on Computers and Communications* (IPCCC'96), 1996.

Dean, D. L. , Lee, J. D. , Orwig, R. E., & Vogel, D. R. (1994, December). Technological support for group process modelling. *Journal of Management Information Systems, 11*(3), 43-63.

Deephouse, C., Mukhopadhyay, T., Goldenson, D. R., & Kellner, M. I. (1995, December). Software processes and project performance. *Journal of Management Information Systems, 12*(3), 187-205.

Delen, D., Dalal, N. P., & Benjamin, P. C. (2005, April). Integrated modeling: The key to holistic understanding of the enterprise. *Communications of the ACM, 48*(4), 107-112.

Estublier, J., Amiour, M., & Dami, S. (1999, March). Building a federation of process support systems. In *ACM SIGSOFT Software Engineering Notes, 24*(2), Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration WACC '99.

Gary, K., Lindquist, T., Koehnemann, H., & Sauer, L. (1997, November 16-19). Automated process support for organizational and personal processes. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge*, (pp. 221-230), Phoenix, AZ, USA.

Glass, R. (1999, February). The realities of software technology payoffs. *Communications of the ACM, 42*(2), 74-79.

Gopal, A., Mukhopadhyay, T., & Krishnan, M. S. (2002, April). The role of software processes and communication in offshore software development. *Communications of the ACM, 45*(4).

Gruhn, V., & Wolf, S. (1995). Software process improvement by business process orientation. *Software Process—improvement and Practice* (Pilot Issue).

Kahen, G., Lehman, M. M., & Ramil, J. F. (1999, September 3-4). Empirical studies of the global software process—the impact of feedback. In

Proceedings of the Workshop on Empirical Studies of Software Maintenance (WESS-99), Keble College, Oxford, UK.

Keating, E., Oliva, R., Repenning, N., Rockart, S., & Sterman, J. (1999). Overcoming the improvement paradox. *European Management Journal, 17(*2), 120-134.

Kellner, M. I., Madachy, R. J., & Raffo, D. M. (1999, April 15). Software process modeling and simulation: Why, what, how. *Journal of Systems and Software, 46*(2/3), 91-105.

Kueng, P., & Kawalek, P. (1997, August). Process models: A help or a burden? In *Paper presented at the Association for Information Systems 1997 Americas Conference*, Indianapolis, IN.

Lehman, M. M. (1997, April 14-15). Feedback in the software process. In *Proceedings of the Software Engineering Association Easter Workshop, SEA'97, ICSTM*, (pp. 43-49).

Lehman, M. M. (1998, April 20). Feedback, evolution and software technology—the human dimension. In *Proceedings of the ICSE Workshop on Human Dimension in Successful Software Development*, Kyoto, Japan.

Lin, C. Y., & Levary, R. R. (1989). Computer-aided software development process design. *IEEE Transactions on Software Engineering, 15*(9), 1025-1037.

Madachy, R., & Tarbet, D. (2000). Case studies in software process modeling with system dynamics. *Software Process: Improvement and Practice, 5*(2-3), 133-146.

Martin, R. H., & Raffo, D. M. (2000). A model of the software development process using both continuous and discrete models. *Software Process: Improvement and Practice, 5*(2-3), 147-157.

McGrath, G. M. (1996, August 19-21). Representing organisation and management theories in software engineering process modelling. In *Proceedings of the IASTED Conference*, Honolulu, HI.

McGrath, G. M. (1997, September 28-October 2). A process modelling framework: Capturing key aspects of organisational behaviour. In *Proceedings of theAustralian Software Engineering Conference (ASWEC '97),* Sydney, Australia.

McGrath, G. M., Campbell, B., More, E., & Offen, R. J. (1999). Intra-organisational collaboration in a complex, rapidly-changing information services company: A field study. In *Proceedings of ISDSS'99*, Melbourne, Pacific Mirror Image, Melbourne, Australia, ISBN 0732620740.

Muehlen, M. Z. (2004, July-October). Organizational management in workflow applications—issues and perspectives. *Information Technology and Management, 5*(3-4), 271-291.

Mishali, O., & Katz, S. (2006, March 20-24). Using aspects to support the software process: XP over Eclipse. In *Proceedings of the 5th International Conference on Aspect-oriented Software Development*, Bonn, Germany.

Oliveira, T. C., Alencar, P. S. C., Filho, I. M., de Lucena, C. J. P., & Cowan, D. D. (2004, March). Software process representation and analysis for framework instantiation. *IEEE Transactions on Software Engineering, 30*(3), 145-159.

Paech, B., Dorr, J., & Koehler, M. (2005, January). Improving requirements engineering communication in multiproject environments. *IEEE Software, 22*(1), 40-47.

Raffo, D. M., Harrison, W., & Vandeville, J. (2000). Coordinating models and metrics to manage software projects. *Software Process: Improvement and Practice, 5*(2-3), 159-168.

Richardson, G. P., & Andersen, D. (1995). Teamwork in group model building. *System Dynamics Review, 11*(2), 113-137.

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Wohed, P. (2006, January 16-19). On

the suitability of UML 2.0 activity diagrams for business process modelling. In *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling,* (pp. 95-104), Hobart, Australia.

Sawyer, S., & Guinan, P. J. (1998). Software development: Processes and performance. *IBM Systems Journal, 37*(4).

Scacchi, W. (1999, June 27-29). Understanding software process redesign using modeling, analysis and simulation. In *Proceedings of the ProSim'99 Workshop on Software Process Simulation and Modeling*, Silver Springs, OR.

Sharp, A., & McDermott, P. (2001). *Workflow modeling: Tools for process improvement and application development.* Norwood, MA: Artech House.

Stelzer, D., & Mellis, W. (1998). Success factors of organizational change in software process improvement. *Software Process: Improvement and Practice, 4*, 227-250.

Verlage, M. (1996, October 16-18). About views for modeling software processes in a role-specific manner. In *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, (pp. 280-284), San Francisco, CA, USA.

Wang, Y., & Bryant, A. (2002, December). Process-based software engineering: Building the infrastructures. *Annals of Software Engineering, 14*(1-4). J. C. Baltzer AG, Science Publishers.

This work was previously published in Designing Software-Intensive Systems: Methods and Principles, edited by P. Tiako, pp. 1-20, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 6.4
Computer-Aided Management of Software Development in Small Companies

Lukáš Pichl
University of Aizu, Japan

Takuya Yamano
International Christian University, Japan

ABSTRACT

This chapter focuses on the principles of management in software development projects and optimization tools for managerial decision making, especially in the environment of small IT companies. The management of software projects is specific by high requirements on qualified labor force, large importance of the human capital, low level of fixed costs, and highly fluctuating product demand. This yields a dynamic combinatorial problem for the management involving considerable risk factors. The key aspects addressed here are structuring of the project team, cost estimation, and error avoidance in the implementation phase of the project. Special emphasis is put on human resource and fault-tolerant management of the project cycle. Discrete faults and continuous stochastic inputs are used to test and evaluate project variants. We have developed an online simulation tool for this purpose that facilitates findings of optimal resource structures and creation of optimal network from task relations. General principles of software project management are presented along with the analysis of the software project simulation tool in a brief case study.

INTRODUCTION

The history of modern project management in general dates back to the 5th decade of the last century in connection with large military projects. About 1 or 2 decades ago, a close attention started to be paid to risk assessment and coordination of mammoth software projects (MS Windows development, etc.). Although it is fully recognized that the way of management of software projects

often matters more than financial resources (and a frequent success of small software companies sold at astronomic profits to giant SW development companies demonstrates this point), relatively little is known what are the crucial factors for success. A project can be defined as a temporary endeavor undertaken to create a unique product or service (i.e., in the present case, software) or another product by using software at a large scale. It is noteworthy that a complexity limit was empirically discovered in the software development, which is as low as about 10 software engineers working on one project. Therefore, an appropriate management is crucial since most of the software projects exceed this number.

Software development and its successful management is a key issue for a number of small IT companies and, with increasing importance, also for their clients. The project management (PM) common fundamentals are integration, scope, time, cost, quality, human resource, communications, risk, procurement, delivery and service to customers. Software project management (SPM) is, in addition, characterized by unique success factors derived from the unique components of IT projects. There are specific requirements on the applicability of standards, fault-tolerance, risk management, project scheduling, code development and testing techniques. Further important issues are selection and use of third-party software and also the intellectual property rights.

It has been noted in recent surveys that most software projects suffer from inadequate management techniques that ignore the unique characteristics of this field (cf. Northwest Center for Emerging Technologies, 1999; US Government Accounting Office, 1979, 2000). The most cited reasons are poor strategic management and underestimation of human factors in particular. It is known that about one half of software projects was delayed in completion and one third was over budgeted in 1997-1999, similar to the first study conducted in 1979 on this problem by the US Government Accounting Office. This remarkably persistent problem has been gaining increasing attention in scientific literatures for about a decade (cf. Abdel-Hamid & Madnick 1991; Humprey & Kellner, 1989). Since then, books and practice guides (e.g. Bennatan, 1995; Jalote, 2002) have appeared with different levels of rigor, but the number of detailed investigations in scientific journals has been rather limited (cf. Drappa & Ludewig, 1999; Rodrigues & Bowers, 1996). There is also a nuance to be noted: Traditional PM aims to solve certain types of *problems*, while SPM is rather a *process* than a solution of a problem, and therefore it requires a different approach.

Major authorities among the professional organizations in the field of SPM are Project Management Institute (PMI), Software Engineering Institute (SEI) and IEEE Software Engineering Group. These recognize the following important factors for a successful project:

- Leadership,
- Communication,
- Negotiating,
- Problem-solving methodology,
- Information sharing and training, and
- Relevant technical expertise.

Coordination and cooperation are the key factors; this is within the responsibility of the administrative hierarchy that typically includes a coordinator, assistant project manager, program manager, and a software development coordinator. Each project typically involves a team, targets certain customers and relies on contractors, and must be backed by sponsors, executives, and functional managers.

The first principle of project management is that there exists no universal principle at all. Attention has to be paid to project size, project type, culture of the project team and other factors. Software projects, in addition, require a special emphasis on the communication of *technical* experts in order to guarantee code portability

and program compatibility. Thus, one may raise a question whether a rigorous methodology for SPM is, in fact, possible. In this chapter, we (a) give an overview of managerial approaches in the field, and (b) address the gap in the standard SPM theory and practice, which is the lack of portable and customizable computer simulations for accurate estimation of project costs. In the early (but crucial) project phase when decisions are made so as to whether start a particular software project or not, such estimations are typically very crude. Such strategic decision making then inevitably leaves a space for cost increases, software delivery delays and even project failures.

Software companies are complex environments in which managers are faced with the decision-making problem involving uncertainty. Because of the complexity in the interactions among project tasks, resources, and people, estimates using average values of the project factors are very crude, and the errors are typically in the orders of 25%-100 % or even more. It is well known in the queuing theory that average output of a system with stochastic inputs can be substantially different from system output based only on average inputs. Many software projects at present disregard this point, or attempt to address it by using the best, mean, and the worst scenario, which still ignores the queuing structure of the project components (two blocks in a queue, each with the mean scenario, can produce a result even beyond the average worst-case scenario, for instance when a peak congestion in the queue results in a hardware damage or suspension of software service). Therefore even the overall worst-case estimates may be too optimistic and vice versa.

A deterministic algorithm can hardly be applied to estimate project costs, but the cost of false decision is typically enormous. Simulation techniques form a bridge to overcome this problem and to find the probabilistic optimum. In this work, we deal with a decision-making problem in the context of the software project management using three levels of detail, namely, (a) a decision whether to accept or refuse a new contract for a specific software project (complete computer simulation screening), (b) how to organize the project team (human aspect), and (c) what measures to take in order to optimize the cost of an accepted project with a given project team (resource optimization). Because of the importance of human factor in project management, we have decided to develop and provide a customizable, object-oriented, and free software project simulation environment that facilitates duration and cost estimates and supports decision making. Such a tool is consider more applicable than a fully deterministic optimization program with an implicit hard-encoded "general" project topology, however complex its parameterization might be.

The chapter is organized as follows. In Section 2, we review the managerial recommendations for project management, focusing on the specific features of software projects. Then we proceed to computer simulation of software projects in Section 3, discussing general design issues along with their particular implementation in the presently developed object-oriented simulation tool. Section 4 gives the simulation results for a selected case study along with discussion of their broader implications. Concluding remarks close this chapter in Section 5. We also recognize that the SPM area is, in fact, very appropriate for agent-based simulations, although it has been largely neglected in AI applications thus far.

MANAGEMENT OF SOFTWARE PROJECT

A successful project strategy is a balanced blend of development fundamentals, risk management, schedule-control, and mistake-avoidance techniques, adjusted to a certain trade-off in product quality, project cost and delivery schedule. One of important specific features in software projects is the huge range in productivity and ability of human

resources. Therefore selection, organization, and motivation of the team are the key factors of SPM success or failure. In this chapter, we elaborate especially on these factors. Considering what has been outlined above, it is unlikely if not impossible to find a generally applicable SPM strategy. Instead, we focus on the development of a software simulation tool that helps to select project teams, estimate the project risks in a variety of possible scenarios, and to identify possible failures before these really occur. Here we develop a SW project simulation tool that is customizable for a particular product, human resource structure and development environment. The source code is open and the tool is free to download (Online simulation application, 2005). Let us note that there exist commercial PM tools too, e.g. MS Project 2000. Their inbuilt computer simulation features are often limited. Since no two projects are really the same, the proprietary source code of the commercial products which does not allow any modification means also a serious limitation to their applicability.

Principal functions of a general project management can be listed as (Northwest Center for Emerging Technologies, 1999):

- Define scope of project
- Identify stakeholders, decision-makers, and escalation procedures
- Develop detailed task list (work breakdown structures)
- Estimate time requirements
- Develop initial project management flow chart
- Identify required resources and budget
- Evaluate project requirements
- Identify and evaluate risks
- Prepare contingency plan
- Identify interdependencies
- Identify and track critical milestones
- Start the project and track its progress
- Participate in project phase review
- Secure resources as needed
- Manage the change control process
- Report project status
- Finalization or quitting of project

In order to plan or monitor a certain project, the basic useful tools are project flow charts and network queuing diagrams for interdependent operations (visualization of tasks and their relations). Any project design should start from the final product. Therefore it is important to asses the product characteristics, size, requirements and methods of management. Project planning then means determination of available resources, selection of life-cycle model, and the design of a development strategy. Once the project starts, it needs to be tracked for costs, schedule and human efforts. When discrepancies between the plan and real state arise, a portfolio of appropriate measures should be available to handle such case. Each project can be classified into certain phases (i.e., milestones in the project tracking and project management). In case of SPM, these are:

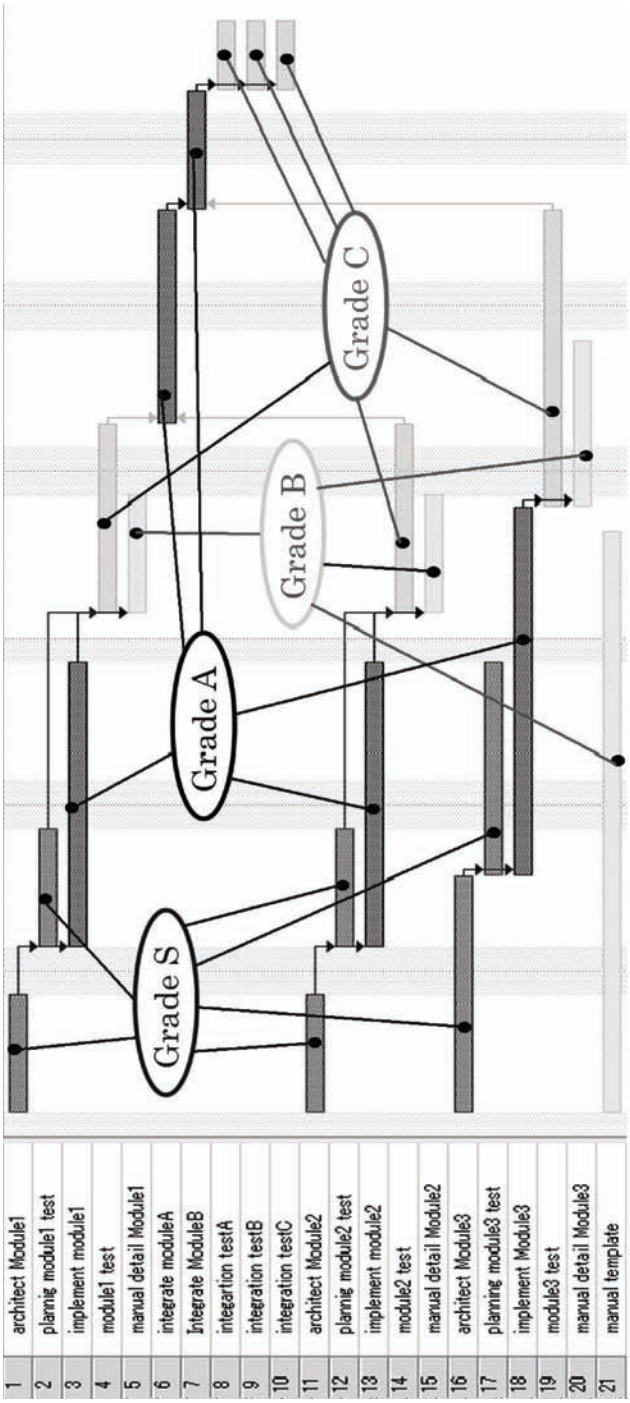
1. Software concept;
2. Resource requirements;
3. Architecture development;
4. Detailed design;
5. Implementation, programming, and debugging;
6. System integration, testing, and quality assurance; and
7. Product deployment, distribution, and maintenance.

Interestingly, major mistakes in failed software project (identified ex post) appear to be very alike. It is therefore crucial to identify their emergence based on certain fingerprint patterns and eliminate them as early as possible. To that aim, one can used the so-called McConnell's Anti-Patterns, related to failures in human resource, process, product, and the technology. For instance, these are the customer-developer friction (the programmer "knows better" what the customer "should" need), "politics over substance" (e.g., prestige competi-

tion on international level in science policy and R&D), wishful thinking (withholding cooperation quietly by sticking to formal procedures), or the lack of money ("priority shift" in the middle of the project) rank among the most serious issues.

Process related mistakes include unrealistic schedules (following "optimistic variants"), contractor failure (resources in computer engineering are often stochastic), insufficient planning for phases under time pressure (e.g., complications

Figure 1. Work breakdown structure (WBS) for a sample project



arising in debugging of "nasty codes"—applies to the author of the code, the more to someone else). Product-related mistakes typically include lack of customers (e.g., "products for researchers in bioinformatics or nanoscience" or certain "software products for the elderly"). Technology related mistakes often include unrealistic extrapolations of available resources ("counting on the Morse law", or admiration to new platforms "preference over a CPU maker without caring for compiler availability").

The important issues to check are: What will be the total cost? How long will it last? How many developers does it need? What resources are required? What can go wrong? And finally, the most important question for financial managers is the rentability, measured in the net present value (NPV), return on investment (ROI) or payback period. Since the required rentability can be viewed as an associated fixed cost, we do not need to consider it explicitly in what follows.

Before proceeding to the design and simulation issues in the next section, we would like to note that there exist various movements attempting to change the landscape of software development completely. One of these is eXtreme Programming (XP), a lightweight methodology for small teams, a highly iterative approach in which programmers closely interact with application users about the SW test releases in development (originally for small release applications based on certain HW or software architecture spikes). In a cycle of test scenario, user stories (bug reports, feedback from on-site customers) and incremental release version planning, the code development can be enormously accelerated. The reverse side of this methodology is high requirements on coders skills and enthusiasm for the project. In this respect, there is also an important problem of measuring the programmer's output and determining appropriate rewards. One measure frequently applied is the number of lines of code (LOC) together with the number of program function points. This is certainly an informative criterion but a care should be taken when to use LOC as a motivation and remuneration basis. LOC may work well in case of standard large-size projects but is certainly inappropriate in case of XP and other lightweight SW development methods.

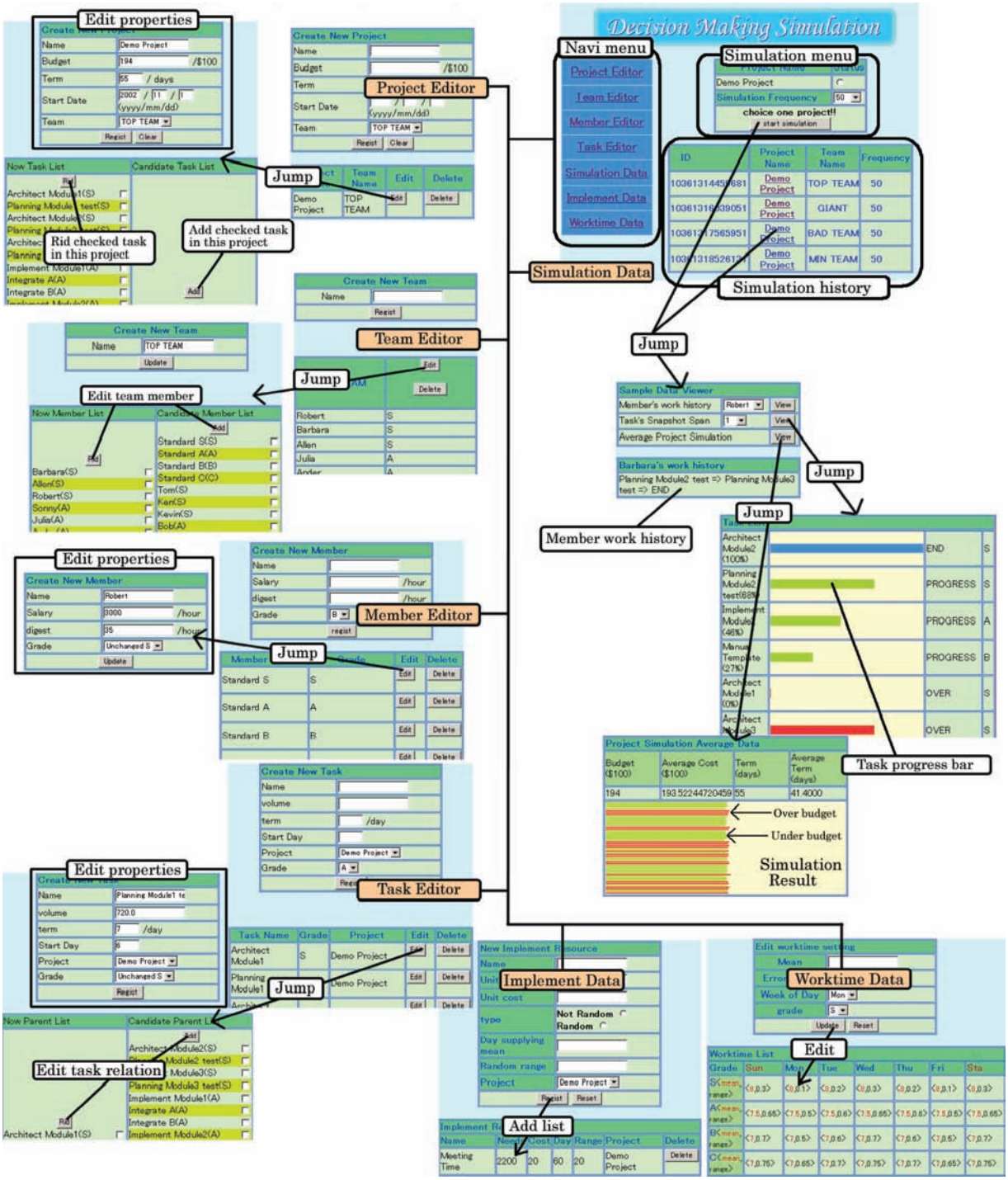
In the next section, we develop a general purpose simulation tool for software projects, focusing on the project structure, planning and stochastic resources (including stochastic failures).

COMPUTER SIMULATION OF SOFTWARE PROJECT

In order to facilitate the software development process in a specific way, we have developed a java-based online application with Graphical User Interface (GUI), which allows the user to define tasks, properties, resources, and team members including various attributes. The structure closely follows the principles of SPM. Java was selected because it is an object-oriented programming language with an easily implementable and well-established Web interface. Structure of project activities is outlined in Figure 1. The screen shots of the simulation environment are shown in Figure 2, starting from the front page entitled "Decision Making Simulation" in the right upper corner of the figure.

In the application program, there are four types of data: project data, task data, human resource data, and team data. These are all subject to optimization. The stochastic environment is simulated in two modes: (a) binary process that occurs with a given probability per unit of time (typically a false outcome of a certain task, for example, we set higher probability for coding mistakes on Monday mornings and Friday nights), and (b) probability distribution of some input data with a predefined range of values (data bandwidth available in external network, supercomputer CPU time available for rent, etc.). Both the discrete probabilities and continuous probability densities can be derived

Figure 2. Application interface includes project, team, member, task, resource and worker editors, and displays all data



from histograms of real systems and input into the model by specifically tailored non-uniform random number generators. In particular, this procedure is as follows:

- Divide the range of the stochastic factor, $<x_1, x_2>$, into representative bins $\{b_i\}$.
- Create a statistical record of the stochastic factor over a suitable period of time.
- Generate probability for each bin $p_i = N_i/N$, where N_i is the number of observations falling in bin b_i and is the total.
- Tailor a custom random number generator by using the following algorithm:
  - Repeat
  - Generate a uniform random number $r = x1 + (x_2 - x_1)\times$rnd()
  - Generate a uniform check number $c = p_m \times$ rnd() ($p_m = \max_i \{p_i\}$).
  - Determine bin i_o (r) for the number r
  - If return r (and exit).
  - Until forever

Here stands for a uniform float random number generator with values between 0 and 1 (a standard function in the libraries of most programming languages). Our simulation environment (cf. Figure 2) allows to set the relationship of tasks and structure human resources in a flexible manner. The task's determinants are time, grade, deadline and the queueing structure (standard and priority queues). Human resource (worker) determinants are skill, grade, and wage. Workers pick up tasks from the queue based on custom project team structure. Human resource is divided into four grades. Any group can take a task, if the group grade permits. Last is the project resource determinants, such as CPU time required or office supply items. Registered resources are acquired and consumed by units of days and increase the project cost. Cost of each resource is in principle time dependent, therefore various cost pricing schemes (FIFO, LIFO, opportunity costs) can be included in a straightforward manner.

Data Structures in the Simulation

The particular data structures in any SPM tool should derive from project resources and project organization. In order to design the simulation environment as general as possible, we do not hard-encode the project structure. Instead, several online editor forms allow to add project components as needed and register their mutual relations (Project, Team, Member and Task Editors in Figure 2; also Resource Editor, not shown).

The present java application stores all simulation data in a database (implemented with the MySQL relational database management system). The main database components are Member, Task, Project and Resource. Member is determined by name, identification number, wage or salary, and performance measures (A, B, C and S). Task is determined by identification, queuing schedule, progress indicator, deadline, and difficulty grade. Projects are distinguished by name, identification number, budget limit, team available, task composition and the deadline. Resources are determined by identification, number of units required by each project, unit cost, average daily supply, distribution width, and identifications of calling projects. Functional relations of the data during the course of simulation are shown in Figures 1 and 2. The simulation program is executed in discrete units by taking series of snapshots of the current progress (project, task, worker), incrementing the project immediate cost step by step. For the sake of simplicity, we implemented two particular types of random parameters: worker's fault probability p_i and a continuous randomized resource x (e.g. daily cash flow from

$$\rho(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}.$$

Generalization to a general histogram is straightforward as discussed above.

Application Data

In order to test the simulation environment, we have adopted sample model data. The respective Work Breakdown Structure (WBS) is shown in Figure 1. WBS is a useful tool to estimate project duration and workload of project participants. It is a 1D hierarchical list of project work's activities which shows inclusive and sequential relations. The 21 items listed in Figure 1 are the project activities. The labels "A", "B", "C" and "S" in the Figure show the lowest grade of the labor force qualified to deal with them. More detailed WBS in managerial applications may also distinguish the managerial level (project, task) from the technical level (subtask, work packages with definable end results, effort level). Instead of a chart, decimal catalogue-like form is sometimes used.

Next, we have created a sample case study project consisting of three modules. Each module has an architecture phase, implementation phase, integration phase and testing phase. The number of project modules and resource sharing among them is completely flexible. This is an extremely important point for the decision making support, since the entire company can be simulated in such framework by combining various modules (or projects) in one large scale simulation. Therefore

Table 1. Prototypical teams

TOP TEAM: high output, low cost					id	name	salary	out pt.	No
5	Robert	3000	35	S	1	S	2000	20	S
6	Barbara	2780	27	S	2	A	1500	20	A
9	Allen	2500	25	S	3	B	1000	20	B
11	Sonny	1800	23	A	4	C	650	20	C
12	Julia	1650	23	A					
17	Ander	1850	25	A	BAD TEAM: low output, expensive				
18	Susan	1250	25	B	7	Tom	2000	18	S
19	Diana	1300	23	B	8	Ken	2200	20	S
23	Ernie	1200	23	B	10	Kevin	1800	18	S
24	Melissa	900	26	C	14	Eva	1500	18	A
25	Nancy	800	25	C	15	Larry	1450	17	A
27	Nita	700	24	C	16	Jerry	1300	16	A
GIANT: too large team size					20	Mike	1100	22	B
7	Tom	2000	18	S	21	David	1000	20	B
8	Ken	2200	20	S	22	Eliot	1050	18	B
9	Allen	2500	25	S	26	Sander	750	23	C
12	Julia	1650	23	A	28	Anne	650	20	C
13	Bob	1500	20	A	29	Richard	600	18	C
14	Eva	1500	18	A	MINI TEAM: size is small				
15	Larry	1450	17	A	8	Ken	2200	20	S
19	Diana	1300	23	B	9	Allen	2500	25	S
20	Mike	1100	22	B	11	Sonny	1800	23	A
21	David	1000	20	B	12	Julia	1650	23	A
23	Ernie	1200	23	B	19	Diana	1300	23	B
26	Sander	750	23	C	23	Ernie	1200	23	B
27	Nita	700	24	C	24	Melissa	900	26	C
28	Anne	650	20	C	25	Nancy	800	25	C

the general design above and our java application tool in particular should not be misunderstood as a mere "single project simulation" unrelated to other activities in the SW company.

In particular, we implemented four sample teams: "TOP TEAM", "GIANT", "BAD TEAM" and "MINI TEAM" (see Table 1). These teams consist both of common and extra members. The project optimized here uses a stochastic resource—external supercomputer time—of 2,200 minutes in total (normal distribution with 1 hour mean and 20 minute dispersion). The flow of task relations among workers of 4 grades (S>A>B>C) in time is shown in Figure 1. Simulation is executed for each team, and the results are evaluated in the application program. Because of the general project structure that can be flexibly created using the online "Editor" forms for each project component, a universal optimization routine cannot be efficiently used, except for a full screening (gradually building teams by adding members still available; gradually including resources from the pool of resources available). Although this brute-force (full screening) approach feature was implemented in our program for the sake of completeness and suffices in case of small companies, it is not recommended in large simulations for obvious efficiency reasons. Let us note that the application performs two types of computations: (a) multiple simulation runs and the best-case, average, and worst-case analysis for each managerial decision (i.e., for each fixed project structure), and (b) brute-force full screening for the best project structure (small-sized projects). Because the SPM in practise is a multivariant decision making process, it is also preferable that the management modifies project structures as desired and then evaluates each variant separately, learning the most important trends and worst-case scenarios from the simulation results.

Interface and Technology

Let us briefly summarize the software project management tool developed in this work. In the simulation environment, users change and input all data with the Graphical User Interface based and Java Server Page. Logic programming is also implemented in Java. Mysql database is used to store simulation data. The interface described by Figure 2 allows the user to choose from a variety of functions in the left navigation menu of the window "Decision Making Simulation," to set the simulation properties and to check simulation progress and history. There are five editors for members, projects, teams, tasks and resources. The editor page can create and delete relations among the data. There are three main parts in Figure 2. The first one is the team member's working history. The second is a progress snapshot of all project tasks, each having one status assigned from "READY", "PROGRESS", "OVER", "END", and "NO STATUS." At last, the simulation result graph shows the total cost and indicates whether the simulation run is over the project budget or not (long and short lines indicated by arrows in the window "Simulation Result"). Work time data sheets enable editing of worker's grade and performance. The parameters are the mean and the stochastic error range. The resource editor page adds the resource data and their possible stochastic distributions for any project. Resources can be fixed or random; these are especially important to decide the need of the project and its final cost.

After multiple simulation runs, project variants are compared in order to find the optimum. This environment is used to design the best team possible for a given project (or a set of projects). The optimization is conditioned, (i.e. the Top Team in Table 1 is chosen if the project manager

needs the most economical team, and the Giant team in Table 1 is chosen only when maximum speed is the criterion). Whenever a task over the deadline is found, its margin is checked, the length is edited and all other related tasks are adjusted. Thus the optimal decision making is possible with using the simulation data. Input data can be changed flexibly, including task relationships, team members, project teams, project budgets, workers ability or random input streams, and then reused for the simulation.

CONCLUDING REMARKS

In spite of various established project management models and quality management systems, such as ISO-9001, Capability Maturity Model, or Constructive Cost Model (COCOMO), SPM simulation has not received sufficient attention in academia. Also software projects in businesses often suffer from inadequate management. The complexity of software projects with various stochastic features involved implies that an object-oriented computer simulation is a very appropriate approach. In addition, because of the autonomous human factor in program coding and complexity in motivation of software developers, agent-based simulations are expected to contribute to this field in the future.

This chapter summarized principal features of software project management along with presenting a newly developed SPM simulation tool. The tool is a general purpose object-oriented simulation environment with emphasis on fault-tolerance in the development process. Randomized inputs, randomized faults, variable team structures, branching queues and idle time analysis are included (Online simulation application, 2005). The online Java program adopts flexible data structures for the teams and stores all simulation data in a dynamic database. The contribution of this work consists in developing simulation technology for the new area of SPM.

ACKNOWLEDGMENT

Lukáš Pichl acknowledges partial support by the JSPS Grant-in-Aid. We are grateful for the comments of anonymous reviewers on the background of this work.

REFERENCES

Abdel-Hamid, T. K., & Madnick, S. E. (1991). *Software project dynamics: An integrated approach*. New York: Prentice Hall.

Bennatan, E. M. (1995). *Software project management: A practitioner's approach*. New York: McGraw-Hill.

Drappa, A., & Ludewig, J. (1999). Quantitative modeling for the interaction simulation of software projects. *Journal of Systems and Software, 46*, 113.

Humphrey, W. S., & Kellner, M. I. (1989, May). Software process modeling: Principles of entity process models. *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh (p. 331).

Jalote, P. (2002). *Software project management in practice*. Adison Wesley.

Northwest Center for Emerging Technologies (1999). *Building a foundation for tomorrow: Skills standards for information technology*. Belleview, WA.

Pichl, L. (2005). Software process simulation. Retrieved March 1, 2005, from *http://lukas.pichl.cz/spm.zip*

Rodrigues, A., & Bowers J. (1996). System dynamics in project management: A comparative analysis with the traditional methods. *System Dynamics Review 12*, 121.

US Government Accounting Office. (1979). Document FGMSD-79-49. Retrieved September 2003, from *www.gao.gov:8765*

This work was previously published in Computational Economics: A Perspective from Computational Intelligence, edited by S. Chen; L. Jain; C. Tai, pp. 205-216, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 6.5
A Survey of Competency Management Software Information Systems in the Framework of Human Resources Management

Alfonso Urquiza
Francisco de Vitoria University, Spain

ABSTRACT

One of the greatest surprises of the Internet economy is that far from replacing people, the use of advanced technology is confirming that talent is the most valuable asset in today's organizations. In this context, competency management (CM) software automation practices become the most valuable business approach to define, measure, and manage talent needs, the human capital of the organization.This chapter's position is that CM process automation in competitive, knowledge-intensive e-business oriented organizations requires that information technology (IT) address software strategy in a comprehensive human resources management (HRM) framework. Core competency management-related applications are deployed in current corporate e-business transformation processes in association with the use of innovative employee—facing relations management technology and reengineering most HR transactional domain type of applications in place.The chapter shows the CM software evolution from a previous fragmented market situation to a much more integrated scenario in which best-of-breed single-function oriented products preferences are now swiftly moving to the enterprise resource planning (ERP) type of architecture.

INTRODUCTION

As the Internet age transforms the way people work and live, organizations keep continuously embracing the new opportunities and challenges

generated by this relatively recent and significant change, introducing a new knowledge revolution (Nordstrom & Ridderstrale, 2000).

Today's economy is creating a new breed of "intelligent" organizations, where a very high percentage of the total workforce is comprised of knowledge workers. In this context, the ability to effectively manage human capital investments becomes essential to ensure business success. Organizations gain real advantages by applying Internet technology to the measurement and management of their talent needs, the human capital of the organization.

The long transition from traditional *"personnel administration"* activities to most recent *"human resources (HR) management,"* has meant an evolution from a purely functional to a process-oriented approach in which all those activities associated with the management of employment and work relations are included (Boxall & Purcel, 2003).

Traditionally, organizations of any size or activity used to focus primary attention on automating payroll & basic administrative functions. Other administrative-required functions (like recruiting, training, etc.) were largely assumed and performed in a non-automated way, thus creating large staff departmental units in these areas, non-associated with the organization's primary business.

IT solutions at the time were not designed to manage *knowledge assets*; they were focused on managing *physical assets.* Individual employees and managers used to call upon HR to satisfy different kinds of requests. Individual employees typically required tracking and processing personal information, such as compensation, benefits, or other related data. Managers required HR to provide information on recruitment or training services**.**

In recent HR management, the new e-business context has transformed and automated most HR operations, thus generating additional efficiency: process flows are handled like "automated transactions" and self-service functions appear, simplifying individual employee/manager relations within the organisation, automating administrative tasks and enhancing task-driven routines formerly performed by HR departments.

In the new "*human capital*" (HC) paradigm, it is not just about modeling and automating "tactical" HR functions. Two new dimensions are introduced. The first to consider is that a new role appears for HR: that of *strategic asset (talent) management.* The second is that HR becomes just another component in the organisation, like financial management, supply chain management, customer relations management, or IT, all of them driven to produce a product or service that generates value to the customer (Laudon, 2004).

It is in this new, comprehensive management context where competency management fully develops itself, becoming the integrating "glue" element in HC management systems (Sagi-Vela, 2004), thus reshaping today's and future HR management implementation strategies.

In the context of this work, competence is understood as the set of knowledge, skills, and attitudes required in people to perform a specific task in an efficient way (Sagi-Vela, 2004). CM is a comprehensive HR process that starts by defining the required organizational competencies, assigns them to employees, observes them through behaviour, asses them according to an organization's defined values, and permanently improves them (Levy-Leboyer, 1997). Unlike in traditional transaction-oriented HR practices, a CM strategy should pursue the following goals:

- Support business objectives, providing information to *acquire*, *maintain*, *influence*, *develop*, and *retain* the *right employees.*
- *Align* people, processes, and technology around *shared values.*
- *Measure* the strategic value of human capital investments.
- *Anticipate* human capital changes.
- *Learn* from industry-best practices, leveraging benchmark data.

In today's knowledge-intensive organizations, competency management is not viewed just like a new function associated with a single job topic (learning, employee career development, etc.), nor is it an additional responsibility, to be added to the traditional list of activities that the organization expects to be accomplished by the HR department.

Experience shows that effective CM strategies succeed most when all HR processes in operation—not just a portion of them, like career development, performance management, or learning, for example—are reviewed and aligned to the talent management vision, accomplishing a real e-business transformation in organization's processes. This remains true even in situations where CM strategies are focused to a limited extent in overall taskforce, applied only to strategic level of employees within the organization (i.e., managers), which happens very often, particularly in large-size organizations.

The main objective pursued in this chapter reflects this business reality: There is not just a simple, unique solution that automates CM process in an organization. CM is not an IT vendor product; it is a full *comprehensive strategy* that transforms the HR management function from a mere, although mature, administrative level to a strategic business alignment role.

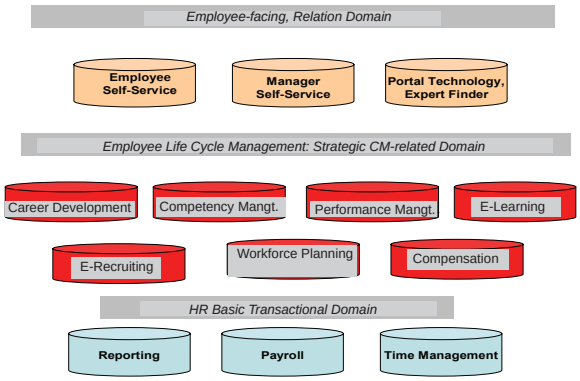
This chapter's proposed detailed position is that, when it comes to implementing a CM strategy in a competitive, knowledge-intensive e-business-oriented organization, IT components in the three areas showed in Figure 1 should be addressed (to the extent placed upon defined CM deployment requirements) or sometimes re-engineered as a 'combined' operation:

The *transactional domain* type of applications are not obviously related to CM functionality, but some processes in this area require small re-engineering enhancements to work in a CM driven e-business environment. Functions included in the *employee life cycle domain* are driven to enhance employee value, core in CM deployment. *Employee-facing* relations management technology deployment usually exceeds the scope of HR, but the use of such tools significantly improves CM-related process efficiency: use of paper and forms-oriented types of activities are replaced by Web-based self service processes, generating significant return on investment (ROI) in large organizations and employee satisfaction in online experience.

In order to fully understand this chapter's business-oriented perspective, the rest of this chapter is structured as follows:

The next section shows the background in which this work is based, followed by an analysis of IT industry response to the CM evolutionary organizational requirements. Then, a characterization of most relevant HR processes is presented, depicting inter-process relations, and supporting

Figure 1. HC key components in CM deployment strategies



and clarifying the proposed comprehensive approach to CM implementations. Finally, today's commercial CM-related technology scenario is presented, followed by future trends and final conclusions uncovered in this survey.

BACKGROUND

Competency management is considered by some authors (Sagi-Vela, 2004) as a complementary methodology to other related HR management practices, such as emotional intelligence or, more often, knowledge management. Knowledge management (KM) is a very close concept to CM, and technical literature (Alle, 1997) tends to generate some confusion when analysing KM and CM management systems.

In the scope of this work, KM management systems are those directed to capture, analyze, apply, and re-use organizations' know-how, with the objective of performing higher-quality business processes at a lower cost and generating a competitive advantage. CM systems' focus is on employee life cycle, covering competency requirement analysis (Lindgren & Stenmark, 2002) and other related HR management topics, such as personnel selection and compensation (Sagi-Vela, 2004).

Contributions on KM designs and product evaluations are widespread in scientific literature (Benson & Standing, 2001; Friss, Azpiazu, & Silva, 2004; Kamara, Anumbad, & Carrillo, 2002; Rollet, 2003), but technical literature on CM is still scarce and dedicated to specific areas, such as organizations adjustment to CM (Lindgren, 2005) or competency development (Hardless, 2005).

In addition to above, the software capability maturity model initiatives (CMM I, 2002) have been complemented by the Software Engineering Institute with the people capability maturity model (People CMM) (Curtis, Hefley, & Miller, 2001) as the foundation for a model of best practices in managing an organization's workforce. And competency management is obviously present in various significant People-CMM process areas:

In this context, a CM software information systems survey in the framework of a comprehensive HR management process scenario might be of great value in further CM software research, for example, investigating compliance relationships between the five People CMM evolutionary maturity levels with CM market products research and development.

COMPETENCY MANAGEMENT AND THE SOFTWARE INDUSTRY

Today's SW products and services marketplace in the HR domain has been shaped in the last 25 years by the business evolution described above.

It is no surprise to anyone that until the e-business transformation with the arrival of the

Figure 2. The Process Areas in People CMM

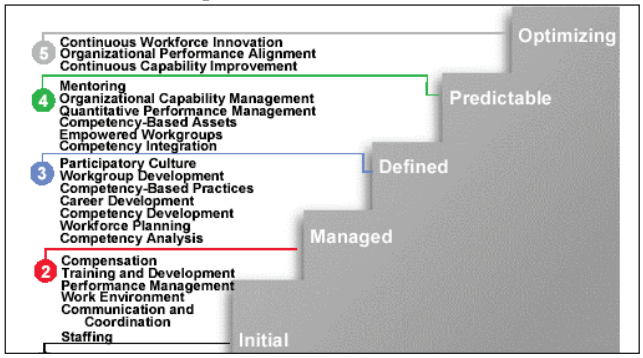
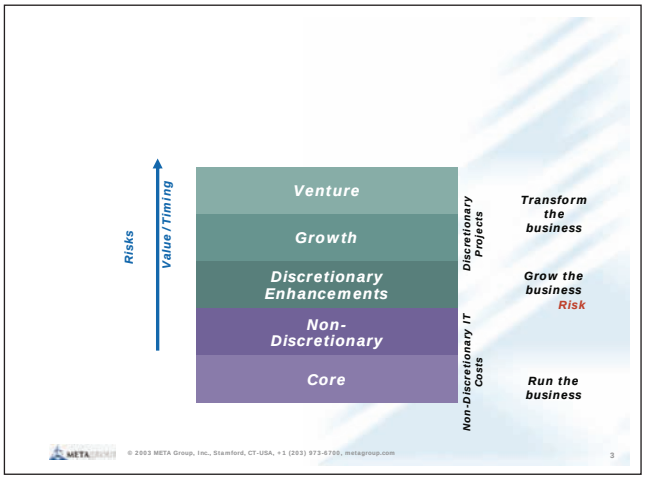


Figure 3. IT portfolio management methodology



Internet age during the late ’90s, HR businesses have created a sort of IT industry *fragmented* market. There exist hundreds (if not thousands) of different HR administration and management solutions (payroll, learning, PM, CM, etc.) in organizations around the globe. Some are global market-oriented, other vertical cross-industry oriented; most are local and country-driven solutions, aimed to satisfy specific single-market demands.

In addition to the above situation, the amount of in-house developments and custom-made implementation solutions in this field far exceeds expected figures compared to other types of support applications, such as financials, business analytics, and so forth.

But the above scenario is now changing very fast today, and the e-business evolution is driving industry from the previous IT-fragmented scenario to an extremely *concentrated* one. The vast majority of today’s large organizations (those in which CM has the largest deployment potential) are automating HR (or planning to do so) within the scope of one out of three global, integrated ERP products: SAP, PeopleSoft or Oracle.

To better understand this evolution that is reshaping the market in which CM software progress takes place and is making many best of breed stand-alone HCMS solutions disappear, let’s apply a modern, innovative approach that is useful to map organizations’ business requirements with information technology decisions. The Meta Group Technology Analysts call it portfolio management:

Independently of the economic environment, a key challenge for IT professionals has been to assess and permanently communicate the value of IT investments to the business units.

Today’s advanced organizations don’t just require a functional justification and a “business case” study to make a technology decision. The *portfolio management approach* introduces financial issues to any IT investment, as for example:

Level of risk versus expected benefits/value: Just like a pure financial investment. In IT, project deployment time, magnitude of investment, and so forth are taken into account.

Current fair value, at any time.

Expected life cycle: when will investment pay off?

Perceived relationships: defining the IT domain, its platform, and architecture.

IT assets and projects are categorized as shown in Figure 3.

The run-grow-transform classification defines the primary goal for any IT project:

"*Run the Business*" investments are focused to keep business operational (i.e., maintenance contracts, utilities, etc.). *Core* spending is for business-critical activities, like customer service, sales ordering, and so forth. *Non-discretionary* relates to organic growth in core IT assets (servers, DBMS, etc.). Business *risk* is low and expected *reward* usually medium to high.

"*Grow the business*" IT expenditure is applied to expand organization's scope, in product or services. Learning activities to develop new skills fall in this category.

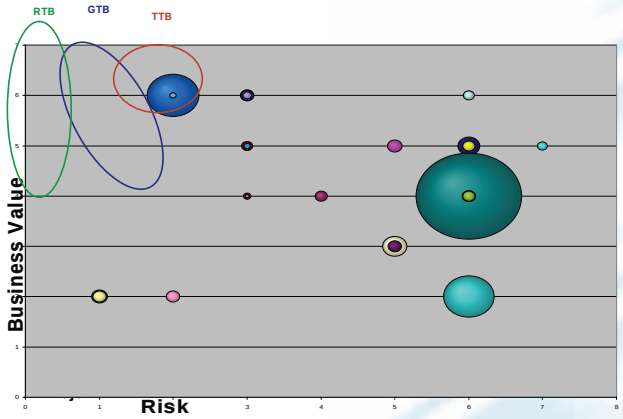
"*Transform the business*" initiatives are related to opening new markets or issues having a major impact on the current business model. Business *risks* and expected *rewards*, in the last two categories, are both moderate to high. Organizations sometimes afford the high risk (of unplanned events) and expect also high reward in *venture* initiatives, where speed is usually the way to proceed (i.e., be the first to deliver a product).

The IT industry in most recent e-business process transformations (that obviously applies to HR) drives SW product development & projects investment decisions, analysing three dimensions: investment business impact, cost and performance expectations, and risk and opportunity appraisal.

There are many SW tools in the market to analyse and manage IT portfolios: ProSight (www.prosight.com/solutions/software/), Primavera (www.primavera.com/about/trillion.asp), Niku (www.niku.com/), and so forth. The example in Figure 3 depicts a view of the different SW implementations in place in an organization, associating the value and the risk dimension.

Now, if we apply portfolio management's approach to HC management and CM, and clearly understand IT's decision-making process, current technology scenario, and, also, expected trends for the future, it will be much better understood. For this purpose, we will analyse IT-phased evolution, within the proposed HC framework in which CM strategy develops in today's advanced

Figure 4. IT portfolio classification example



organizations: the employee transactional, life cycle management, and relations management domains.

Let's first focus on the employee transactional domain. Traditionally, payroll and basic employee administration services have been considered the primary, core HR organization's service. Mistakes in this area do have a clear "dissatisfying" effect, thus damaging HR internal image. And particularly in medium- to large-size organizations, some specific processing difficulties led to prompt IT automation: earning and deductions complexity derived from different, concurrent labour agreements to apply in single organizations, continuous updates in country regulatory requirements, gross-to-net permanent calculations, and finally, banking reconcilement and automated interfaces requirements.

Payroll, therefore, rapidly became the first core "*run the business*" type of investment in HR management, parallel to other basic back-office applications like accounting. *Business risks* were *low* (processes are quite stable) and *benefits* were *medium*.

Other transaction-oriented processes followed automation in a segregated manner, just when business required the need due to growth, changes in HR policy, merging events, and so forth. Some of these processes were compensation and benefits or time management, typically non-discretionary enhancements, *low business risks*, and *high benefits*.

Initial market products' scope was local, coexisting with many made-to-measure solutions in place. Outsourcing of services models began to develop, particularly in the low- to medium-size type of organizations.

Employee life cycle management is the primary core domain for HC competency management strategy deployment in any organization. Competencies are created to manage organizations' required *talent* across the employee life cycle:

Attract talent (e-recruiting) ➔ *Assign* talent (workforce planning) ➔ *Influence* talent (performance/compensation management) ➔ *Developing* talent (e-learning) and ➔ *Retaining* talent (career & potential development).

Although integrated solutions are the most relevant implementation solutions in today's IT industry (see section *"HC market applications: The CM contribution"*), stand-alone products are also growing. We will shortly analyse performance management and CM and E-Learning, these two being the most relevant convergent solutions in the CM market today.

a. Performance and competency management

Early non-integrated deployed performance management (PM) systems were perceived by employees as non-useful, time-wasting applications. It is not until the e-business transformation process generated during the last five years that organizations perceived real business impacts in developing values and competencies, enhancing employees' business alignment. Medium- to large-size organizations started to create and track measurable skills and competencies through the deployment of advanced e-business-oriented PM solutions. About 40% of large organizations have already defined competencies for some type of jobs, and 10% have a well-defined enterprise-wide set of competencies. And the IT market today still has large expectations: 60% are still non-automated, paper-based solutions, about 25% in-house developments, and 10% product based (PeopleSoft, SAP, Workscape, etc.).

CM solutions are mostly "*grow the business*" and, to a lesser extent, "*transform the business*" *venture*-type applications. Business *risks* and *reward* are, typically, *moderate to high*.

b. E-learning

Early (1980s to mid 1990s) learning management systems (LMS) were just tools driven to automate instructor-led training activities. E-learning

today is somehow different. Cisco chairman John Chambers said in August 2003 that "*the two great equalizers in life are the Internet and education.*" More than 70 million people received education over the Internet that year.

LMS' most relevant change in the Internet age is that employees now become themselves *responsible for their own learning requirements.* Training is delivered in a personalized way, thus facilitating individual *competency development* as required and planned by the organization.

IT's e-learning products have a singularity compared to the rest of HC applications related to competency management implementations: The LMS market is dominated by stand-alone solutions. Platforms like Saba (www.saba.com/),Centra (http://www.centra.com/education/resources/index.asp), IBM's Lotus ("www-306.ibm.com/software/info/ ecatalog/es_ES/products/N105931Y77809P74.html"), and 70-plus more represent today 95% of the total market, and just 5% (although growing) are tied to ERP-integrated packages (i.e., SAP, PeopleSoft).

E-learning solutions fit into the "*grow the business*" category. Business risks are moderate and reward is, typically, moderate to high.

IT products in the employee relations management domain don't address or directly process CM-related information. They are viewed as communication-integrated tools, driven to "automate" most (if not all) HC processes, as in other parallel e-business environments. Typical applications in this domain are employee self service (ESS), manager self service (MSS), workflow, and expert finders. Let's shortly describe them.

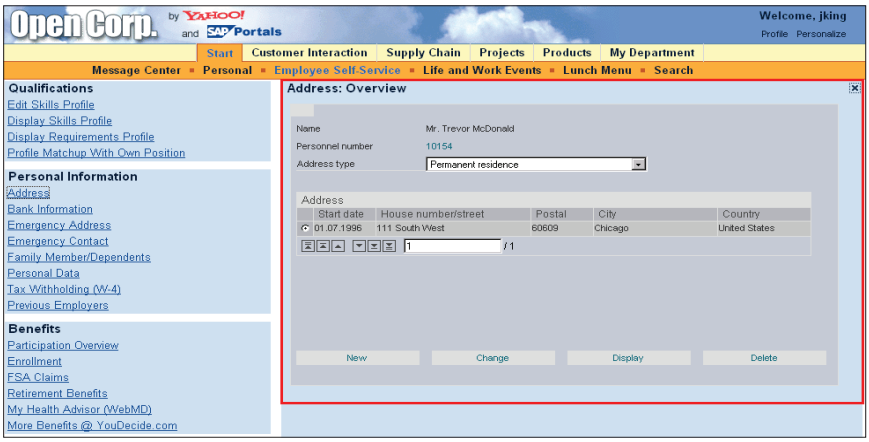
ESS are portal-driven solutions created to ensure comprehensive and controlled employee access to internal or external information and applications. Employee satisfaction and dramatic reduction in transaction cycle time over HC services are proven benefits in using this tool, typically integrated within HR ERP application.

ESS Portal solutions are in the "*grow the business*" category. Business *risks* are moderate and *reward* is, typically, *moderate.*

MSS technology and purpose is of the same nature as ESS. The only difference is in the nature of the type of applications that are relevant to this type of employee's role within the organization. Managers require a single access point to information, otherwise disseminated throughout the organization. They manage people, budgets and

Figure 5. Generic structure of ESS portal implementation

Figure 6. Example of a real ESS



groups assigned responsibilities in an effective, proactive way.

Again, MSS solutions are in the "*grow the business*" category. Business *risks* are moderate and *reward* is, typically, *moderate.*

Workflow represents a highly critical enabling technology to automate HR processes, thus critical too for comprehensive CM deployment policies: workflow replaces paper notifications with *messages notification*, *routing*, and *approval.*

Workflow servers solutions are mostly *discretionary enhancements* within the "*grow the business*" category, creating new levels of process efficiency and effectiveness (or process agility). Business *risks* and *reward* are, typically, *moderate*.

Expert finders are Web-based tools typically found in integrated ERP HC solutions used to provide the means to find and locate employees having specific knowledge or competency profile ("*who is who*"-type of approach) and, finally, ensure contact with the expert for the required period of time. Usually works tie to the workflow server functionality.

Expert finders solutions are considered discretionary enhancements within the "*grow the business*" category. Business risks and reward are typically moderate.

HUMAN RESOURCES MANAGEMENT PROCESSES

The foremost trend in information systems restructuring in recent years is process-based working. This approach enhances knowledge development and sharing, forcing IT professionals and business decision makers to collaborate and share information concerning IT and business trends. Successful corporate e-business transformation initiatives necessarily incorporate process improvement and transformation. Process understanding provides the bridge between corporate strategy and IT implementation.

CM software technology is no exception to this reality. As indicated in the introduction, CM market analysis requires a comprehensive approach within the HC domain. This brings about an excellent opportunity to critique and model value delivery mechanisms, thus clarifying business relationships.

In today's advanced organisations, *a process* is defined as a "set of activities directed to obtain a service or a product that creates value to the client." Although human capital management fits into the corporate management support domain functions (same as financials, planning, etc.), the same definition applies. Competency management

is the cornerstone in this proposed client-oriented process approach: People drive results.

The following is a generic level 1 definition of all relevant industry-accepted HC management processes representing the framework in which CM software is deployed. Best-process practices indicators are proposed, as well as information concerning the most significant internal and external application inter-relationships.

This framework will facilitate the understanding of the competency management business "ecosystem." Some HR management processes are very much tied to CM strategy deployment, and others are not. This comprehensive view will have significant value to further evaluate and assess the various software system architectures and market products available against the People CMM model.

Persons Administration and Payroll

Payroll processing represents the primary HR automated function in most organizations. It ensures that the workforce gets accurately paid on time. Critical activities are usually linked to specific country regulatory requirements, the amount and structure of labour-related accounting models to consider, and other specific reconcilement tasks.

Best practices key process indicators in this area are the following:

- Employee's administration and information flow is integrated within HR or global corporate ERP.
- Administrative tasks are performed using ESS methods, implementing portal/intranet-driven solutions and workflow technologies.

The Process Reference Model for Payroll & Administration is shown in Table 1.

The internal and external process relationship is shown in Figure 7.

The external-related applications are financials, ESS, travel management, and CRM (to directly manage incentives typically associated to sales or commercial goals).

Time and Attendance Management

In the context of knowledge persons management, TM (time control of employee time) is not a high-priority process and often deployed to a very limited extend. See Table 2.

Figure 8 shows internal and external process relationships.

The only significant interface to consider here is with any clock-in system (when required). Notifications are often made via ESS or simple workflow dialogs with authorising persons.

Best practices key process indicators in this area are the following:

- Presence control management integrated within HR or global corporate ERP.
- Decentralization of absence and time off registry.
- Absence/permit and vacation request approvals flow is fully automated.

ORGANIZATIONAL DEVELOPMENT AND CHARTING

Structure management is often a centralized process that creates, organizes, and manage organizational directory information. See Table 3.

Figure 9 shows the internal and external process relationships.

Best practices key process indicators:

- Structure definition and job description information processed within HR or global corporate ERP.
- Decentralization of organizational changes across corporation.

Table 1. Process reference model for payroll & administration

Activity	Name	Description
PA_01	Employee data maintenance	Workforce data and changes in employee status, relevant to payroll, are introduced in HR system. Sometimes, employee time reporting is the front end of payroll overall process.
PA_02	Payroll process	Taking into account monthly changes, backdate payment rules, corporate and country regulatory requirements, gross and net earnings, and deductions are calculated.
PA_03	Checking of calculated results	Post-payroll process report writing is made to check possible errors prior to HR results formal approval, prior to post-information further processing.
PA_04	Individual payroll forms generation	Payroll forms are generated and made available to employees in printed or electronic format for individual ESS access.
PA_05	Legal Social Security report generation	Adequate automated monthly files (FAN type or other) are generated, together with established payment forms (TC1s)
PA_06	Tax reports generation	According to company type and local regulations, adequate report (i.e. 110, 111, 190) is periodically generated. Earning and retention information is prepared for personal direct consultation (vía ESS).
PA_07	Social benefits information delivery	System interfaces with agreed social benefit provision companies
PA_08	Prepare bank transfer files	Files generated for bank use with net earning to be transferred to employee's account.
PA_09	Accounting File generation	As agreed with the financial department for accounting processing (general ledger, etc.)

- Change organization approval process is managed using workflow mechanisms via ESS/Intranet.

WORKFORCE PLANNING AND ANALYTICS

HR requires automated processes to evaluate generic and specific income change proposals, benefits, or any other HR cost-related issue that may be considered. Simulation and analytical tools are essential to assume control of present and future HR budget responsibilities. See Table 4.

Figure 10 shows the internal and external process relationships.

Best practices key process indicators:

- Organization HR budget calculated through HR or global corporate ERP.
- Full integration with payroll and payroll simulation and analysis.
- Full integration with the organization module (to simulate organisational changes).
- Automated budgetary analysis and measurements, identifying unusual patterns in data through technologies such as data mining and understanding correlations between measurements.

Figure 7. Internal and external process relationships

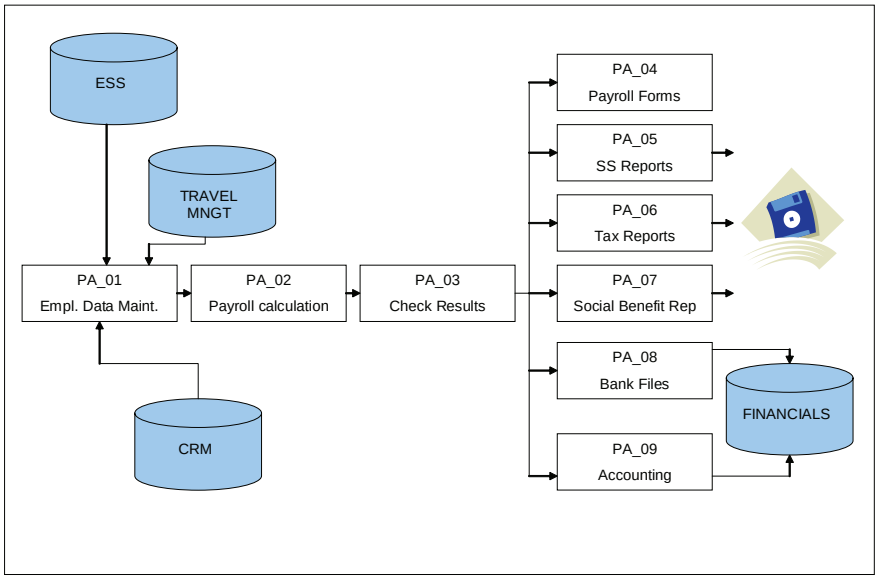


Table 2. Process reference model for time and attendance management

Activity	Name	Description
TM_01	Time schedule definition	Relevant labour calendar information is prepared: work hours, flexible schedules, and so forth. Relevant only if clock in control is in place.
TM_02	Time off request	Applicable to vacation periods or special permits.
TM_03	Time off authorisation	Authorise or reject employee request and inform HR department.
TM_04	Collect clock-in data	Exit and entry information is registered by external clock-in device, located in work premises.
TM_05	Identify deviations	Based upon calendar and work hours information, a contrast task is performed with clock-in data in order to identify eventual deviations.
TM_06	Deviations justifications	The employee justifies detected work absence, then approved or rejected by superior.
TM_07	Registry of time incidences	Time off accruals, vacation, sick leave type of information for HR information and management.
TM_08	Inform payroll	Only the payroll-relevant information is transferred for adequate processing.
TM_09	Statistics	Periodically, HR obtains statistic information to be delivered to employees via ESS.

COMPETENCY MANAGEMENT

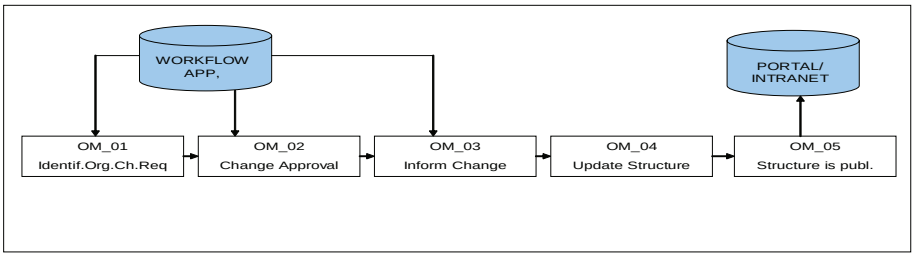
CM processes flow across organizational divisions and units as the most challenging practice in workforce management activities.

Managing knowledge assets like managerial behaviours, problem-solving skills, and so forth is somehow different than managing physical assets. The ultimate CM process goal is to provide the means to ensure *continuous development* of

Table 3. Process reference model for organizational development and charting

Activity	Name	Description
OM_01	Identify organisation change request	Every year each department performs a business strategic and functional review identifying recommended changes as business develops. Structure updates proposals are then submitted for approval.
OM_02	Change request approval	Change proposal is aggregated at organisation department level, then approved by board of directors.
OM_03	Inform on change to HR	Once new structure has been approved (includes new assignments, positions, etc.) HR assumes control to update organisation-dependent information.
OM_04	Update structure	Updating structure often requires changes in HR master data and cost centres allocation.
OM_05	New structure is made public	Internally (via B2E mechanisms) and externally if required. Quality assurance manuals also updated.

Figure 9. Internal and external process relationships



employees and organizations' competencies. It implies **observing** them through behaviour, **assessing** them according to organizations' defined values, and permanently **improving** them.

Competencies are usually linked to career planning, so employees permanently have a view of the potential career paths they are offered by the organization.

CM strategies are as good or effective as the information available. Even though CM is about managing intangibles, without the right information measuring or predicting how employees or the organization will be affected is just hit or miss. See Table 5.

Figure 10 shows the internal and external process relationships.

Best practices key process indicators:

- Corporate competency catalogue defined within HR or global corporate ERP.
- Integration with all relevant applications, supporting HR processes.
- Automated gap analysis process, matching employee profile with profile required.
- Competency profile information through B2E communication-based links.

EMPLOYEE RECRUITMENT

Even though procedures vary from one region and organization to another, the process is driven to ensure the best tracking of applicants (internal or external to organization), based upon skills and competencies to match into the required job profiles. See Table 6.

Figure 6 shows internal and external process relationships.

Figure 10. Internal and external process relationships

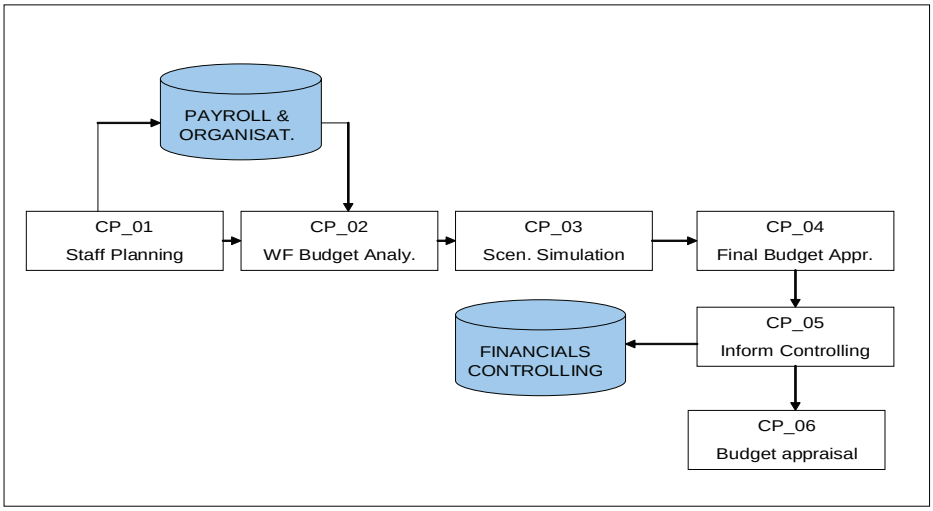


Table 4. Process reference model for workforce planning analytics

Activity	Name	Description
CP_01	Staff planning	Departments and business unit staff requirements and status requirements are processed. Aggregated information is then validated and budgeted.
CP_02	Actual cost evaluation	Current payroll cost is used to estimate cost for next period. Planned staff increase cost is estimated based upon current figures.
CP_03	Workforce budget analysis	Payroll cost models are simulated based upon previous calculations. Cost and budget effects due to organizational changes, new statutory requirements, or updated payroll and benefits configurations are simulated.
CP_04	Final configuration approval	Final proposal is approved and submitted to management committee.
CP_05	Accounting control is updated	New approved budget is incorporated in accounting.
CP_06	Budget appraisal	Permanent budget appraisal.

Best practices key process indicators:

- Recruitment process supported by HR or global corporate ERP.
- Full integration with administration application.
- Automated CV information data registry (through B2E).
- Automated mail service.

TRAINING

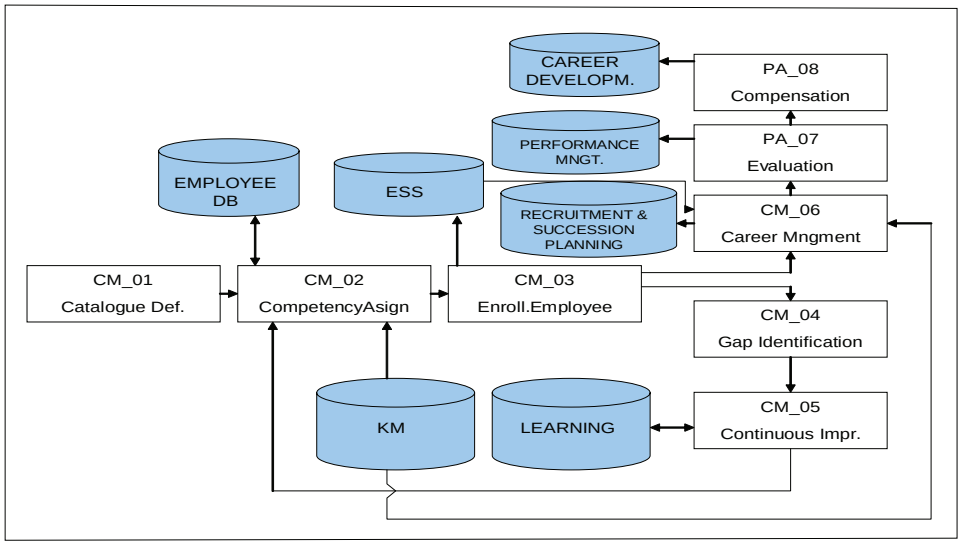
This process helps managers to evaluate and plan organization's training requirements and events. Information is processed on prerequisites, objectives, content, schedules, and locations, as well as appraisal course's results and additional management required data. See Table 7.

Figure 7 shows internal and external process relationships.

Table 5. Process reference model for CM

Activity	Name	Description
CM_01	Catalogue definition	Core and other required competencies, plus proficiency level required for each job profile within the organisation, are established and catalogued as a corporate dictionary.
CM_02	Competency assignment	Once competencies are assumed by management, an assessment process defines procedures and level-assignment rules. Competencies are then matched to employees. This is often a time- and cost-consuming non-automated set of activities.
CM_03	Employee enrollment	Individual employees may leverage competency and potential career paths within organization.
CM_04	Competency gap identification	Gap between competency level definition and employee status is identified. This information is the basis for employee continuous improvement process, usually linked to CM.
CM_05	Employee-continuous improvement activities	Individual person's development is supported in learning (e-learning) programs, providing the means to develop required competencies.
CM_06	Career management	Supported on KM information and individual development, "profiles" matching job vacancies are identified within organizational scope. Succession planning information is analysed.
CM_07	Performance evaluation	Continuous matching employee-assigned business objectives with actual performance based upon the CM model. Evaluation information is created, then managed by HR evaluation process.
CM_08	Compensation	Individual salary and benefits compensation is triggered by evaluation, thus closing the periodically updated employee development cycle.

Figure 11. Internal and external process relationships



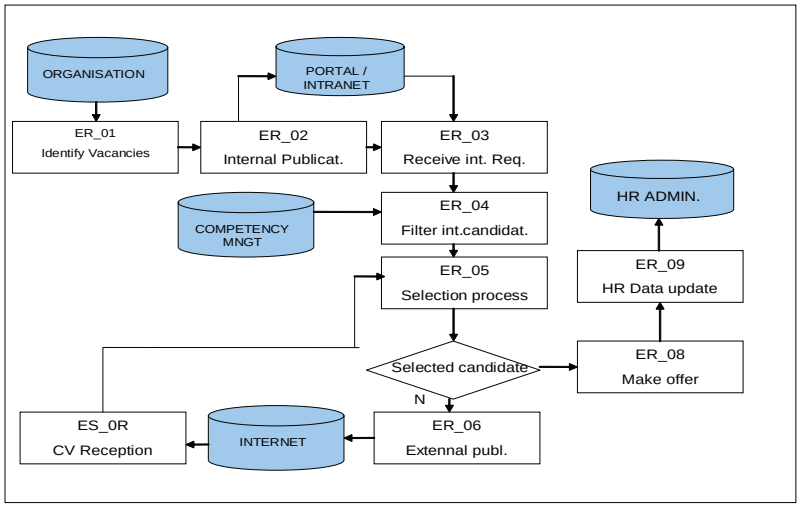
Best practice's key process indicators:

- Training management supported by HR or global corporate ERP.
- E-learning integration within ERP.
- Integration with other related ERP processes.
- Automated mail service activities required to manage training courses.

Table 6. Process reference model for employee recruitment

Activity	Name	Description
ER_01	Vacancies identification	Next-period strategic HR planning is taken as input information to identify vacancies to be assigned.
ER_02	Organization internal publication	Required position and expected candidate profile are made public within organization.
ER_03	Internal candidates reception	CV is usually updated by employee prior to candidature submission.
ER_04	Internal candidates filtering	HR analyses candidatures and filters position profile compatibility against candidate profile.
ER_05	Selection tests	HR and required technical and managerial personnel are involved during the selection process. Tests are designed in accordance to position. Technical knowledge and person's profile suitability tests are executed.
ER_06	External recruitment	Applies when no internal candidate gets position assignment. Scheduling.
ER_07	Prequalification and CV's reception	The selection application inputs candidates' information for further assessment and tracking.
ER_08	Recruitment proposal	Selected candidate receives notification and formal offer to accept position.
ER_09	HR administration incorporates selected candidate information	Once formal offer is accepted, starting work date is fixed. Organization-required administrative information is entered and legal employment agreement is created.

Figure 12. Internal and external process relationships



EMPLOYEE'S CAREER DEVELOPMENT

This process creates a roadmap for success, determining future development activities. With CM, focus is made on high-potential employees and leadership-oriented careers.

Figure 8 shows internal and external process relationships.

Best practices key process indicators:

- Employee's development plans defined in HR or global corporate ERP.
- Individual development initiatives managed from ERP results registry .
- HR ERP integration with other relevant processes.
- Employee's potential and preferences registered in ERP.

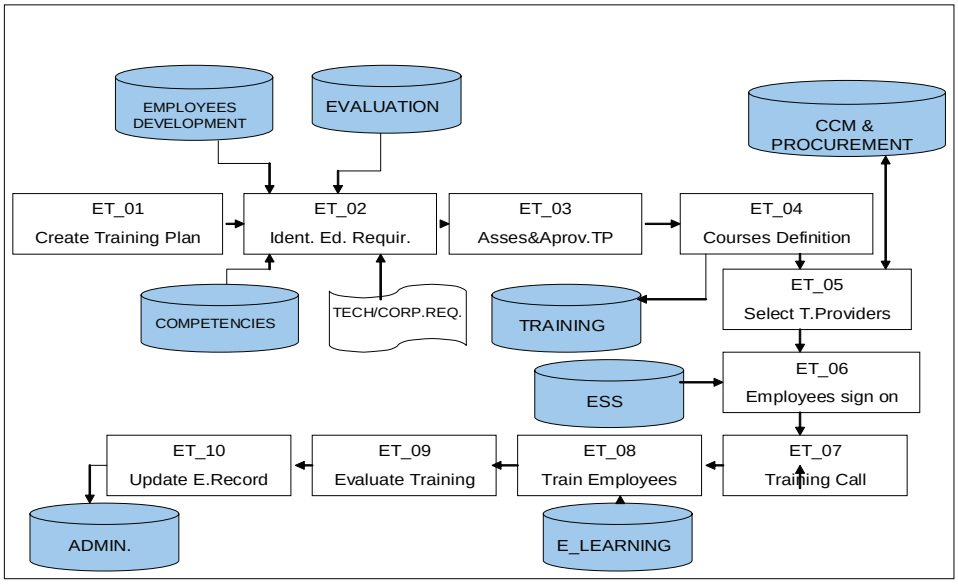
PERFORMANCE MANAGEMENT

Associated to any CM strategy, organizations require employees' daily execution to be aligned to corporate strategy, committed to achieve measurable business results. HR focus is on skills

Table 7. Process reference model for training

Activity	Name	Description
ET_01	Create training plan	A training plan is established every business year. It includes all educational methodologies applied within the organization and is finally approved by executive committee.
ET_02	Identify education requirements	Following inputs are considered: Business units plans. Cross-organization corporate initiatives. Training topics generated by management, derived from evaluation results. HR corporate-defined improvement needs.
ET_03	Training plan assessment and approval	Executive committee approves plan and budget.
ET_04	Courses & training units definition	According to budget, priority, and expected employee usage.
ET_05	Select training and education providers	Internal and external providers, content and methodology evaluation against organization requirements, and cost and budget negotiations through established business-to-business commerce chain management procurement process.
ET_06	Assign employees to training units	Assignment is based upon profiles. E-learning initiatives are usually managed by organization based upon employee's self-initiative.
ET_07	Training call	Arrangements are made according to employees' job schedules, presence, or distant learning method used, instructors' and logistics availability.
ET_08	Perform training	In accordance with plan. HR receives assistance and evaluation report.
ET_09	Training evaluation	Following aspects are evaluated: Quality: content, instructor (by attendance). Student's attitude, knowledge acquisition (by tutor/teacher). Employee contribution to business (after training).
ET_10	Update employee competency record	By HR and management, after complete evaluation.

Figure 13. Internal and external process relationships



tracking, targeting leadership and high-potential employees. It includes skills and competency personal assessment, and, sometimes, 360-degree type of evaluations and feedback.

Best practices key process indicators:

- Evaluation definition and results registry defined in HR or global corporate ERP.
- Individual self-evaluation made via ESS and workflow-based approval process.
- Integration with other relevant ERP modules.

COMPENSATION AND INCENTIVE MANAGEMENT

Compensation is critical to fix adequate wage structures within the organization. Incentive management is required to manage employees' variable retribution and benefits. Complexity varies according to extent and type of benefits (pension plans, private medical services, insurances, etc.). Usually third-party required information tracking and reports are used. Best practices key process indicators:

- ERP-based variable retribution, calculated for employee evaluation.
- ERP-based market salary survey.

Market Applications: The CM Contribution

HCMS is considered today a mature market for the IT industry. Although the administrative, transaction-based type of applications have very limited growth potential (only outsourcing, hosted services in small and medium size business), the strategic talent management-type of applications, driven primarily by *competency management* business interest, is on the rise.

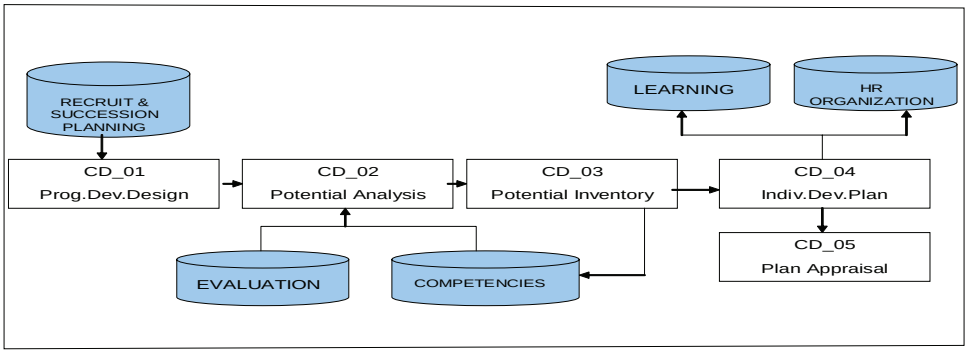
Similarly to other enterprise-wide applications, where process integration becomes essential, the vast majority of medium to large organizations are now automating HC process in one out of three products: SAP, PeopleSoft, or Oracle, and two companies, because Oracle finally acquired PeopleSoft during 2005. Around 40% of the worldwide HCM market is concentrated in these two companies.

This merger and acquisition wave is not an isolated exception (Adonix acquired Meta4, Cy-

Table 8. Process reference model for employee's career development

Activity	Name	Description
CD_01	Career development plan design	Based upon employee's potential and strategic HR CM development objectives (includes target employees segment, criteria to identify potential, application period, etc.). Succession planning information is filtered.
CD_02	Employee potential analysis	Following actions are taken: Identify employees matching profile employee appraisal.
CD_03	Employee's potential inventory	Update HR master employee data with employee's information to ensure that opportunities are considered: job vacancy opportunities, special task assignments, and so forth.
CD_04	Generate personalized development plans	Individual career plans for competency development. Usually based upon employee performance evaluation, training opportunities, special task assignments, etc.
CD_05	Development's plan appraisal	Periodic checks to confirm suitability and compliance to organization's business objectives.

Figure 14. Internal and external process relationships:



borg merged with HR Services, Infinium with SSA, etc.). Microsoft (the world's leading IT corporation but still not an important player in HCMS) and SAP announced merging conversations last year.

Best-of-breed products' previous business preferences are now swiftly moving to the ERP (trough acquisitions). Market fragmentation in HC management is therefore ending, and many small, single-function-oriented products are being incorporated in ERP suites or just disappearing. Maybe the LMS market is the exception to the rule, with an approximately 15% growth expectation in a $450M world market (2005).

It is quite difficult to quantify which percentage of the global HCMS market corresponds just to CM processes. Most organizations enter ERP-based solutions (acquiring licences), automating just part of the functionality. Usually, deployment corresponds to most critical business functionality, with limited CM process automation. Companies invest in ERP suite licences and deploy functionality over the years with the confidence that the ERP vendor will ensure a long process integration.

Worldwide HRMS and payroll administration revenues in 2004 reached around 3.900 M de € with an estimated growth rate for 2005 of 7%. About 30% of the revenue corresponds to payroll and 70% to HR management processes. This survey assumes that, although not all product licences are devoted to comprehensive compe-

Table 9. Process reference model for employee's performance management

Activity	Name	Description
EV_01	Competencies and business objectives definition	Frequently mix of global, business unit, and personal objectives plus competencies and shared values.
EV_02	Self-evaluation	According to person's perceived achievement.
EV_03	Formal evaluation	Performed by management, and employee agrees.
EV_04	Formal approval	HR validates and ensures organization's coherence.
EV_05	Results analysis	Information is obtained to shape next period's educational requirements and assigns variable benefits.

Figure 15. Internal and external process relationships

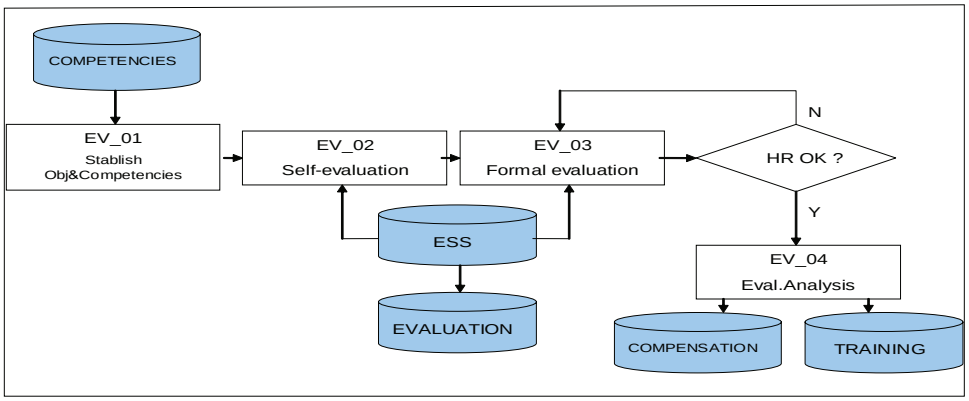


Table 10. Process reference model for employee's compensation and incentive management

Activity	Name	Description
CO_01	Compensation plan	Plan is established according to strategic retribution policy, considering internal and market reference information.
CO_02	Evaluation results analysis	Competency development and business objectives and accomplishments are evaluated.
CO_03	Compensation schema elaboration and review	Based upon results and time schedule to be applied to individuals. HR approves and submit schema for approval.
CO_04	Management approval	Management board approves final plan.

Figure 16. Internal and external process relationships

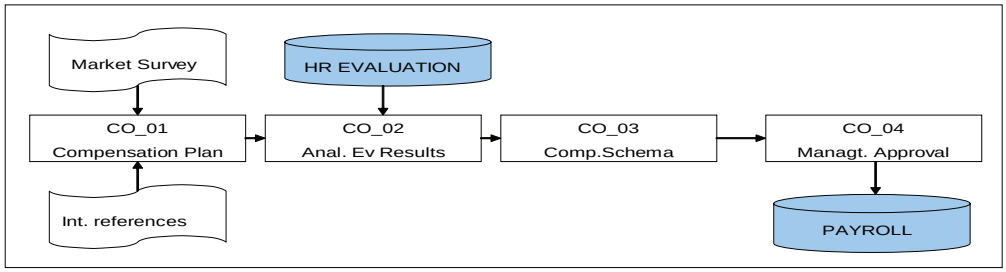
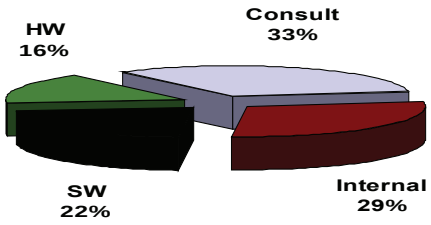
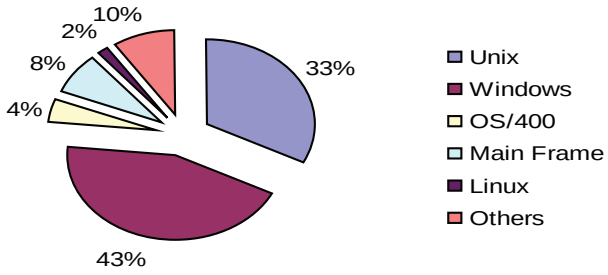


Figure 17. Best practice cost distribution for competency management



Concept	*%*	*Ratio*
Consultancy & Engineering	33%	1,50
Product Licences	22%	1,00
Hardware	16%	0,73

Figure 18. OS Environments in CM Implementations (2005)



tency management deployment strategies, about 40% ($1,300 M) is the market size for HC direct CM-related process.

The estimated figures reported in this study refer only to product licence fees and maintenance costs. Consultancy and implementation engineering costs are not included. In order to obtain overall comprehensive IT-related cost distribution statistics, the following experience-based statistics method may be applied:

Figure 17 shows current OS environment usage for CM-related solutions. Trends indicate a significant increase of Windows 32 and 64b. and reductions in AS400, others, and mainframe environments.

Figure 18 shows that, for each Euro spent in the software product licence, 1.5 Euros are spent on consultancy and engineering services and 0.73 Euros in hardware.

Technical Architecture Reference Characteristics

The next point in this survey shows a generic evaluation model used to asses IT and HR business specialists in selecting the most adequate technology to deploy a competency management strategy in any medium- to large-size organization. As we will see later on, the product's technical architecture is an important issue in this process.

This section depicts the technical architecture to be considered as a reference in CM strategy deployments.

Which are the relevant aspects define a good (or poor) technical architecture?

- Quickly and flexibly.
- Facilitating the introduction of new performance and scalability mechanisms.
- In a true multi-tiered approach, using comprehensive standards-based interfaces:

The client side is basically a user terminal (Web browser, PDA, etc.). Technologies: HTML, JavaScript, WML, Servlets, JSP, and so forth.

The Server side consist on the Server elements, as a Web or WAP Server, that send information to the User Terminal.

The business logic is the core application intelligence that makes the CM logic possible (application server with Java support, EJBs, servlets, etc.).

Data Logic is responsible for data storage and transactional services (DBMS, etc.).

Enhancing SW reusability and development skills specialisation, using component-based software.

Competency Management Market Products

Product solutions vendors that automate CM processes fall in one of these four categories:

- Large integrated ERP suites: These vendors dominate the high-end HCMS market; also competitive in SME Markets. Organizations select these solutions usually to automate more than one corporate business processes (in the areas of financials, CRM, HC management, e-procurement, etc.). Even recognising that these vendors do not always deliver the best solution for any given particular function, their value proposition is founded on *process integration* and *proven, reliable comprehensive process automation*. Most market CM strategy deployment demand comes from large-size organizations to an extent that in the coming years, CM may be considered the most relevant *business driver* that will facilitate HCM market growth.
- SME integrated ERP suites: Similar functional scope to previous high-end ERPs with simpler parametrization options, much faster to deploy, *tailored to SME-type organizations.*
- Standalone HCMS: Large- and SME-driven solutions from vendors exclusively dedicated to payroll & HCMS, not providing support for other corporate business processes (as

Figure 19. Deliver Comprehensive E-Business functionality, across and between Multiple cChannels

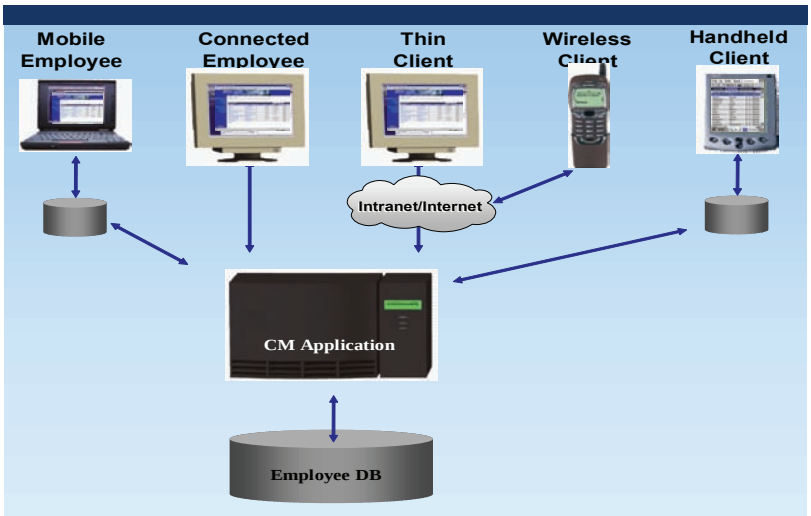
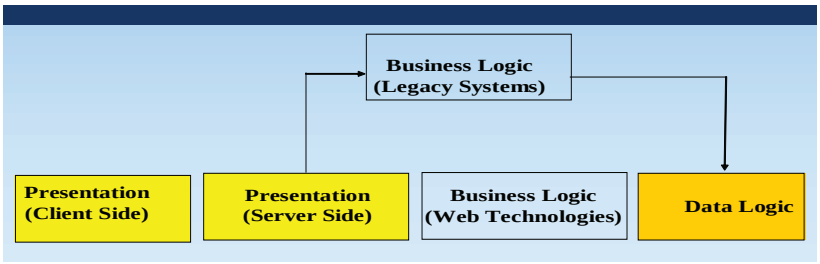


Figure 20. CM Development Architecture



supply chain management, CRM, financials, etc).

- Standalone CM solutions: They automate CM processes, sometimes in combination with performance management and eventually providing support for some other related CM process, in a non-comprehensive, integrated way.

The following part of this chapter shows for each CM product-type category previously indicated, IT- and business-relevant vendor information, evaluating their product offering according to the next proposed evaluation method. Some additional markets are based on four evaluation criteria, as indicated hereafter:

CONCLUSIONS AND OUTLOOK

Following are the most relevant market conclusions and outlook uncovered in this survey:

- HCM and CM application market scope is strong, clearly on the rise. Organizations have to adapt fast to changing, unstable business conditions in a way that CM paper and spreadsheet manual-based processes will have to evolve to a much more flexible, automated employee management-type practice.
- Market preference for ERP-integrated CM solutions instead of best-of-breed standalone products will continue in the future. ERP integration ensures that HR and business managers can efficiently align, measure, and automate individual employee competencies and organizational goals and objectives.
- In this context, IT managers estimate that the integration value provided by global ERP suites far exceeds the benefit of using best-of-breed solutions for each HC business function. The e-business technological transformation and the processes model depicted in this survey clearly show the inter processes and software dependencies required in CM deployment strategies.
- And again, the fact that the trend has been widely adopted creates additional confidence in large vendors' ability to deliver HC-proven process automation. This loop affects CM implementations, where process automation is critical. In some particular CM tasks (like competency assessment), process efficiency, when adopting a stand-alone option, increases fairly moderately compared to the asset they replace. This situation also increases pressure in IT managers to adopt the comprehensive, integrated ERP suite way.
- Large, multinational companies require comprehensive multi-region, often centralized, HC deployment solutions. Global processes and local country localizations makes the three large ERP solutions analysed in this survey the strongest CM option in the future for this type of organization, thus

Table 11.

Evaluation Criteria	Key Factors	Description
COMPANY	Financial stability strategy and vision Partners and alliances Research and development Professional services	Vendors with high potential to survive in the IT market and strong product maintenance and support are highly rated.
MARKET PRESENCE	Product current and previous versions References CM functionality references	Evaluate product stability, installed base and market credentials in previous implementations.
FUNCTIONALITY	Competencies definition Competencies diagnosis Competencies development Competencies monitoring Rest of CM-, HC-related processes	Matching of vendor product standard functionality vs. organizational requirements in current needs and expected required support in the future.
PRODUCT TECHNOLOGY	Technical architecture E-business support Flexibility Integration facilities with required organizations Applications Security compliance	Although e-business engineering practices are widely used, product architecture, scalability, and configuration remain an important differentiator.

displacing fragmented solutions.

- Small and mid-size organizations (less than 4,000 employees) and large, single-country type installations have many more options. According to required functionality, together with associated licence and engineering costs, they may select in the high-end or mid-market type of products, thus accessing a wider menu of options.
- Hosting and outsourcing HRMS services keep growing at a fast pace, particularly in small- to medium-size market companies. This trend represents an intelligent business option for small best-of-breed vendors, having experienced competency assessment experts and staff (apart from being acquired or just disappearing). Some product vendors, apart from the traditional, large-size HRMS service providers (i.e., ADP), are successfully, adopting this delivery model(i.e., Ultimate Software, Ceridian, etc.).

REFERENCES

Alle, V. (1997). *The knowledge evolution: Expanding organizational intelligence*. Boston: Butterworth-Heinemann.

Benson, S., & Standing, C. (2001). Effective knowledge management: Knowledge, thinking and the personal corporate knowledge nexus problem. *Information Systems Frontiers, 3*(2), 227-238.

Blain, J., & Dodd, B. (ASAP Internacional Group) (1999). *Administering SAP R/3: The HR-human resources module, QUE*. Indiana.

Boxall, P., & Purcell, J. (2003). *Strategy and human resource management*. New York: Palgrave Macmillan.

Curtis, B., Hefley, W. E., & Miller, S. A. (2001). *People capability maturity model (P-CMM), (Version 2.0)* (Tech. Rep. No. CMU/SEI-2001-

Table 12. Large Integrated ERP Suites Software Products

SAP AG

Evaluation Criteria	Considerations
COMPANY	SAP AG (Waldorf, Germany) is the world's third-largest independent SW company, devoted to enterprise management, and is the world's leading ERP vendor. More than 28,000 clients, 1,500 partners, and 34,000 employees with local support in more than 50 countries. World's leading vendor in combined administration and HC management, with a 650 M € year revenue (2004), second in HC management (560 M €), close behind PeopleSoft, although No. 1 in Europe.
MARKET PRESENCE	SAP started in the 1970s with the first market product to automate financial applications (R/1). Evolution from the first R/1 installations to the present's comprehensive corporate processes (financials, HCMS, CRM, SCM, e-procurement, business warehouse, portals, etc.), applications under R/3, in which HC and CM solutions are integrated, is a history of sound evolution, incorporating worldwide process excellence and real customer migration support in an all-new product version (currently, once a year).
FUNCTIONALITY	All CM-related processes identified in this survey are included in the integrated, SAP R/3 Solution. Its international focus has ensured extensive local regulatory worldwide support for administrative functions (i.e., payroll). Some processes remain rigid to implement (compensation, benefits) compared with best-of-breed solutions, but overall CM associated functionality rating is very good.
PRODUCT TECHNOLOGY	SAP NetWeaver's architectural design comes from a rigid client-server Model, supported on a vendor's proprietary environment (ABAP). Product evolution in the last three years offers a reasonable Java support and interoperability with .NET and IBM WebSphere (J2EE). Development, deployment and execution environments run on its own SAP Web Application Server (WAS), providing combined ABAP, J2EE, and Web services support.

Table 12. Continued

PeopleSoft

Evaluation Criteria	Considerations
COMPANY	Acquired by Oracle in 2005, world's second-largest independent SW company (See Oracle, next company).
MARKET PRESENCE	PeopleSoft's roots within the enterprise management business are in HR, then extended to financials, SCM, CRM, and e-commerce applications. Leader in the US, far behind in European sales due to SAP's dominance and lateness in incorporating country-required regulations in HHRR administrative and payroll processes in some countries.
FUNCTIONALITY	PeopleSoft Enterprise suite **represents the most advanced, integrated process automation solution for HC management on the market**. Last product versions have introduced enhancements in CM management and workforce analytics, integrating financial, transactional HR, and external market data to provide strategic performance management. Workforce rewards manages compensation and retention policies. PeopleSoft's own Balance Scorecard is effectively used to manage HR effectiveness under workforce scorecard module.
PRODUCT TECHNOLOGY	PeopleSoft architecture is a very flexible combination of database, application, Web, and file Server, fully e-business compliant. The PeopleTools Application Server runs the business logic and facilitates access to most client types, using the Internet architecture and allowing interfaces to visual basic-visual C-based user programs.

Table 12. Continued

Oracle

Evaluation Criteria	Considerations
COMPANY	Oracle is the world's second-largest independent SW company. The combined Oracle + PeopleSoft + JDEdwards (previously acquired by PeopleSoft) has made Oracle the world's second-largest business application vendor (behind SAP) with an estimated annual revenue (fiscal year 2004) in this business of 450 M €, and first vendor in HC and competency management-related solutions, largely because of PeopleSoft enterprise market share. Extensive global partner support and country-assigned services in most locations.
MARKET PRESENCE	Oracle's HCMS has 3,000-plus installations in more than 80 countries.
FUNCTIONALITY	Full HCMS process support as part of e-business suite, integrated suite; includes all required functionality to deploy CM-related strategies. Its *Daily Business Intelligence* supports excellent HC analytical service. Performance, compensation, and e-learning management are excellent and efficient CM-related modules.
PRODUCT TECHNOLOGY	Oracle has now three independent suites to manage: its original ERP, PeopleSoft Enterprise, and JD Edwards' Enterprise One/World. The biggest challenge for Oracle is to ensure a product strategy that combines strengths from all different products, delivering customers a reliable and affordable evolutionary roadmap from its current products' diversity. During the next two to three years, it is expected that Fusion Oracle's applications will incorporate many of the current features from PeopleSoft Enterprise.

Table 12. Continued

Lawson Software

Evaluation Criteria	Considerations
COMPANY	Started operations in the U.S. in 1975 in the corporate analytics and management applications market to become a global ERP solution provider, with particular focus on financials, HR, and procurement. Its business growth strategy has been based on best-of-breed product acquisitions (Account4, ijob, Armature, etc.) and most recently (June 2005), Intentia (Swedish leading SW company, small-size organizations ERP vendor), to create a 3,500-employee company, with 4,000 customers distributed in 40 countries. Large U.S. technology partners list but lacks international, long-term local support and consulting partners outside U.S. Lawson software sales in HC- and CM-related business reached 44 M € in 2005 with financial growth difficulties in last three years.
MARKET PRESENCE	Until recently, business was focused in the United States, Canada, and the U.K. markets. Once merging with Intentia concludes sometime next year, more European technical support may be expected. Product stability and roadmap evolution needs to be ensured in following years.
FUNCTIONALITY	Offers HR suite as part of Lawson. Insight series application includes some CM-type functionality within its personnel administration module but lacks a comprehensive CM functionality (poor performance management, no e-learning part). Strong workforce analytics that compare internal metrics with external sources (provided by Saratoga) in areas such as compensation, staffing, and organizational effectiveness.
PRODUCT TECHNOLOGY	E-business compliant, Web-based architecture, with interesting software extension tools to facilitate integration with other applications (business component integrator).

Table 13. SME-Integrated Suites

Northgate Information Solutions

Evaluation Criteria	Considerations
COMPANY	U.K.-based software applications and outsourcing services vendor. Created in 1969, has become leading U.K. HR and payroll supplier in the U.K. trough, small and leading state-of-the-art companies (Prolog, PWA, CaraPeople, Rebus HR Group). Apart from HR, includes SME-driven, ERP-type solutions for financials and CRM. Currently employs 3,300 people, having 5,000-plus SME clients, 58 M € revenue in HC management, mainly in the U.K.; presence in the U.S., Australia, and New Zealand.
MARKET PRESENCE	In the U.K., long history of success in SME organizations, offering comprehensive product licence sales, engineering, and outsourcing services.
FUNCTIONALITY	It offers wide functionality in HR, starting with basic processes and product-based automation. Usually requires extensive specific software development, particularly in areas as workforce management. Includes recruitment, training, benefits, and ESS-type access. Most CM implementation requires product made to measure changes integration with external applications.
PRODUCT TECHNOLOGY	Open, scalable, Web-based, non-integrated, standard ERP-type of architecture.

Table 13. Continued

Microsoft Business Solutions

Evaluation Criteria	Considerations
COMPANY	Microsoft is the world's largest SW company, providing products and services to individuals and organizations around the globe. Its business solutions division (Microsoft Dynamics) is an international provider of integrated solutions for most SME functional areas. Microsoft entered the ERP market through acquisition of small leading market vendors (Great Plains, Solomon Applications, Navision, etc.). The most significant ones in HRMS are Navision and Axapta, the latest one incorporating an advanced competency management solution for SME.
MARKET PRESENCE	Microsoft's market share in HCMS is low (30 M €, representing just a mere 1.3 % of the world's market) in relation to company profile and business market size, mainly because of current SME-driven strategy, until now. In this segment, implementations history is consistent with good customer references, although each product within the family (Axapta, Navision, Great Plains) follows individual development road maps.
FUNCTIONALITY	Microsoft Axapta is easy to implement and operate with integrated HCMS (plus financials, CRM, project management, etc.) in SME-type organizations. Excellent CM functionality, driven to detect training requirements among individual employees, as well as career plans and organizations' re-structuring functions.
PRODUCT TECHNOLOGY	Open, scalable Web-based architecture .NET and Java, e-business-compliant solutions.

Table 14. Standalone HRMS Products

Kronos

Evaluation Criteria	Considerations
COMPANY	Kronos incorporated began operations in 1977 as a hardware time-clock vendor and has successfully evolved into an HR payroll and management company, reaching $519 M in sales in fiscal year 2005. **It's the world's third-largest vendor in HC management** (first non-global ERP provider), just behind SAP and Oracle, with total 2004 sales of 210M €. About 2,900 employees, offering comprehensive licences sales, consulting, implementation and learning services around the company's HR products. Included in Forbes' "200 Best Small Companies."
MARKET PRESENCE	An excellent record of customer retention through service satisfaction, combined with visionary product evolution from a hardware to the most dynamic software HC management business, sometimes by acquisitions (i.e., AdOpt, excellent workforce planning module). Business focus is driven primarily to the U.S., and some customers are in Canada and U.K.
FUNCTIONALITY	Kronos offers a comprehensive HR administration and management solution for SME and some large-size organizations, including most relevant CM-related processes. Probably the most successful WF management solution in the U.S. market. Its workforce central suite includes core CM functionality. Focus in aligning skills and competencies to corporate objectives, tracking skills, and certifications, and competency enhancements trough training initiatives.
PRODUCT TECHNOLOGY	Open, scalable Web-based architecture. Deploy Java applets to facilitate dynamic user interfaces, as well as HTML access.

Table 14. Continued

Meta4

Evaluation Criteria	Considerations
COMPANY	Spanish-based company, founded in 1991, acquired by Adonix in November 2004, strengthening previous company's financial situation and clearing future market development. In November 2005, the Sage Group Plc announced the acquisition of Adonix under an operation that explicitly excludes the business of Meta4 (leaving control of French entrepreneur Emile Hamou).
MARKET PRESENCE	Large and medium-size extensive customer list (1000-plus) in 20 countries. Leading HC, CM market product in Spain with a strong competitive position in other European markets (France, Portugal, U.K.) and South America, with implementations in large as well as mid-size organizations. In October 2005, Meta4 reopened operations in the U.S., supporting its new operations for the Americas in Miami, through partnership with independent HR consultant Vision 3. Extensive technology, consultancy, and service, outsourcing providers in operational Markets.
FUNCTIONALITY	Meta4 PeopleNet/MindSet suite has an excellent administrative process functionality, except in benefits administration, and a powerful competency management strategy and set of tools, particularly competency assessment and performance management. Full CM-related processes automation, including employee/manager self-service and comprehensive integrated knowledge management environment.
PRODUCT TECHNOLOGY	State-of-the-art e-business technology design and implementation. 3-Tears components-based architecture: Supports pure HTML, Java (applets), and Windows (Active X) Clients, application server and DB Server.

Table 14. Continued

Ultimate Software

Evaluation Criteria	Considerations
COMPANY	Ultimate Software is a successful payroll and HCMS dedicated company, expanding its business from a mere licence sales vendor to an outsourcing service provider, reaching 37 M € in fiscal year 2005 in HC management sales. Currently employs 500 professionals and has about 1,200 customers, mainly in the US. HROA (Human Resources Outsourcing Association) 2005 Provider of the Year.
MARKET PRESENCE	An excellent record of customer satisfaction, combined with visionary product to service business orientation. Business focus is driven primarily to the U.S., and some customers are from Canada and U.K.
FUNCTIONALITY	Its Ultipro Workforce Management Suite incorporates most required CM required process automation functions.
PRODUCT TECHNOLOGY	Open, scalable Web-based architecture. Eamed certification from HR-XML Consortium in 2003, creating HR-specific XML vocabularies.

Table 15. Standalone CM Products

Mindsolve

Evaluation Criteria	Considerations
COMPANY	Fast-growing company created in 1994, exclusively dedicated to the employee **performance management** business, licensing its Mind Solve Visual Product and providing associated required consultancy services. Last year business revenue estimated at 13 M €.
MARKET PRESENCE	Niche player, with consolidated HR expertise support to deploy PM functionality in heterogeneous U.S.-based (some outside US) medium- and large-size organizations. Product licence sales and MVP Express suite, tailored for outsourcing, easy to deploy Web-based solution.
FUNCTIONALITY	Includes end-to-end performance appraisals, 360-degree performance assessment, competency alignment and accountability, and talent and development planning. Skills and competency management support, requires interfaces with other external tools for full CM strategy deployment (e-learning, workforce management, etc.).
PRODUCT TECHNOLOGY	Open, scalable Web-based architecture. It uses proprietary technology (MindSolve's Visual Profiler™) of unique drag-and-drop system to deliver accurate, trustable information. Visual Profiler's ability to comparatively display multiple employees on a single screen produces ratings that better differentiate superior, good, and unacceptable performers.

Table 15. Continued

Geo Learning

Evaluation Criteria	Considerations
COMPANY	Geo Learning is an SME-type company, U.S. based, **dedicated to e-learning** and associated educational activities. It sells learning products (LMS, authoring tools, etc.) and other stand-alone SW products, as its *competency plus* CM tool. Started business in 2000; last 3 years, annual growth rate of 125%; has more than 300 SME-type clients, mainly in government and financial services.
MARKET PRESENCE	Good record of customer base. Works with small reseller and systems integrators partners.
FUNCTIONALITY	Exxceed competency plus is a stand-alone CM product that includes competency modeling (using profiles matching jobs, projects, etc.), career management (job profiles matching employee's competencies), individual development planning (based on competency gaps, assigning learning objects, imported from external LMS), and comprehensive 360 competency assessment process. The foundation of Exxceed functionality is based upon a competency dictionary wherein individual competencies are stored and organized in the form of families, identifying assessable behaviours linked to competencies. Typically records and employee information is obtained using crystal reports or other Web-based reporting tools.
PRODUCT TECHNOLOGY	Microsoft, PC, SQL-type application, thin client Web-based architecture. Low cost and maintenance. Delivered with preconfigured competency DB (10,000 competencies in areas such as business, management, engineering, etc, and 400-plus industry predefined job profiles).

MM-01). Pittsburgh, Carnegie Mellon University, The Software Engineering Institute.

Friss de Kereki, I., Azpiazu, J., & Silva, A. (2004). Knowledge management in learning environmental design. In *Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference*, Savannah.

Hardless, C. (2005). *Designing competence development systems.* Unpublished doctoral thesis, University of Göteborg, Göteborg, Sweden.

Hartman, A., & Sifonis, J. (2000). *Net ready. Strategies for success in the e-conomy.* McGraw-Hill.

Kamara, J. M., Anumbad, C. J., & Carrillo, P. M. (2002). A clever approach to selecting a knowledge management system. *International Journal of Project Management, 20*(3), 205-211.

Levy-Leboyer, C. (1997). Gestión de las competencias. Cómo analizarlas, cómo evaluarlas, cómo desarrollarlas, ediciones gestión 2000. Barcelona: SA.

Lindgren, R. (2005). Adopting competence systems in fast growing knowledge intensive organizations. *Journal of Information & Knowledge Management, 4,* 1-13.

Lindgren, R., & Stenmark, D. (2002). Designing competence systems: Towards interest-activated technology. *Scandinavian Journal of Information Systems, 14,* 19-35.

Means, G., & Schneider, D. (2000). *Meta-capitalism.* The e-business revolution and the design of companies.

Nordstrom, A. K., & Ridderstrale, J. (2000). *Funky business. Talent makes capital dance.* Stockholm, Sweden: Book House Publishing.and Markets in the XXI century. Pricewaterhouse Coopers, Deusto Ediciones.

Rollett, H. (2003). *Knowledge management processes and technologies.* Boston: Kluver Academic Publishers.

Sagi-Vela, L. (2004). Gestión por competencias. *El reto compartido del crecimiento personal y de la organización.* Madrid: ESIC Editorial, Pozuelo de Alarcón.

The Hay Group. (1996). *Las competencias: Clave para una gestión integrada de los recursos humanos.* Bilbao, Spain: Ediciones Deusto SA.

This work was previously published in Competencies in Organizational E-Learning: Concepts and Tools, edited by M. Sicilia, pp. 41-82, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 6.6
Becoming a Learning Organization in the Software Industry:
Is CMM the Silver Bullet?

Dev K. Dutta
Richard Ivey School of Business, The University of Western Ontario, Canada

ABSTRACT

This chapter examines to what extent the implementation of Software Engineering Institute's Capability Maturity Model (CMM) of software process improvement enables a firm to transform itself into an learning organization (LO). It argues that even though the CMM does lead the software firm forward on the route to learning, it does not go far enough. By recognizing organizational knowledge and organizational learning as the twin pillars of the LO, the author develops a conceptual framework against which the five maturity levels of CMM can be mapped and examined. This allows for an assessment of whether the CMM serves as a silver bullet in achieving the software firm's goal of reaching the visionary state of the LO.

INTRODUCTION

Today, across the world, business firms are exposed to increased environmental turbulence and uncertainty. There is rapid change in technology and its usage, and competition in the marketplace has intensified, with customers becoming highly knowledgeable and demanding. With economic realities and priorities shifting constantly, there is now the emergence of a new global economic order in which knowledge or intellectual capital—rather than labor, machine power and capital—constitutes the most critical factor of production as well as a source of competitive advantage (Zack, 1999). Nowhere is this more evident than in the software industry. By its very nature, a firm engaged in developing software applications as its primary product shares all the features of what Alvesson terms as "knowledge-intensive" firms. These firms depict the following characteristics:

1. "Highly qualified individuals doing knowledge-based work, using intellectual and symbolic skills in work;
2. A fairly high degree of autonomy and the downplaying of organizational hierarchy;
3. The use of adaptable, ad hoc organizational forms;

4. The need for extensive communication for coordination and problem-solving;
5. Idiosyncratic client services;
6. Information and power asymmetry (often favoring the professional over the client);
7. Subjective and uncertain quality assessment." (2004, p. 21)

To survive in such a turbulent business environment and achieve global standards with respect to quality, cost and customer expectations, a software firm must not only treat knowledge as its most critical resource but also learn to be highly adaptive in everything it does with the knowledge. It must proactively anticipate emerging trends and directions with regard to the business environment, customers and technology. It must assimilate the knowledge and use it effectively to best meet the customer requirements. Therefore, the software firm must work towards building for itself an all-pervasive learning culture. It must become what is termed as an LO (Senge, 1990).

The CMM, developed by the Software Engineering Institute (SEI) at Carnegie Mellon University, is suggested to be a step in this direction (Levine, 2001). The CMM enables firms to view software development as an engineering discipline and ensure its progression from being an immature, ad-hoc process to a mature, managed process (Paulk, 1998a). Ramanujan and Kesh (2004) note that the last few years have seen a significant investment on the part of software firms to implement the CMM; in most cases, firms also a report a spectacular improvement in financial performance after they have gone through the implementation process. However, the question that remains unanswered is, how sustainable is this improvement effort? That is, does adoption of the CMM enable the software firm into becoming an LO? The aim of this chapter is to investigate this research question. This is important because after the initial fanfare associated with the CMM has died down, the initiative will continue to prove useful only if it enables a software firm to develop an organization-wide learning culture and derive sustainable competitive advantage based on knowledge.

ESSENTIAL CHARACTERISTICS OF THE LO

The idea of an LO is difficult to grasp. This is because not only is it is far from a homogeneous concept but also because the terms "LO" and "organizational learning" are used interchangeably in literature. In tracing the concept of organizations as learning systems, Yeung, Ulrich, Nason and von Glinow (1999) identify eight properties that they suggest constitute the "basics" of LOs: (1) they focus not only on learning but also in meeting organizational goals; (2) they follow a systems logic and engage in out-of-the-box thinking; (3) they build upon but are not limited by individual learning; (4) they follow the learning continuum that stretches from superficial to substantial; (5) they recognize that learning comes from many small failures; (6) they adopt a process approach to learning, understanding that learning often evolves along a predictable set of processes; (7) they give cognizance to both direct experience and vicarious experience as being useful inputs to learning; and (8) they treat learning as being important, not only for "exploiting" existing opportunities but also for "exploring" new opportunities. This suggests that the LO is an ideal or visionary state that firms aspire to reach and one that requires them to engage in transformational, organization-wide, real-time learning.

Senge defines the LO as one where "people continually expand their capacity to create the results they truly desire, where new and expansive patterns are nurtured, where collective aspiration is set free and where people are continually learning how to learn together" (1990, p. 17). Even though this gives an idea of the philosophy behind an LO, Mumford provides a definition that is more practice oriented: "The LO is one that

creates an environment where the behaviors and practices involved in continuous development are actively encouraged" (1995, p. 12). In turn, then, OL refers to "a variety of practices and values that enable a company to explore continually new directions and anticipate, or even lead, change in the marketplace and in society at large" (EIU & IBM, 1996, p. 11).

Even though the LO appears to be an ideal or visionary state that a firm endeavours to reach, it is clear that its foundations rest on two specific organizational characteristics: (1) organizational knowledge, and (2) organizational learning. The former denotes the unique resource available to firms to use in order to progress. It can usefully be represented as the *content* aspect of the LO. Similarly, the latter can be referred to as the *process* aspect of the LO; that is, the specific mechanisms that actually allow learning to happen. In the next few paragraphs, I examine these two pillars of the LO and develop a theoretical framework that elaborates on how and when a firm truly becomes one.

Organizational Knowledge

Organizational knowledge is not static. It is created within the firm as a continuous dialog between its constituents; that is, explicit and tacit forms of knowledge available with employees and within organizational artefacts. Though such an exchange arises primarily among individuals, it is the organization that creates the facilitating mechanisms that lead to emergence of dialog. According to Nonaka (1994), the organization manages the four constituent processes of dialog: socialization (creating tacit knowledge through sharing of tacit experience), combination (creating explicit knowledge by combining different explicit knowledge), externalization (conversion of tacit knowledge into explicit knowledge) and internalization (converting explicit knowledge back into tacit knowledge). When all four modes of knowledge sharing occur as a continual shift, organizational knowledge is generated (Nonaka, 1994). Thus, close cooperation among organizational members is necessary for organizational knowledge to develop. This view endorses that organizations are social entities populated by individuals who have an interest and desire to share and learn from each other, and it is through the organizational platform that knowledge and expertise of individuals get converted into economic products and services that have an economic value in the marketplace (Kogut & Zander, 1992). In other words, it is in the organizational space that situated learning occurs between individuals, and which leads to a dynamic evolution of organizational knowledge. This is what is also called the distributed knowledge system (Tsoukas, 1996), where individual knowledge is constantly getting merged, shaped and reshaped as an organizational resource.

The firm's base of explicit knowledge can be expanded in several ways. For example, the firm's current knowledge position may be benchmarked vis-à-vis what are viewed as world-class practices; the firm can also focus on learning as a visible and central element of its strategic intent (by making explicit mention of it in the mission or vision statement); also, the firm can actually go for concrete investments in learning (through adoption of "hard" mechanisms, such as sponsoring employees to training programs, or through "soft" mechanisms, such as devising creative ways and means of employee feedback and suggestions, and by sharing best practices across vertical/horizontal divisions within the firm). Organizational learning does not happen by chance; nor does it happen overnight. Thus, LOs must exhibit "a purposeful learning approach designed to create knowledge and translate it into effective action" (Bohlin & Brenner, 1996, p. 2). By experimenting with collective learning, most firms engage in work practices that generate "ideas with impact." This is possible through continuous improvement, competence acquisition, experimentation and boundary spanning. Accord-

ing to Ulrich, Von Glinow and Jick, "combining the ability to learn by going outside a business' boundaries (i.e., boundary spanning), coupled with a culture focused on internal management processes such as empowerment and teamwork, is most conducive to competitiveness" (1993, p. 64). Firms also generalize ideas that have impact by creating an organizational infrastructure that moves ideas across boundaries. This can be through building up organizational competence (staffing, training and organizational development), management action (appraisal, rewards), governance (management style, policies, organization design, communication, feedback) and work processes (systems, processes, teams).

However, given that the firm competes in a radically shifting and turbulent environment, the knowledge it draws upon is not only inherently indeterminate but also continually emerging (Tsoukas, 1996). This requires that knowledge be viewed as being more than simply tacit or explicit or a sum of the two. Rather, it brings into focus an alternate view of knowledge as "knowing." Cook and Brown (1999) have suggested that if the former (i.e., knowledge) constitutes an "epistemology of possession" (with knowledge being treated as something people possess), the latter (i.e., knowing) calls for an "epistemology of practice" (where the focus is more on people knowing). The authors further suggest that knowledge and knowing are not competing but mutually enabling; together, they constitute a "generative dance" that leads to continuous innovation on the part of the firm. Orlikowski (2002) notes that "knowing in practice" is a social activity that has as its core several important attributes, such as sharing identity, interacting face-to-face, coordinating across time and space, learning by doing and supporting participation. Similarly, using a structuration perspective, Hargadon and Fanelli (2002) suggest that organizational knowledge arises as a recursive outcome of interaction between knowledge as action and knowledge as possibility. In that sense, by its very nature, knowledge is uncertain and never completely revealed.

Organizational Learning

Organizational learning has been defined as the process by which: (1) the organizational knowledge base is developed (Shrivastava, 1983), (2) organizational action is improved through better knowledge and understanding (Fiol & Lyles, 1985),(3) organizational inter-subjective meaning is changed (Cook & Yanow, 1993), or(4) change is effected in individual and shared thought and action (Crossan, Lane & White, 1998). Thus, by its very nature it is suggested to effect both cognitive and behavioral change among organizational members. In a general sense, the cognitive perspective regards organizational learning to be the summation of learning of individuals. It suggests the organization is a cognitive or information processing system (March & Simon, 1958). Any requirement of learning or strategic change is viewed in terms of cognitive maps of managers and employees, as a gap between actual and desired results measured in terms of business performance. Knowledge is taken to be primarily resident in the minds of individuals and from where it can be extracted, packed and stored in repositories for future use or transferred to others (Nidumolu, Subramani & Aldrich, 2001). In other words, the cognitive view takes learning to be primarily an individual activity separate from and, in some sense, even opposed to other activities individuals engage in within organizations (Gherardi, Nicolini & Odella, 1998). However, such an extreme position also seems to suggest that viewing organizational learning exclusively based on the cognitive perspective brings about its own inadequacies.

Though Kolb's (1979) cyclic model of individual learning (concrete experience, reflective observation, abstract conceptualization and active experimentation) has been applied in the organizational setting (Carlsson & Martin, 1976), individual learning forms only a necessary but not a sufficient condition for organizational learning. Huysman (1999) notes that current literature on organizational learning suffers from several

biases: (1) it tends to treat the individual as the actor in learning; (2) it considers environmental adaptation as the primary motivator for why organizations learn; (3) it suggests that learning primarily is a planning activity; and (4) it focuses on improvement as the only expected result of learning. While organizational learning necessarily must incorporate these elements, the question that arises is whether fulfilling these conditions will transform the firm into a LO. It is impossible to see cognition occurring at the level of the organization (Cook & Yanow, 1993). Also, most of the time, organizational members cannot agree on common cognitive models to interpret information or take action (Mirvis, 1996). Even if all individuals in the firm engage in learning, does it ensure learning at the organizational level? Again, does planning necessarily ensure optimal learning?

To mitigate these difficulties, researchers have proposed an alternate view of organizational learning based on the social, behavioral cultural and interpretive characteristics of organizations, with the recognition that context is an important parameter in learning (Fiol & Lyles, 1985; Daft & Huber, 1987). Under this perspective, organizational learning occurs when a group of learners (as social beings) engage in joint construction of reality and shared meaning-making, based on social interaction within specific socio-cultural settings (Gherardi et al., 1998; Miner & Mezias, 1996; Nicolini & Mezner, 1995). Thus, learning is a process that is socially constructed and "… focuses on the way people make sense of their experiences at work" (Easterby-Smith & Araujo, 1999, p. 4). When the individual learns, learning gets linked to changes in an individual's interpretation of events and action (Daft & Weick, 1984). Proponents of the social view of learning suggest that individual knowledge is possible only because of the social practices individuals find themselves in (Tsoukas, 1996). Therefore, the social view on learning not only endorses the active participation of individuals in the joint sense of making and learning, but also suggests the extremely important role context plays in this endeavor. The idea of socially constructed knowledge recognizes it as closely following a socio-historical context, made available through the everyday experience of individuals (McAdam & McCreedy, 1999). An ongoing, circular interaction between individually held latent knowledge and the knowledge manifest in the surrounding environment is what enables organizational knowledge to emerge (Hargadon & Fanelli, 2002). Specific mechanisms that may aid in this process of interaction include situational factors, such as the unit's tasks (process or content orientation) and domain of learning (focused or broad) (Becerra-Fernandez & Sabherwal, 2001).

In fact, Lave and Wenger (1991) go a step further. They suggest that not only is learning *situated* in practice, it is in fact hidden from the other regular mechanisms of the organization. Therefore, studying it systematically is extremely difficult (Brown, Collins & Duguid, 1989; Richter, 1998). According to the situated or social learning view, the learning context and relationship of the learners are as important as learning itself. As new members get inducted into the process of collaborative learning and meaning-making, individual schemata, scripts and beliefs get institutionalized into organizational knowledge structures (Cook & Yanow, 1993; Daft & Weick, 1984; Schank & Abelson, 1977; Weick & Bougon, 1986).

At the same time, apart from the individuals engaging in knowledge sharing and exchange, the team/group as well as the whole organization are influenced by organizational knowledge structures. Socialization of individuals in the learning process helps in the dynamic transfer of knowledge from the individual to the organization, as described in Nonaka's knowledge spiral (Nonaka &Takeuchi, 1995). Following this process, the knowledge structure and form of the organization also undergoes changes. It becomes a product of complex relations and interaction between individual beliefs and perceptions and organizational routines (Garud & Rappa, 1994), thus creating an appropriate collective for knowl-

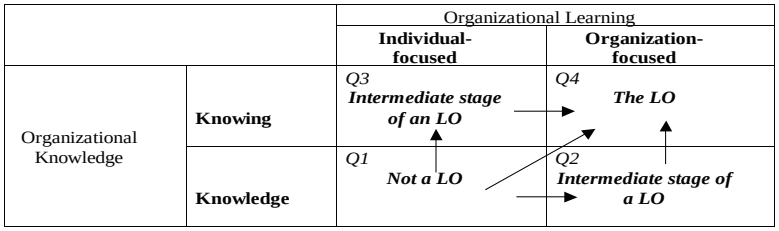
edge sharing across the community of practice (Nonaka & Konno, 1998). Crossan, Lane, and White (1999) suggest that the core processes that constitute organizational learning can be depicted as the 4I (Intuiting, Interpreting, Integrating and Institutionalizing), with intuiting and interpreting focused on the individual level, interpreting and integrating focused on the group level, and integrating and institutionalizing focused on the organizational level. The authors further suggest that a system of feed-forward and feedback learning flows characterizes a back-and-forth flow of knowledge across each of these three levels.

In view of the preceding discussion, I suggest that each of the two pillars of the LO—organizational knowledge and organizational learning—can be usefully looked upon as captured in two distinct states. For organizational knowledge, this state varies from "knowledge" to "knowing." Similarly, for organizational learning, the states vary from "individual-focused" learning to "organization-focused" learning. By identifying each of these two states of organizational knowledge and organizational learning, I arrive at a 2x2 framework that describes the different stages of a LO. This is depicted in Figure 1. Quadrants 1, 2 and 3 are intermediate stages that the firm finds itself in during its journey towards an LO. In my depiction, a true LO comes into existence in Quadrant 4, where the focus is on knowing over knowledge and organizational learning over individual learning. While it is expected that a firm in Q1 would reach Q4 by either traversing through the intermediate stage of either Q2 or Q3, it is possible that it may reach Q4 directly. This rare event would happen under a visionary change agent within the firm, who pulls up the organization from Q1 directly into Q4, thus by-passing the intermediate stage. Finally, it is important to remember that in Figure 1 the categorizations "knowledge" vs. "knowing" and "individual" vs. "organizational" are merely stylized ways of representing the two orthogonal dimensions. In practice, a firm will describe elements of both knowledge and knowing simultaneously, just as it will depict both individual and organizational learning at the same time. Rather, what I suggest is that in each of the four cells, one of the two forms of organizational knowledge focus will predominate, just as one of the two forms of organizational learning will assume greater importance.

Quadrant 1: This quadrant represents firms that focus on organizational knowledge over organizational knowing. In other words, these firms focus on knowledge possession over knowledge practice. While these firms may also accord importance to tacit knowledge in addition to explicit knowledge, they do so with the idea of possession rather than practice. In effect, these firms seem to adopt a rather static view of how knowledge is acquired, exchanged, utilized and updated. Similarly, these firms tend to lay a greater emphasis on explicit knowledge over tacit knowledge. If they do focus on tacit knowledge, they tend to view it as knowledge that is possessed rather than practiced.

Quadrant 2: This quadrant represents firms that give emphasis to organizational learning over

Figure 1. The LO framework



individual learning. Thus, they recognize that organizational learning is more than the sum of learning of individuals. However, by continuing to focus on the idea of knowledge as possession rather than practice, these firms fall short of the target of becoming a true LO.

Quadrant 3: Like Quadrant 2, this quadrant represents firms that are at an intermediate stage in their journey towards becoming an LO. While these firms do focus on knowing—that is, the idea of knowledge as practice rather than as possession—they continue to believe that learning in organizations is simply synonymous with learning efforts of individuals. As such, these organizations miss out on certain important linkages of how individual learning can expand into learning at the organizational level.

Quadrant 4: This quadrant represents firms that not only accord higher emphasis to knowing (i.e., knowledge as practice) over knowledge (i.e., knowledge as possession) but also focus on developing organizational mechanisms, processes and practices that expand individual-level learning into organizational learning. By doing so, these firms transform them into LOs in the true sense of the term.

Having described the essential characteristics of a firm's journey towards becoming an LO based on its understanding of organizational knowledge as well as organizational learning, I now examine to what extent adoption of SEI CMM enables the software firm to achieve this vision of transforming itself into an LO.

THE CAPABILITY MATURITY MODEL OF SOFTWARE DEVELOPMENT

The CMM was developed by SEI to enable software firms to treat the process of software development as an engineering discipline, and develop "reliable and usable software that is delivered on time and within budget … The progression from an immature, unrepeatable software process to a mature, well-managed software process also is described in terms of maturity levels in the model." (Paulk et al., 1993a, pp. 0-1). SEI commenced work on the model in 1986, based on an initial request on the part of the United States (U.S.) federal government, to provide the latter with a method for assessing the capability of its software contractors. The model was first released in 1991. Thereafter, periodic improvements on the model have been continued to be made, based on feedback from organizations that adopted the CMM.

Immature vs. Mature Software Firms

A computer software firm, assessed on the basis of robustness and maturity of the processes used by it to develop software, can be placed along a continuum that progresses from immature to mature. The immature software firm is characterized by an overall ad-hoc, reactive approach to software development. Not only does the firm not have consistently followed processes and procedures for production, testing and quality assurance of software, but whatever processes that may be in place are also thrown to the wind whenever the firm is in the throes of a crisis. In contrast, a mature software organization ushers in a certain consistency and uniformity of approach in the development of software. It possesses established, consistent processes for software development. Planning forms an integral part of the process cycle. Periodic re-evaluation and improvement of currently adopted processes is also institutionalized within the organization. In view of these measures, the mature software tends to produce software that is relatively less prone to defects and errors. To summarize, while immature software firms operate on a reactionary basis, mature software firms display a proactive approach towards managing the processes used to develop software products (Paulk et al., 1993b).

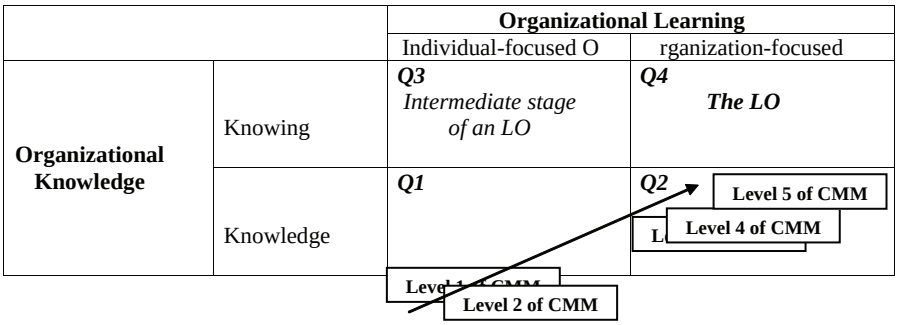
CMM Process Maturity Framework

When it adopts the CMM, the software firm demonstrates a commitment that it intends to move from a software development regime characterized by ad-hoc, immature, reactive processes to one that is disciplined, mature and proactive. The CMM process maturity framework has been conceptualised at five maturity levels (1 to 5), which "define an ordinal scale for measuring the maturity of an organization's software process and for evaluating its software process capability" (Paulk et al., 1993b, p. 7). Each of the five maturity levels of the CMM exhibit certain behavioral attributes. At Level 1, the software process is an amorphous black box that generates a software product when provided with certain specifications; however, it is unable to define or use any process controls whatsoever. This situation improves somewhat in Level 2, where both customer requirements and work products are controlled. At this level, the organizational software process can be visualized as a succession of black boxes that "allows management visibility into the project on defined occasions" (Paulk et al., 1993b, p. 21). Internal structure of the black boxes, the tasks in the project's defined software process, becomes visible in Level 3. This is, therefore, the level at which the software development process is viewed as an organization-wide activity, even though it is broken down into individual projects, and common routines and standards are applied to ensure both process and product quality. Level 4 is a further improvement; processes are instrumented and controlled quantitatively, and ability to predict outcomes and initiate corrective action is high. This culminates in the organization progressing into Level 5, where it initiates a culture of continuous process and product improvement. At this stage, "disciplined change is a way of life" (Paulk et al., 1993b, p. 21). Since each level builds a foundation for the next level, it is preferable for a software firm to progressively go up the maturity hierarchy; skipping levels is counterproductive.

SOFTWARE FIRMS AND CMM IMPLEMENTATION: A JOURNEY TOWARD THE LEARNING ORGANIZATION

Being an organization-wide change initiative based on the principles of total quality management (TQM), adoption of CMM provides an organization with many benefits. Some of the tangible benefits of CMM implementation include "… shorter development cycle, reduced rework, higher development productivity and higher-quality systems" (Duggan, 2004, p.9). The CMM concentrates on developing an organization-wide process architecture that enhances some aspects of both knowledge management and organizational learning through easy acquisition, storage, management and utilization of knowledge on the part of software firms. CMM plays this enabling role

Figure 2. CMM and its overlaps with the LO framework



mainly through transformation of the processes associated with the software development life cycle, making them less person-specific and based more on robust institutional systems, processes and practices. As a result, the firm necessarily progresses from "ad hoc, chaotic processes to mature, disciplined processes … [which reflect a culture of continuous process and product improvement, and where] disciplined change is a way of life" (Paulk et al., 1993b, pp. 3, 21).

In attempting to map each level of the CMM (with its associated challenges and benefits) onto the 2x2 LO framework, I arrive at Figure 2, which is a depiction of the extent to which the CMM actually helps a software firm to become an LO.

Under Levels 1 and 2, the firm continues to focus on individual training and learning, with little effort being taken to bring these processes up to the level of the entire organization as a common unit of analysis. Therefore, firms that are either at Level 1 or have progressed to Level 2 cannot be termed as LO at all. For instance, Level 1 firms may produce quality software from time to time, but if this happens, it is due to the personal initiative and heroics of the firm's managers rather than the firm's process architecture. When a firm has progressed to the stage of Level 2, it has initiated a system of managing development of software by imitating and implementing a repeatable set of software development processes. It is expected that at this level the firm has been able to articulate and share most of its explicit knowledge with other organizational members. However, even here the firm is focused too much on the present, with little efforts being made toward learning from the current set of experiences and applying it the future.

Figure 2 also suggests that Levels 3, 4 and 5 emerge as shifting positions within Quadrant 2. In these successive stages of software process maturity, the firm is clearly focused on adopting the entire organization as the unit of analysis. Thus, when it is in Level 3, the firm has already begun to use a set of standard software processes. In addition, under Level 4 the firm begins to practice a quantitative orientation as far as measuring and managing projects are concerned. Further, it has developed the requisite repositories for knowledge storage and retrieval. Finally, at Level 5 the firm is focused on managing and improving the processes on a proactive basis, with the objectives of defect prevention, innovation, knowledge sharing and dissemination of lessons learned across the organization. It is expected that at this stage there will be interplay of tacit knowledge among members, just as there is high emphasis on explicit knowledge exchange.

Even though the CMM and, especially, upper Levels 3, 4 and 5 make a software firm engage in organizational learning, by its very structure it also acts as a constraint in holding back the firm from becoming a true LO. This is because the CMM essentially over-emphasizes the structural and process aspects of an organization's journey towards becoming the LO and neglects the equally important people aspects. The framework essentially accords importance to the process and capability aspects of learning at the organizational level. It even emphasizes the skills and behavior aspects of learning at the individual level. In doing all this, however, the CMM continues to follow and emphasize an "epistemology of possession" (Cook & Brown, 1999) with respect to knowledge and learning. It is silent on what further must be done to usher in an organizational culture that values an "epistemology of practice" (Cook & Brown, 1999) with regard to knowledge and learning. In fact, the CMM seems to foster a notion within the firm that whatever gets measured is managed. While this is true with regard to many of the structured, relatively simple aspects of organizational life, it is well known from mainstream theories of the organization that this may not completely represent the totality of the firm's existence. Not only is organizational reality highly complex and ambiguous, but also managers are boundedly rational. This makes managers go for a strategy of satisficing rather than optimizing most of the

time. Unfortunately, the CMM does not recognize any scope for ambiguity within its processes.

In his study of what he terms as "high maturity organizations," Paulk (1998b) finds that these firms address several organizational issues that go beyond the scope of the CMM. These relate to developing a culture of openness and communication, commitment to quality, customer responsiveness and other "peopleware" issues. In contrast, the CMM seems to overemphasize what may be termed as the "hard" aspects of management and, in doing so, neglects its "soft" aspects. For example, process and capability mainly denote the structural or harder constituents of learning at the organizational level, whereas leadership and climate denote the visionary, style and cultural or softer constituents. Similarly, at the individual level skills comprise the harder constituent while behaviors and, to a greater extent, values constitute the softer components. The CMM implicitly assumes that the softer prerequisites are given or already existing within the firm. For example, top management vision, foresight and commitment are taken for granted, assumed as being exhibited by the decision to implement the model. Similarly, individual values may be said to exist, in the desire to use the model, achieve error-free software programs and continuous improvement, and thus ensure teamwork, transferability and sharing of knowledge.

According to Elkjaer (2001), a problem that plagued the initial efforts of organization towards becoming an LO related to the ways in which learning was understood. It was supposed that if individual learning was taken care of, organizational learning was bound to result. It is only later that specific concern, such as the assimilation of individual learning into organizational learning, began to be examined. In her study, Levine (2001) finds that learning and a program of adoption of technical change (such as the CMM implementation) mutually reinforce each other. Yet, it can be said that though adoption of CMM takes the software firm forward towards becoming the LO, it does not go far enough. Mathiassen and Pourkomeylian (2003) find that to become effective at managing knowledge, a software process improvement (SPI) initiative (such as the CMM) must strike a balance between personalized and codified aspects. This is because the organization in which information systems development occurs is essentially a socio-technical system in which technical processes combine with social ones and involve stakeholders at multiple levels of the organizational hierarchy (Robey, Welke & Turk, 2001; Sawyer & Guinan, 1998). Therefore, not only does the SPI need to be flexible and amenable to change over time, but it must also maintain a tension between dialectical opposites—codified vs. tacit, individual vs. the organization, informal vs. formal. Unless it is able to do this, the knowledge management efforts engaged in through the SPI will achieve adequate results but certainly not transform the firm into an LO. This is an area where the CMM leaves much to be desired.

Ravichandran and Rai believe that "… many software process improvement frameworks, including the CMM, do not pay adequate attention to the organizational factors that enable or constrain process improvements" (2000, p. 401). In fact, even with respect to knowledge management, which is only a part of the organization's overall journey towards becoming an LO, the CMM has further to go (Ramanujan & Kesh, 2004). Though adoption of the CMM does position the software firm on the road to transforming itself into an LO, to ensure sustainability of collective learning within the firm, its leadership must continually demonstrate in no uncertain terms the commitment to organizational learning. This will need to be reinforced through creation of an organizational climate that fosters innovative behavior on the part of employees and adoption of the essential "epistemology of practice" (Cook & Brown, 1999) based on adoption of an organization-wide enabling value system (trust, spirit of sharing and respect for the individual).

Table 1. SEI-CMM process maturity framework (adapted from Paulk et al., 1993a, 1993b)

Level	Name	Process Qualifier	Behavioral Characterization
1	Initial	Ad hoc	• Unstable internal environment for software development • Planned procedures abandoned during crisis • Quality of output depends on maturity and resilience levels of staff • Schedules, budget, functionality and quality remain unpredictable
2	Repeatable	Disciplined	• Policies established for managing the processes • Repetition of successful practices developed earlier • Basic software management controls exist • Software project standards are defined and adhered to • Software costs, schedules and functionality are tracked and monitored
3	Defined	Standard, consistent	• Standard software engineering/management processes documented and integrated • Adequate training programs imparted to staff to use software engineering concepts • Individual projects use baseline, standard software processes already available with the organization
4	Managed	Predictable	• Productivity and quality standards measured for all important software activities across projects • Organization-wide process database used to collect, store and analyze information on projects • Quantitative measurement standards available to evaluate projects' software processes and products • Meaningful variation in process performance can be distinguished from random variation
5	Optimizing	Continuouslyimproving	• Focus on organization-wide continuous process improvement • Means available for identifying and strengthening process weaknesses proactively • Software engineering innovations identified and adopted across the organization

CONCLUSION

In this chapter, my main aim has been to synthesize the growing body of literature on knowledge and learning with the primarily practitioner-driven framework of software quality assurance and process improvement—that is, the CMM. I have done this to identify whether the CMM process framework indeed acts as a silver bullet in the software firm's quest for becoming an LO. My analysis reveals that even though adoption of the CMM does take the software firm forward on the route to learning, it does not go far enough. Being a quality assurance framework, the CMM mainly concentrates on strengthening the structural or "hard" issues of ensuring collective learning based on predominantly codifiable or explicit knowledge. Therefore, it falls far short of concomitant changes in behavior, beliefs, values and culture that are required of employees if the firm is to truly transform itself into the LO. Future research efforts could concentrate on exploring how to build these "softer" concepts of the LO into the CMM.

This discussion on explaining to what extent the CMM transforms a software firm into an LO has several limitations. I have restricted the scope of the discussion to include only the CMM. It must be recognized that there are other associated/variant models also available with the SEI and adopted by software firms—for example, the People CMM, CMMI, PSP and IDEAL. However, these are outside the scope of the present discussion. To conclude, it is fair to suggest that even though the CMM has become the most popular TQM framework currently being adopted across the software industry, it has certain inherent limitations that do not take the software firm adopting the CMM to go far enough on the road to becoming an LO. The quest for becoming an LO, thus, continues to remain a journey along an arduous path, and one that the software organization must traverse with a great deal of foresight, experimentation and humility.

REFERENCES

Alvesson, M. (2004). *Knowledge work and knowledge-intensive firms.* New York: Oxford University Press.

Becerra-Fernandez, I., & Sabherwal, R. (2001). Organizational knowledge management: A contingency perspective. *Journal of Management Information Systems, 18*(1), 23-55.

Bohlin, N.H., & Brenner, P. (1996). The LO journey: Assessing and valuing progress. *The Systems Thinker, 7*(5), 1-5.

Brown, J.S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher, Jan-Feb,* 32-42.

Carlson, B.P., & Martin, J.B. (1976). R&D organizations as learning systems. *Sloan Management Review, 17*(3), 1-15.

Cook, S.D.N., & Brown, J.S. (1999). Bridging epistemologies: The generative dance between organizational knowledge and organizational knowing. *Organization Science, 10*(4), 381-400.

Cook, S.D.N., & Yanow, D. (1993). Culture and organizational learning. *Journal of Management Inquiry, 2*(4), 373-390.

Crossan, M., Lane, H., & White, R. (1998). *Organizational learning: Toward a theory* (working paper). London: The University of Western Ontario.

Crossan, M., Lane, H., & White, R. (1999). An organizational learning framework: From intuition to institution. *Academy of Management Review, 24,* 522-538.

Daft, R.L., & Huber, G. (1987). How organizations learn: A communications framework. *Research in the Sociology of Organizations, 5*(2), 1-36.

Daft, R.L., & Weick, K.E. (1984). Towards a model of organizations as interpretive systems. *Academy of Management Review, 9*(2), 284-295.

Duggan, E.W. (2004). Silver pellets for improving software quality. *Information Resources Management Journal, 17*(2), 1-21.

Easterby-Smith, M., & Araujo, L. (1999). Organizational learning: Current debates and opportunities. In M. Easterby-Smith, L. Araujo & J. Burgoyne (Eds.), *Organizational learning and the LO.* Thousand Oaks: Sage Publications.

EIU & IBM. (1996). *The LO: Managing knowledge for business success.* Research Report of the Economist Intelligence Unit in cooperation with the IBM Consulting Group.

Elkjaer, B. (2001). The LO: An undelivered promise. *Management Learning, 32*(4), 437-452.

Fiol, C.M., & Lyles, M.A. (1985). Organizational learning. *Academy of Management Review, 10*, 803-813.

Garud, R., & Rappa, M. (1994). A socio-cognitive model of technology evolution: The case of cochlear implants. *Organization Science, 5*(3), 344-362.

Gherardi, S., Nicolini, D., & Odella, F. (1998). Toward a social understanding of how people learn in organizations: The notion of situated curriculum. *Management Learning, 29*(3), 273-297.

Hargadon, A., & Fanelli, R. (2002). Action and possibility: Reconciling dual perspectives of knowledge in organizations. *Organization Science, 13*(3), 290-302.

Huysman, M. (1999). Balancing biases: A critical review of the literature on organizational learning. In M. Easterby-Smith, L. Araujo, & J. Burgoyne (Eds.), *Organizational learning and the LO.* Thousand Oaks: Sage Publications.

Kogut, B., & Zander, U. (1992). Knowledge of the firm, combinative capabilities, and the replication of technology. *Organization Science, 3*, 383-397.

Kolb, D. (1979). On management of the learning process. In D.L. Rubin & F. McIntyre (Eds.), *Organizational psychology: A book of readings.* Laxington: Prentice Hall.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation.* Cambridge: Cambridge University Press.

Levine, L (2001). Integrating knowledge and process in a LO. *Information Systems Management, 18*(1), 21-33.

March, J.G., & Simon, H. (1958). *Organizations.* New York: Wiley.

Mathiassen, L., & Pourkomeylian, P. (2003). Managing knowledge in a software organization. *Journal of Knowledge Management, 7*(2), 63-80.

McAdam, R., & McCreedy, S. (1999). A critical review of knowledge management models. *The LO, 6*(3), 91-100.

Miner, A., & Mezias, S. (1996). Ugly-duckling no more. Pasts and futures of organizational learning research. *Organization Science, 7*(1), 88-99.

Mirvis, P.H. (1996). Historical foundations of organizational learning. *Journal of Organization Change Management, 9*(1), 13-31.

Mumford, A. (1995). The LO in review. *Industrial and Commercial Training, 27*(1), 9-16.

Nicolini, D., & Mezner, R. (1995). The social construction of organizational learning: Concepts and practical issues in the field. *Human Relations, 48*(7), 727-746.

Nidumolu, S.R., Subramani, M., & Aldrich, A. (2001). Situated learning and the situated knowledge web: Exploring the ground beneath knowledge management. *Journal of Management Information Systems, 18*(1), 115-150.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science, 5*(1), 14-37.

Nonaka, I., & Konno, N. (1998). The concept of 'Ba': Building a foundation for knowledge creation. *California Management Review, 40*(3), 40-55.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge creating company.* New York: Oxford University Press.

Orlikowski, W.J. (2002). Knowing in practice: Enacting a collective capability in distributed organizing. *Organization Science, 13*(3), 249-273.

Paulk, M.C. (1998a). Using the capability maturity model for software to drive change. In T.J. Larsen & E. McGuire (Eds.), *Information systems innovation and diffusion: Issues and directions* (pp. 196-219). Hershey: Idea Group Publishing.

Paulk, M. (1998b). *Practices of high maturity organizations.* Retrieved August 16, 2005, from www.sei.cmu.edu/pub/cmm/high-maturity/survey98.pdf

Paulk, M.C.W., Charles V., Garcia, S.M. Garcia, C., Bush, M.B., & Marilyn, W. (1993a). *Key practices of the capability maturity model, Version 1.1.* Retrieved January 30, 2005, from www.sei.cmu.edu/publications/documents/93.reports/93.tr.025.html

Paulk, M., Curtis, B., Chrissis, M., & Weber, C. (1993b). *Capability maturity model for software, Version 1.1.* Retrieved January 30, 2005, from www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html

Ramanujan, S., & Kesh, S. (2004). Comparison of knowledge management and CMM/CMMI implementation. *Journal of American Academy of business, 4*(1/2), 271-277.

Ravichandran, T., & Rai, A. (2000). Quality management in systems development: An organizational system perspective. *MIS Quarterly, 24*(3), 381-415.

Richter, I. (1998). Individual and organizational learning at the executive level: Towards a research agenda. *Management Learning, 29*(3), 299-316.

Robey, D., Welke, R., & Turk, D. (2001). Traditional, iterative, and component-based development: A social analysis of software development paradigms. *Information Technology and Management, 2*(1), 53-70.

Sawyer, S., & Guinan, P.J. (1998). Software development: Processes and performance. *IBM Systems Journal, 37*(4), 552-568.

Schank, R.C., & Abelson, R.P. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge systems.* Hillsdale: Erlbaum.

Senge, P.M. (1990). *The fifth discipline: The art and practice of the LO.* New York: Doubleday.

Shrivastava, P. (1983). A typology of organizational learning systems. *Journal of Management Studies, 20*(2), 7-28.

Tsoukas, H. (1996). The firm as a distributed knowledge system: A constructionist approach. *Strategic Management Journal, 17*(Winter Special Issue), 11-25.

Ulrich, D., Von Glinow, M.A., & Jick, T. (1993). High-impact learning: Building and diffusing learning capability. *Organizational Dynamics, 22*(2), 52-66.

Weick, K.E., & Bougon, M.G. (1986). Organizations as cognitive maps: Charting ways to success and failure. In H. Sims Jr. & D. Gioia (Eds.), *The thinking organization.* San Francisco: Jossey-Bass.

Yeung, A.K., Ulrich, D.O., Nason, S.W., & Von Glinow, M.A. (1999). *Organizational learning capability.* New York: Oxford University Press.

Zack, M.H. (1999). Developing a knowledge strategy. *California Management Review, 41*(3), 125-145.

This work was previously published in An Overview of Knowledge Management, edited by B. Walters; Z. Tang, pp. 104-120, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 6.7
Agile Practices in Project Management

John Gómez
Ericsson Chile, Chile

Alejandro Núñez
Practia Consulting S.A, Chile

ABSTRACT

This chapter introduces agile project management as a way to improve the processes for software development in small organizations. The chapter contains a description of the main concepts and techniques used along with practical recommendations for their application in real situations. The chapter also analyzes the relationship between these practices and recognized process improvement models like the CMMI and the PMI PMBOK and presents case studies to illustrate implementation.

INTRODUCTION

Most of the reasons behind failure in software projects lie in the lack of sound project management practices. The results of many industry studies and surveys show that the absence of appropriate strategies for scope management, risk handling, or project planning are frequently found in challenged projects.[1] For that reason, process improvement (PI) initiatives start with the project management discipline. For example, the maturity level 2 of the Capability Maturity Model Integration (CMMI®[2]) model is focused on the development of basic project management capabilities (CMMI Product Team, 2006, p. 55). This means that although an organization should improve the project management and the engineering process, beginning with the first one may allow it to obtain better results.

Nevertheless, starting process improvement is an overwhelming endeavor no matter the size or nature of the organization. The improvement initiatives have to compete with "delivery projects" that always seem to be more important (or urgent) especially from the business user's point of view. The benefits of a PI initiative are usually difficult to perceive or measure in the short term. This causes the organization motivation to decline progressively and lead the initiative to failure. This situation is even worse for small or medium organizations where resource limitations are higher. Small and medium organizations must approach process improvement in a way that benefits are realized sooner.

Another aspect of PI projects that reinforces the situation previously described is that many times PI teams replace the absence of good practices with over-engineered processes where formalism and control exceed what is needed due the nature of the work on the project or organization. Managers, users, and practitioners start to perceive these new processes as obstacles and not as tools and refer to the new way of doing things as bureaucratic, rigid, or heavy-weight. Product quality may be improved (initially), but team productivity and motivation remain low which is going to impact product quality in the long run. Also, team focus deviates from reaching project objectives to blindly follow procedures. Small and medium organizations are also more affected by this situation since usually their environments (team size, product size, project duration, cost, etc.) are smaller, and over-engineered methods may have a greater impact on project delivery.

The application of agile practices for project management addresses these common problems and may allow an organization (especially small and medium ones) to manage effectively a process improvement initiative. The development of project management capabilities facilitates the establishment of the environment to control not only project delivery but also the improvement project itself. The agile approach (by definition lighter and goal-oriented) may reduce the effort (and cost) and contribute to realize benefits sooner, keeping high morale and motivation. Our intent is to describe briefly how agility is understood and applied within the project management context and how this may benefit a process improvement initiative.

There has been a lot of discussion between agile and traditional methods authors and supporters. We do not adhere to any of them, and our purpose is not to contribute to any side of that polemic. What we see is that the limits known as the usual home grounds for agile and traditional methods are blurring creating environments where no one of them is enough or complete. A mixed approach is needed to find the best solution. Also, as many other disciplines, agile project management (APM) may be enhanced by a proper use of tools. Choosing the right tool and deciding how to use it is not easy, so we created a special section to make some recommendations on that subject.

Agile Project Management

Origin of APM

APM has its roots in the agile methodologies for software development created specially during late 1990s and publicly formalized as a movement with the formation of the Agile Alliance and subsequent publication of the Agile Manifesto (see Figure 1. *Manifesto for Agile Software Development*) in 2001 (Beck et al., 2001). Every agile method adheres to the declaration of values and principles stated in the manifesto, but the approach to deliver the solution varies from one to another; however, since agile methodologies are designed to handle a product development project, project management practices are present in most cases. We took our main references from the work of Jim Highsmith who recently published a book on agile project management, the SCRUM method created by Ken Schwaber

Figure 1. Manifesto for Agile Software Development

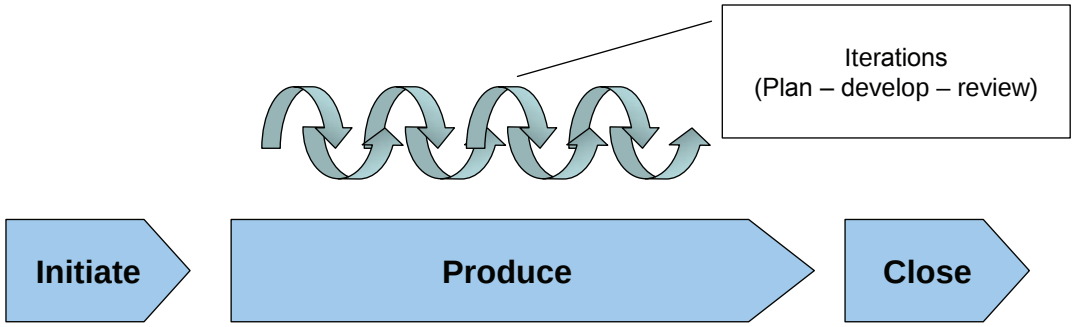
Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

- Through this work we have come to value:
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 2. Agile project life cycle



and Jeff Sutherland, and the Crystal Clear method developed by Alistair Cockburn.

Since its official birth in 2001, agility has grown beyond software development, and one of these areas is project management. This means not only that practices that were initially used for software development projects are currently applied for nonsoftware projects but also that there is an ongoing effort within the agile community to improve and extend the current state of the practice. The formation of the Agile Project Leadership Network (APLN[3]) and the publications of books on agile project management like the one from Highsmith (2004) mentioned before showed that work.

Agility Defined for Project Management

For the purpose of the chapter agility in project management is defined by:

- Keeping the team focused on reaching project goals oriented by customer objectives
- Use of a risk-oriented approach to accomplish project goals based on frequent delivery of functional solutions
- Creating and maintaining a healthy environment for team interaction and development, enhancing self-management, transparency, and visibility
- Use of balanced project management processes to allow the appropriate mix of flexibility and control, enhancing team productivity

The APLN also published a declaration of values and principles of agile and adaptive approaches to project management (see Figure 3. *Declaration of Interdependence*). Note how for APLN the appropriate term is "leadership" and not management. We talk about this topic later when discussing the role of the project manager in an agile environment.

Life Cycle of an "Agile-Managed" Project

There are three major phases of an agile managed project:

- Initiate phase, where product vision is defined, initial scope is established, a high level plan is elaborated, and the team and environment are set up.
- Produce phase, which is conformed by a successive set of iterative and incremental deliveries of functional product that ends in a product release.
- Close phase, which ends the project and gathers lessons learned.

Figure 3. Declaration of Interdependence

Declaration of Interdependence

Agile and adaptive approaches for linking people, projects and value

We are a community of project leaders that are highly successful at delivering results. To achieve these results:

- We **increase return on investment** by making continuous flow of value our focus.
- We **deliver reliable results** by engaging customers in frequent interactions and shared ownership.
- We **expect uncertainty** and manage for it through iterations, anticipation, and adaptation.
- We **unleash creativity and innovation** by recognizing that individuals are the ultimate source of value, and creating an environment where they can make a difference.
- We **boost performance** through group accountability for results and shared responsibility for team effectiveness.
- We **improve effectiveness and reliability** through situationally specific strategies, processes and practices.

The produce milestone may be divided into more than one release and each release in more than one milestone to establish different levels of control. The most important concept on this stage is the iterative-incremental approach that aims to reduce risk and uncertainty by delivering fully functional portions of the product to the customer. Every iteration has an internal cycle of plan (reviewing and updating scope and high-level plans and elaborating a detailed plan for the iteration), develop (building functional increments of the product), and review (team self-assessment on project performance and customer reviews of the product). In the next section, we will be seeing practices that fit into this life cycle.

Agile Project Management Practices

Product and Project Envisioning

Establishing a shared vision is critical in an agile environment. This is based on the belief that people work better when they are motivated for the long-term effects of the job. This is done by making the team realize the benefits that the project results will bring to the customers and the performing organization. A product or project vision statement is declared using workshops as team-building tools. Defining the product vision must include:

- Description of customer needs, motivations, or problematic situations
- Expected benefits or value added to customer business (problem solving, reducing costs, increasing revenue, or enhancing competitive advantage)
- Expected benefits for the organization performing the project

Highsmith (2004) suggests the use of a technique called the product vision box, where the team and customer representatives participate in a workshop to design the box that will contain the product. The team designs the front and back of the box and defines a set of statements that should be on the box to help sell the product. This may end with an "elevator test statement" containing the product vision (Chapter VI). For Crystal, this is part of a workshop is called "The Exploratory 360º." The product/project vision is defined in project initiation but is reviewed and communicated along the project to assure that objectives are still on target and that the team is oriented by those goals. The project vision normally ends with some charter of the project that contains the product vision, business objectives, project organization, and environment definitions.

Environment Setup

Enhancing team interaction is one of the goals of an agile managed project. This practice refers to establishing the environment when interaction happens. Environment definition covers essentially:

- **Interaction rules:** Agile methodologies are oriented by principles and values. These two, in combination with the product/vision, have a function within the agile environment: they allow team self-management. In agile environments many decisions on how to accomplish the work are left to the team who uses this rules to keep the work oriented to goals.
- **Processes and tools:** Agile teams should select the processes and tools that allow them to accomplish the objectives. A methodology workshop may be conducted to decide on this topic that results in a set of roles, practices, work products, and tools that the team will use on the project.
- **Physical environment:** Co-location is frequently suggested in many forms within agile environments to facilitate communication and interaction. This can be seen explicitly as a practice in extreme programming (XP) (Beck, 2000, chap. 13) and slightly modified with the osmotic communication property of Crystal Clear (Cockburn, 2005, p. 24). Face-to-face conversation is the most effective way of communication and the physical environment must facilitate that. Also the environment is designed to promote transparency, so it is common to use dashboards with task or feature cards where critical information like issues or progress are published and made visible to the whole team. A very good description of this practice is the information radiators strategy in Crystal Clear (Cockburn, 2005, p. 54). In projects with virtual teams, other collaborative tools are available and frequently used. The use of these kinds of tools will be commented on later.

Scope Management Strategy

Scope creep is one of the most common causes related to project failure, so strategies to handle this are a must. The degree of uncertainty is key in defining an adequate strategy for scope management. Uncertainty leads to change, and agile teams recognize that change is unavoidable and even beneficial. In that sense, change requests are allowed anytime in pursuit of maximizing the value that a customer may get for the product. Anyway, implementing changes as they are requested leads exactly to scope creep, so there must be some sort of change control. This is how it works:

- Scope is established using backlogs of scope elements. A scope element is a requirement (functional and nonfunctional), feature, use case, user story, defect, or change request.
- Changes to scope elements are received anytime but added at the end of the backlog.
- At the beginning of each iteration, the backlog is prioritized to decide which elements are to be developed during the iteration. Change requests are in the backlog so the customer has the chance to include them for the next iteration if they are considering more important that previous elements.

Engagement rules are established at the beginning of the project to manage customer expectation, and roles are defined to accomplish the scope management tasks. Normally, a customer representative owns the backlog and is in charge of prioritizing. This customer representative act also as an official provider of scope definitions. This way, uncontrolled changes from unofficial channels are prevented.

With this approach, the customer does not have to wait too long to include a change if he/

she really wants it. When the time of prioritizing comes, many change requests are dismissed because they add little to no value compared to other features previously defined. Iteration duration is critical to the success of this strategy. The iteration duration establishes the "lag" or "waiting time" for the customer to introduce changes to the project. A high change rate suggests a shorter iteration time. Anyway, sometimes changes are critical and must be included right away. When this situation happens, there is no sense to continue the iteration: the iteration is stopped and a new iteration is planned.

The backlog is usually maintained with as little detail as possible to facilitate updates. Scope elements definition may be detailed some more with feature cards or use cases/user stories descriptions. Detail information is usually added only for elements to be developed in the next iteration. Scope definition is also complemented with release or milestone plans. At the beginning of the project, the initial scope is established and release or milestone plans are elaborated to organize the project within a schedule. A release or milestone plan is usually a high level plan that describes which scope elements will be developed for each release or milestone. Release and milestone plans are based on the backlog and are reviewed and updated each iteration. This subject will be covered next.

Layered Planning

The focus on the planning process is to keep it the most cost-effective as possible. According to the scope management strategy, there are scope elements that are really committed (the ones that will be developed the current iteration) and others that are not. Detailed planning for items not committed or with low priority may be a waste. However, there should be a mechanism to elaborate a schedule for the whole project. A common misunderstanding of agile projects is that since changes are always allowed, the project is not able to commit for a fixed date. This is not true. A finish date is required to organize the work. This seems to be a paradox but has a simple solution. At the beginning of the project, an initial scope is established, and with that initial scope, a project plan is elaborated. This plan is kept at high level and its purpose is to establish a base schedule with some levels (or layers) of control. A large project with more than one delivery (release) to production environment may be organized by releases (every six months or more). A release is an organized set of scope elements associated with a date in the schedule. This primary level of control may be decomposed into a second one (milestones, from one to three months) that are also defined in terms of scope elements and dates. The work to accomplish a milestone is then divided in iterations (from one week to one month). The iteration is the minimum unit of work and is where the scope management strategy is fully applied. The iteration work is fixed, and the team is able to focus on that work. Then, predictability for the iteration work is higher, and detailed planning is cost-effective. The plans for the primary (releases) or secondary (milestones) layers are reviewed and updated each iteration.

Iteration planning is usually done in workshops where the customer and the whole team interact. This workshop is present in most of the agile methods (Beck, 2000; Cockburn, 2005; Scwhaber, 2004) and usually works like this:

- During the first part of the workshop, the customer representative explains the scope elements to be developed during this iteration.
- The team raises technological or environmental issues that may affect priorities which are adjusted.
- Based on previous performance results, the team estimates the effort to develop the scope elements. Restriction analysis (resource availability, risk identification, etc.) are incorporated to the estimations to allow

contingencies. Based on estimation, the scope for the iteration is agreed and committed. The whole team participates in the estimations, produced taking into account the complexity and size of the requirement, using a relative scale and assigning "points." This way, the simplest requirement takes one point and so on. The concept of relative scale refers to a predefined set of the amount of points that a requirement may get: 1, 2, 3, 5, 8, 13, 20, 40, 80 (Cohn, 2004, chap. 8). This kind of scale facilitates discussion among team members considering that for more complex or larger requirements uncertainty is higher and may reveal the need for further information or more decomposition.

- The team takes scope elements and breaks them down into tasks; tasks are assigned and effort is estimated. Task estimation is verified against scope element estimation to verify consistency.
- Resource load is reviewed and leveled.
- Team decides on task dependencies and sequencing and defines priorities. Tasks for the first day of work are agreed on and communicated.

The planning workshop is usually time boxed (Schwaber & Beedle, 2002).[4] The tasks are listed in another backlog. To facilitate control, tasks are always mapped to scope elements. This allows the team to keep the focus on the commitments for the iteration.

Frequent Inspection

The purpose of frequent inspection is to detect any obstacles before they can cause a major impact on project goals. The frequency of the inspection is left to be decided by the team, but most of the agile methods detect and remove obstacles on a daily basis. There are two more objectives of frequent inspection: set up a space for team interaction and communication and provide visibility on work progress to the team and project stakeholders. There are two main strategies related to frequent inspection: daily team meetings and daily task re-estimation and burn-down charts.

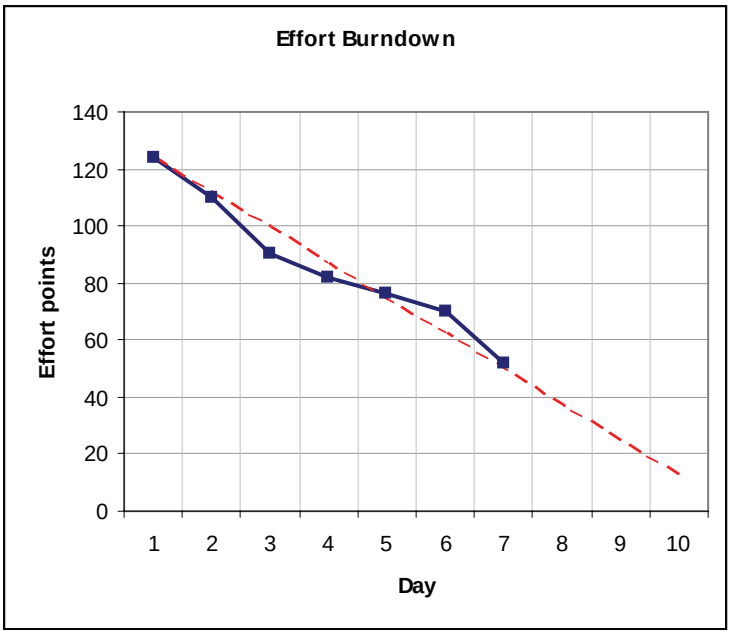
In the daily team meeting, the whole team gathers together for a maximum of 15 minutes with the main purpose of detecting any obstacles that may prevent the team from accomplishing the iteration goal. Each member of the team participates in the meeting, answering three questions:

- What tasks was I working on since the previous daily meeting?
- What tasks am I going to be working on until the next daily meeting?
- What obstacles or problems are or may be on the way?

No other subjects are allowed at the meeting, and problems are scheduled for further discussion later, commonly right after the meeting. Many soft aspects are related to this meeting: the team usually forms a circle (of peers) and is standing up. When a team member speaks, everyone else is listening. No interruptions are allowed, but questions may be asked to clarify some subject. The team leader conducts the meeting but only to keep the focus and register issues. Issues are registered for further action. It is expected that a defined response action must be in place for every issue before the next meeting.

The second strategy is to re-estimate the remaining effort of every not-completed task daily. Based on the daily information of total remaining effort, a "burn-down" is calculated. This information is graphically represented in "burn-down" charts as seen in Figure 4. Burn-down chart. In the figure, the blue/continuous line is plotted between the points of total remaining effort for each day. The trend of this line represents the "velocity" of the team. This actual velocity may be compared against an ideal or planned velocity (the red-dashed line) to obtain daily information

Figure 4. Burn-down chart



about progress and make performance forecasts. The graphics may be interpreted as follows:

- If at any given point, the actual velocity line is over the planned velocity line, the team is advancing slower than expected; probably, there are issues impeding the team progress or maybe the team underestimates the work to be done, so adjustments may be needed.
- If the actual velocity line is below the planned velocity line, the team is advancing faster. Assumptions may have changed, or the team overestimates effort.

With the value of actual velocity, the team is able to forecast its performance and make adjustments to assure that it will meet the iteration objectives. Burn-down information is public, and graphics are printed and published in a dashboard or any other place that everyone can see.

Product Review

Product review is present in agile environments with different techniques. For software projects, the use of automated unit testing linked to a disciplined procedure is frequent. Therefore, customer involvement in reviews is required and techniques like acceptance testing, technical reviews, or customer focus groups are common (Highsmith, 2004, chap. 8). These customer reviews occur normally at the end of every iteration. In SCRUM, there is a practice named sprint (iteration) review, the main focus of which is to review the product developed during the sprint; the sprint review is a meeting time boxed to three or four hours where the team presents the product to the product owner and other stakeholders. Defects and change requests are added to the product backlog to be prioritized at the next sprint (Schwaber, 2002, p. 54). The purpose of this practice is to show the actual progress on the product development and allow the customer to make changes and recom-

mendations if the product does not match stated requirements and expectations.

Team and Environment Tuning

Not only is the product reviewed to verify conformance to requirements and expectations; the performance of the team is also evaluated. At defined moments or at the end of every iteration, the team meets and reflects on its performance trying to find essentially what things worked well (and should be maintained), what did not (and should be adjusted or removed), and what improvements may be made to the team interaction or environment to enhance performance. In SCRUM, this is called the sprint retrospective (Schwaber, 2002, p. 54). Crystal Clear named it the reflection workshop (Cockburn, 2005, p. 65).

Time Boxes

The use of time boxes has a wide range on agile environments and that is why we decided to include a special part on this technique. A time box is a fixed duration time frame assigned to an event in the project. The most important time box is for iterations: once the iteration starts, the finish date is not changed. Time boxes are also applied for the daily meetings and the planning and reflection workshops.

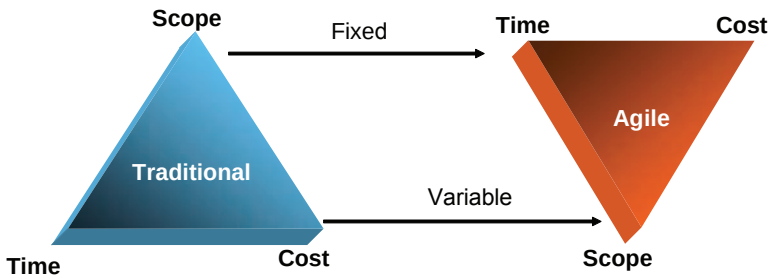
Role of the Agile Project Manager

First of all, it is appropriate to state that the word "manager" is rarely used in agile environments to refer to the person that conducts the project planning and execution. Team self-management is a fundamental concept of agile projects, and having that in mind, there is no place for a traditional "project manager." Instead, the agile project manager's main responsibilities are:

- Facilitating team interaction, removing obstacles from the path of the team to allow goal accomplishment and enhance performance
- Coaching and serving as change agents, promoting the values and principles, and encouraging discipline on the defined processes and rules
- Leading the team by keeping the team focus on customer objectives and the defined product vision and project goals
- Coordinating the work, helping in decision making and problem solving

One of the most interesting aspects of this role is the difference between self-management and self-direction. For authors like Highsmith (2004), agile teams are self-managed since they organize their own work and decide how to accomplish the goals. However, the team is not self-directed: there is a leader that keeps the work on track of the vision and objectives (Chapter III).

Figure 5. Agile vs. Traditional PM paradigms



Relating APM and Traditional PM Models

Agile vs. Traditional Paradigms for Managing a Project

Figure 5[5] shows the way the triple-restriction of projects is managed in agile and traditional environments. From the agile point of view, traditional project management aims to maintain the scope fixed by strict control of changes. When a scope is committed, the project tries to deliver it allowing some variations of time and cost. If the scope changes significantly, then the time and cost are adjusted. In an agile project, the time restriction is fixed (using time boxes), and the scope is allowed to change in benefit of the customer objectives, which is facilitated by the prioritizing practice in the scope management strategy.

Agile Project Management and the CMMI[6]

The CMMI model is one of the most important references in process improvement, especially for the software development industry. The CMMI establishes a set of recognized best practices for product development at the project and organizational contexts. The model is organized in 25 process areas, six of them dedicated to project management. Therefore, there is a cross-sectional focus on project management for the maturity level 2 and capability levels 1 and 2. That is why we decided to comment on the relationships be-

Exhibit 1.

Process Area	Specific Goal	Compatibility of Agile Practices
Requirements Management	SG1 Manage requirements	In an agile environment, there is a defined scope management strategy. There is a documented backlog of scope elements. Understanding is assured by the definition of official providers and the customer involvement in the development. Commitments are made and established in the iteration plan. Normally, there is traceability between scope elements and tasks. Changes are managed.
Project Planning	SG1 Establish estimates	Scope is established trough the development of the scope elements backlog and the release/milestone plan which also contains the defined lifecycle. Rationale for estimations is present and usually scope elements receive an assignation of attributes of size and complexity. Effort estimation is realized at ROM for the project and detailed for each iteration.
	SG2 Develop a project plan	Explicit planning of some parameters is missing (i.e., data management) and may be analyzed for each project. A schedule, budget, and resource plan exists. There are artifacts that represent the planning process.
	SG3 Obtain commitment to the plan	Planning is usually made at workshops when the customer participates. Resource leveling occurs by iteration.
Project Monitoring and Control	SG1 Monitor project against plan	During the daily meeting, issues that may cause deviations are detected. During iteration planning, the parameters for the whole project are reviewed and adjusted. Progress is analyzed daily using burn-down charts.
	SG2 Manage corrective action to closure	Issues detected are analyzed and a corrective action should be initiated as soon as possible.

continued on following page

Exhibit 1. continued

Process Area	Specific Goal	Compatibility of Agile Practices
Integrated Project Management	SG1 Uses the project defined process	Normally, at the beginning of the project, a process is defined to manage the project. This process is reviewed at the end of every iteration and adjusted. Estimations are adjusted using velocity data of previous iterations or projects. Lessons learned are gathered and shared among teams.
	SG2 Coordinate and collaborate with relevant stakeholders	Stakeholders are identified and its participation defined from the beginning of the project.
Risk Management	SG1 Prepare for risk management	The priorities and the choosing of life cycle are driven by project risks.
	SG2 Identify and analyze risks	Risks are identified and monitored as potential blocks in the daily meetings.
	SG3 Mitigate risks	Potential blocks are detected and preventive actions initiated using the same rationale as issues.

tween the CMMI and APM. We are not trying to prove that an organization may achieve a CMMI maturity or capability level using agile practices. Our purpose is to show how the agile practices we commented are related to some key elements of the project management discipline on the CMMI model. The table shows the goals process areas of the project management category until maturity level 3. Supplier management was out of the scope of this work, so the supplier agreement management process area was not included; instead the requirements management process area was included due its relevance to the project management discipline; although for the CMMI, it belongs to the engineering category.

Note how there are many agile elements that give some level of support to the specific goals of this process areas. Considering the nature of the SCAMPI method is difficult to affirm that an organization may achieve a capability or maturity level using only agile practices. In most cases, a refinement or addition is needed. Further, in this chapter, we comment on a case were a CMMI level 3 organization used some agile practices in a project without sacrificing the CMMI implementation.

Agile Project Management and the Project Management Body of Knowledge (PMBOK[7])

The PMBOK (Project Management Institute, 2004) is recognized as one of the top standards in the project management profession. As many traditional methods are usually misunderstood as heavyweight processes that are unsuitable for small environments, the PMBOK contains a series of best practices that are recognized as successful for a wide range of projects, and it is not expected that every one of them should be applied for every project. One statement says that quality should be planned into the project from the beginning and defining adequate processes is part of that definition. The adequate processes should address the objectives and nature of the project and this does not mean that every practice in the PMBOK is adequate for every case.

Exhibit 2.

Knowledge Area	Compatibility of Agile Practices
Scope management	Scope is defined using the scope elements backlog and the release plan, which when combined are equivalent to the WBS. The scope management strategy is defined from the beginning of the project and most of its rules (prioritizing as a key one) are performed every iteration.
Time management	Activity definition and dependency analysis for sequencing occur every iteration. There is not usually a bar (Gantt) chart or network diagrams to show the schedule. Also, techniques like resource leveling, theory of constraints, and rolling wave planning are frequently applied in agile environments. Schedule control also uses time boxes and is supported by burn-down charts.
Cost management	ROM (rough order of magnitude) estimations occur at initiation. Detailed task effort estimations occur every iteration. Effort control is based on daily re-estimation and burn-down charts.
Risk management	The whole life cycle of the project and the priorities are driven by project risk. Quantitative risk analysis is not commonly used. Mitigation occurs at the project level with time boxing and prioritization and at the iteration level with definition of corrective and preventive actions to avoid or remove obstacles
Quality management	Adequate processes are selected at the beginning of the project. Normally, there are quality control techniques in place like acceptance testing, product reviews, or customer focus groups.
Human resource management	The team is the most important factor in agile environments. Ground rules are explicitly established at the beginning of the project and maintained along the way. Team development is enhanced with reflection on causes of good or bad performance. The daily meeting allows for team management and interaction.
Communications management	The PMBOK states that the best way of communication is the face-to-face conversation which is also an agile principle. Communication is key on agile environments and frequently promoted through the workshops, daily meetings, and even the physical environment (i.e., with information radiators). Stakeholder identification and analysis is also conducted.
Integration	Chartering occurs at the beginning of the project by establishing a vision of the product and the project. The project is managed with integrated perspective of the scope, release plan, and iteration plans, which are maintained consistently through the project.

We can start analyzing compatibility with the groups of processes of project management in the PMBOK (initiating, planning, executing, monitoring and controlling, and closing) and the agile project life cycle. The process groups do not happen in a cascade in the project; instead they occur many times in different project phases. In an agile project, initiating and closing are related to the initiate and close phases, and the produce phase are continuous cycles of planning, executing and monitoring, and controlling the iterations. For that reason, most of the planning processes occur every iteration. The following table shows some compatibility of agile practices within the context of PMBOK areas:

Lessons Learned (Tales from the Trench)

Managing a Project Without a Gantt Chart

Power Games Co. (PGC) is a small game development studio[8] with just three years on the market. PGC works for larger game publishers developing small products oriented to the casual gamer segment. A typical project at PGC takes about six months with a team of five to six people integrated by designers and engineers. In the game industry, the product are built incrementally. This is explained since most of the features need to be

tested to prove not only that the software works as specified but also that the gaming experience is fun. PGC delivers a product increment each month or so. The culture is informal, and people are very talented.

PGC has a product manager who is in charge of all projects, and from previous work experiences, she firmly believes that the best way to eliminate risk or uncertainty is to plan up-front the activities with as much detail as possible. The team writes a project plan and produces a detailed schedule of the whole project tasks. Preparing this plan usually takes from two to three weeks. The team is not very comfortable with this way of doing things that, by the way, does not seem to be consistent with the culture and the iterative product development cycle; however, it seems to work for PGC: products are delivered on time and usually exceeding customer expectations. Publishers give PGC bigger contracts which implies developing more exciting games and making people feel happy. There are no complaints.

This success opens new opportunities for PGC, and one shows up quickly: a very important publisher contracts PGC for a new game. This project is different from any other PGC has executed in the past. It is a nine-month project that requires at least 15 people; most of the visual design of the game has to be outsourced and the game involves the use of technology that is unknown to people at PGC. Another challenge rises up: contract and franchise negotiations took almost two months, but the delivery dates remain fixed. A nine-month project is now a seven-month one. The product manager decides to start planning the project right away, as usual: the team takes about three weeks to build a detailed schedule with fine-grained tasks for the seven months of the project plotted in a Gantt chart. The first production iteration starts, and three weeks later (one before the first delivery), the project is completely challenged. Tasks seem to be advancing, but no real progress is visible. Team interaction is difficult: status meetings are held weekly but are not effective. Morale is very low. By this time, many tasks in the schedule do not make sense anymore, and the schedule has to be updated. This sole fact worries the team a lot because they know that it may take at least a week.

It is evident that the management approach is not working in this case, and a change is needed. The team decides to use an agile approach. First, the team is divided into three mini-teams of five people each. Each team has an assigned leader and is self-sufficient in terms of the required skills; this means that each team has at least a game designer, an illustrator, an animator, and a programming engineer. The product is also divided into two major components and assigned to the first two teams. The third team receives the responsibility of integrating the game. Regarding the planning process, considering the business and technology uncertainty, it is recognized that full up-front planning is not useful so a high-level milestone plan is prepared, and it is defined that detailed planning will be conducted iteratively. The iteration's duration is established in two weeks. Every iteration will start with a planning session when the scope is analyzed and tasks are identified and estimated. Tasks will be re-estimated daily, and progress will be reported with burn-down charts. Each team will hold daily inspection meetings. Team leaders will hold a daily meeting too. Agile principles and philosophy are communicated to the team, and ground rules are established. To support this transition, a coaching strategy is launched.

The results for this project improve significantly. After the first iteration pitfall, commitments with the customer are always fulfilled on time. Technology issues are common, but they are handled by the team and the product is delivered with high quality levels. There is also another big gain: people at the company have acquired competencies to lead a team and manage a project. The company feels more confident in taking bigger and more challenging projects. There is a paradox in this case that must be noticed. What

we can call a natural agile environment (small new product developed iteratively by a co-located team of highly talented people) was performing well using a traditional method. However, when they needed to scale-up, agile practices supported that better.

Mixing Approaches

Real Insurance Corp. is a large CMMI maturity level 3 company for which the project management methodology is based on the PMI framework. IT projects are managed using a strong matrix organization and the project management discipline is supported by the establishment of a project management office (PMO). There are senior project managers with more than eight years of experience. The company wants to develop two products for a business unit. The first one is a traditional product that is going to be re-engineered in a 12-month project that is partially outsourced. The second one is a new product which is going to be based in the framework developed for the first one. This second project will start three months before the first one ends and is planned to last about nine months (the first release in the sixth month). The first project starts and is challenged from the beginning: deliveries are always late; the primary supplier is changed twice and deliveries are normally rejected. The progress is very slow and schedules change frequently. The second project starts as planned by the ninth month of the first one with a six-person team. After two months, the first project is cancelled. A quality assurance audit is then conducted on the second project, but no significant methodology issues are detected. Knowing that this project was going to use components developed for the first one (that are not ready), six more people are added to the team. The team struggles trying to use the portions of components previously built, but the first partial delivery is late and rejected. This fact raises alarms to IT management which has been highly committed to the business unit since the first project failure. Now its three months to the first release, and the team realizes that it is less costly to stop trying to work around the half-developed components and start from scratch.

The decision was to include some agile practices to facilitate team interaction and improve morale. Scope management artifacts were maintained but change control procedures were slightly modified to allow fixed scope iterations. The scope of the first release was renegotiated by prioritizing items. The work was divided into four fixed-duration iterations: the first three finishing with a fully functional increment of the product and the fourth dedicated only to refinements and tuning. The project schedule was updated, leaving only milestones and high-level descriptions. The formal estimation method was maintained, but estimations were reviewed according to the defined iteration plan. Iteration planning workshops were conducted to create task backlogs. Daily meetings and burn-down charts were used. The main project management artifacts, like the project charter and project plan, were updated to reflect these changes. The project status reports required weekly by the PMO were maintained and also the risk management processes. Nevertheless, the main focus was not the process itself. A detailed communication plan was put in place and a coaching program was launched to facilitate change. The team was trained in agile principles and values, and workshops to establish ground rules were conducted. The team was co-located. The customer was involved in the methodology changes, and the acceptance testing plans were modified to fit the new approach.

The first iteration was the most difficult. The planning workshop lasted almost three days (not the day that was expected for the three-week iteration). Initially, the 12-person team was separated into two functional groups which was believed to be the most appropriate approach, but this was confusing for the people who decided to create groups by architecture layer. Initial daily meetings were also longer than expected. After

a week, the first integrations started with some suffering, but by the end of the second week, the team solved most of the problems allowing them to finish the iteration on time. Due the slow procedures to release a product to the acceptance testing environment, the result of the first iteration was presented in the development environment. Despite the customer's surprise, there was not too much to see but the team was able to accomplish a goal and deliver for the first time on the project a product that was functional. The team interaction improved a lot, but some political factors were present always and required a lot of attention along the project. The next iterations were less difficult but not exempt of challenges. The team was able to finish the development of all functionalities committed for the first release two weeks before the deadline, which was normally considered too short. However, since acceptance testing was conducted incrementally too, the first release went to production on time. Quality assurance reviews were conducted on the project, and no deviations from the CMMI process were noted. The management approach and practices were registered as lessons learned and made part of the organizational process assets.

Software Tools for APM

Software tools have been around us, helping to achieve goals since the beginning of the computer age. We can easily recall the different approaches to word processing of the late 1980s, the endless race of database engines of the mid-1990s, and the overpopulation of support software for whatsoever one may like in the present day. We may remember many enterprise resource plans (ERPs), customer relationship management (CRMs), operating systems (OSs), information managers, to-do lists, reminders, Web development, CAD tools, 3D modelers, and so on, but something amazing has revealed, in the continuous (and chaotic) race towards the future, that project management has received help from project management support packages. The offer is wide: they come in many flavors and some are more appealing to some clients (organizations) than to others. We will try to give a perspective of the present status of agile-focused project management tools or, to put it in the right way, tools that can be used to assist agile project management, and possibilities of their use in agile organizations, giving hints of alternate uses and warnings when risks are identified. In our search, we have recognized four flavors in which tools fit:

- Plan-based/traditional
- Wiki
- Web (2.0)
- Methodology-specific

As tools evolve and are fed with industry trends, new ideas, or customer feedback, they start to share some inspiration, leading ultimately to the fact that no tool can be allocated exclusively to one of these flavors, but the most influential one will represent its type. We will approach these tools with APM in mind, trying to relate their specific functions to components of APM.

Plan-based/Gantt Chart-Driven Software Tools

This is maybe the most overcrowded segment of consideration. There are many tools available, ranging from feature rich and quite expensive packages to free Gantt drawers. All these tools seem to have a strong bond to the traditional paradigm of project management, but some allow a degree of flexibility to adapt to the principles of APM.

Primavera Project Planner (Primavera) and MS Project (Microsoft) are the top of mind tools for many people, both are being used for a long time, and are the "de facto" standard for certain industries. Both tools are usually perceived as Gantt drawing tools, but they are feature rich and very helpful giving assistance to the project manager.

For product and project envisioning, Primavera Project Planner offers embedded forums on which the envisioning can take place as text post. These posts are published inside the application and are visible for all project members. Another possibility is to put the envisioning documents shared and access them through the embedded file manager. MS Project allows shared documents using project server's collaboration abilities.

For the layered planning, both tools use the traditional planning, so it is possible to specify milestones for the releases and some referential tasks to give a view of the project as a whole. Due to the inherent volatility of the tasks to be performed on an iteration, its inclusion on these tools would result in task/resource administration overhead. A parallel tool to maintain the feature list, task list, and estimations would be recommended. It must be said that a XP (template/extension) is available for MS Project on the Microsoft Web site (http://www.microsoft.com). Both products have an issue manager (in the case of Project, using Project server's collaboration capabilities) that allows tracking issues that might rise during product development.

Some other tools that have the same focus but with different implementations are eGroupware and TUTOS, both Web based, both open source. Each of these collaborative tools have a project module in which tasks and milestones are added with explicit task precedence and assignment. It is able to form a Gantt chart from the data set. Again, changing the content to be developed on the iterations can be a painful experience, especially when page loading time matters.

Wiki

Wikis have radically transformed the collaborative documentation works as we know them; its simplicity of concept has allowed many different implementations on many languages and platforms. Some people are using wikis as a project documentation repository; others are integrating tools around wikis to enhance planning and collaboration capabilities. Inside this group, the numbers point to open source as the most common licensing method. Among the best qualified applications is Trac (Edgewall Software), a software development project management tool with configuration management capabilities. Trac consists on a wiki integrated to an issue tracker and a subversion repository viewer for configuration management.

During the life cycle of the project, the wiki functions will allow the product and project envisioning development and organization in a collaborative way. Using the tool configurations, it is possible to define expected releases for the product. For each planned release, the project members have the chance to link any issue (or requirement) to any release, in such a way that the project features are organized based on the release in which they are expected. The wiki pages can act as concentrators, allowing linking from documentation to issues, change sets, files, including past versions of the project files (stored in a subversion server). The issue tracker provides configurable reports to check the project status according to the defined tasks.

A very lightweight alternative is TiddlyWiki and its branches, particularly d3, an implementation of GTD (Getting Things Done) by David Allen. D3 is a stand-alone wiki with the advantage of having certain tiddlers (wiki pages) with special behaviors, like projects or tasks that can be tracked as pending or checked as completed. Each tiddler has tags that will give us another way to search for them. Although this is not an APM tool, the flexibility that modern wikis give can empower a tool to handle more complex tasks. In this case, for the early stages of the project, the wiki inherent features can be very useful for the envisioning and project setup. Each task resides in a context; for our use, each context will be a release. In that way, a release will have many tasks that fulfill it. Using the task review, it is possible to see a list of all tasks pending, associated to its

particular release. In this case, we can use tags to assign the tasks to a particular team member. Any issue can be inserted into an issues context for its tracking.

Web

In the last year, the Internet community has seen the blooming of the Web 2.0 trend, the principles of which try to serve as a group of guidelines for Web applications towards a better user experience, where collaboration and social interactions are very strong concepts and tend to take applications out of the desk and onto the Net. Web 2.0 has brought some fresh air into project management tools, and some companies are actively working towards a better and more agile project management tool, and the results so far look promising.

Maybe the best example for this category comes with BaseCamp (37 signals), a Web application (service) for project management with an appealing user interface and simple but highly effective design. A BaseCamp project is a group of task lists, milestones, messages, and files. Its use for an APM project would be as follows: For envisioning, messages or files can contain the envisioning results that, in later phases, will become a major input for the planning; the release plan would consist of milestones marking approximate dates for each release; features are created in the form of task lists containing the actions to be performed in order to achieve the feature. Each feature can be assigned to a particular release, so it is possible to have a wide vision of the tasks needed to complete each release. This service was thought to handle many projects simultaneously and has access control features to give specific permissions to administrators, project members, and clients.

An open source clone is available under the name ProjectPier; it presents the same functionality of BaseCamp, with a less polished user interface. This tool can be installed into an organization's intranet (beating some of the fears that rise when one thinks of maintaining projects on the Web) and allows tagging for any element that is created. Although it is in active development, its available version has enough to carry a full project.

Stepping away from Web 2.0 and back to the traditional Web tools, another important referent is TargetProcess which can be used as a service or downloaded and installed in the organization's intranet. Its design denotes a profound knowledge of agile practices for project management and tries to cover the software life cycle in full. It has a user stories management module with mass import capabilities (from comma separated values files [CSV]), layered planning for releases and iterations, effort planning helpers, and team velocity estimation. It also includes per release burn-down charts based on the planning and daily status report. Additionally, it integrates its own bug tracking and test definition tools.

Methodology-Specific

Many of the APM tools available have a strong focus on a particular methodology. Among the more popular are Scrum and eXtreme Programming. In some cases, as we have seen in the past categories, we will notice mixtures of them. In this category, we discuss some tools with a scope limited to the methodology that may help to institutionalize their practices.

Wikiscrum is an open source wiki, adapted to support the Scrum project management methodology. Inside of a wiki page, the planned sprints are listed along the product backlog. Each sprint is clickable, leading to the task list for that sprint, having the opportunity to attach new tasks to it or to the product backlog, assigning the effort estimations for each task and the specific project member involved. One noticeable aspect of this tool is the charting capabilities included to show the burn-down chart for each sprint, based on the daily estimation variations for that sprint's

tasks. These are very basic but enough to get a team on track.

IceScrum is a basic tool for Scrum that lacks advanced functionality like planning support or resource estimating but achieves good works on giving order to releases, sprints, features, and tasks. It has security management that allows the daily review of the project, based on the team information about tasks. It is currently on active development.

On the road of eXtreme Programming, Xplanner is an open source tool made to simplify the project management using eXtreme Programming. The early stages of the project are supported in the way of a user stories management module; these user stories can be assigned to each planned iteration and, from these iterations, can be linked the tasks to perform. Among other aspects, the integration queue can be highlighted as a specific help for the project team, along with its time tracking module with special support for the tracking of pairs, like in pair programming.

CONCLUSION

The application of agile principles and value will continue its expansion to other disciplines. The next step from the project management point of view is the use of agile practices in multiproject environments and even more important in portfolio management. The portfolio management discipline by nature have many similarities with the agile context: a close relationship with business goals, prioritizing opportunities and enhancing resource productivity. Application of concepts like time boxes is starting at the project portfolio level in some companies, and we can expect many other applications in the next years.

We told before about the polemic raised between supporters of agile and traditional methods. For us, much of that discussion was based on misunderstandings. Many times, misusing a method is far more harmful than not having any. This is true for traditional and agile methodology implementations. A misunderstanding on the traditional planning paradigm may lead to the belief that fine-grained (one or two hour's tasks initiating 10 months from today) schedule elaboration is appropriate. A misunderstanding of the agile paradigm may lead to not having any plans at all.

Also it has been said that agile and traditional paradigms have home grounds where their principles and practices are appropriate, but outside of that zone, they are no longer valid. We agree that there is not any single method that may be applied to every context: there are no silver bullets. However, we do not believe that home grounds are mutually exclusive. In today's environments, the characteristics are crossing those limits. In those environments, neither agile nor traditional methods are able on their own to provide the best solution. The most effective solution is obtained by taking the best the two worlds can offer for that specific context. We truly believe that organizations may benefit from knowing how to apply best practices for their process improvement initiatives no matter the origin or philosophy behind them.

REFERENCES

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., & Fowler, M., et al. (2001). *Manifesto for agile software development.* Retrieved December 16, 2007, from http://www.agilemanifesto.org

CMMI Product Team. (2006). *CMMI for development* (version 1.2, CMMI-DEV v1.2, SEI Tech. Rep. No. 06.tr.008). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Cockburn, A. (2005). *Crystal clear: A human powered methodology for small teams.* Pearson Education Inc.

Cohn, M. (2004). *User stories applied for agile software development*. Addison-Wesley.

Highsmith, J. (2004). *Agile project management: Creating innovative products.* Addison-Wesley.

Project Management Institute. (2004). *A guide to the project management body of knowledge* (3rd ed.).

Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

ENDNOTES

1. The Chaos Report and Chaos Chronicles 1994, 1998, 2000, 2004, Standish Group (www.standishgroup.com)
2. CMMI is a Service Mark of the Software Engineering Institute at the Carnegie Mellon University
3. For more information on the APLN, please visit http://www.apln.org
4. Time boxing is commented on later. Scrum suggests a full day planning for a 30 day iteration.
5. This graphic, although common in agile sources, was adapted from the DSDM Consortium site (http://www.dsdm.org)
6. We are using the CMMI-DEV version 1.2 as reference (CMMI Product Team, 2006)
7. PMI and PMBOK are trademarks of the Project Management Institute in the United States and other countries
8. The names of the companies are fictional

This work was previously published in Encyclopedia of Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studiesg, edited by H. Oktaba; M. Piattini, pp. 193-211, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 6.8
Project Management in Enterprises:
IT Implementation Based on Fuzzy Models

Cezary Orłowski
Gdańsk University of Technology, Poland

Zdzisław Kowalczuk
Gdańsk University of Technology, Poland

ABSTRACT

The article discusses how the knowledge of management and artificial intelligence can be used for controlling the budgets and schedules of software projects. The first part of this paper gives an outline of the problems involved in software project management regarding the planning and control of processes and project teams. Next, an overview of changes in management is presented, followed by a description of a method for how these ideas can be used to solve software engineering problems. Consequently, an example is presented of a decision-support system, designed to aid project-team managers in planning and controlling budgets and schedules and helping the team members to adjust.

INTRODUCTION

In this paper, suitable methods for *Software Project Management* (*SPM*) are discussed. New possibilities of modelling SPM processes are indicated as well as an example concerning the issue of building a fuzzy model of a project team is presented. The scope of our research has been focused on specific project implementation performed by international project consortia. Such an ensemble can consist of several or more *project teams*, understood as those "distinguished from the structure of the organisation, commissioned for a defined period and consisting of specialists from various fields, whose knowledge and experience have a bearing on the problem" (Stoner, 1994, p. 143).

PROBLEMS OF MODELLING SPM

With the purpose of building on an SPM knowledge-based model, an appropriate *state of the art* has been established. The knowledge obtained in this way concerns the management techniques used by managers to support a project implementation and, based on process models, to allow for suitably assessing the states of both project processes and human teams (Figure 1). The consequences of these issues can be found in the budget, deadlines, and functionality of the project.

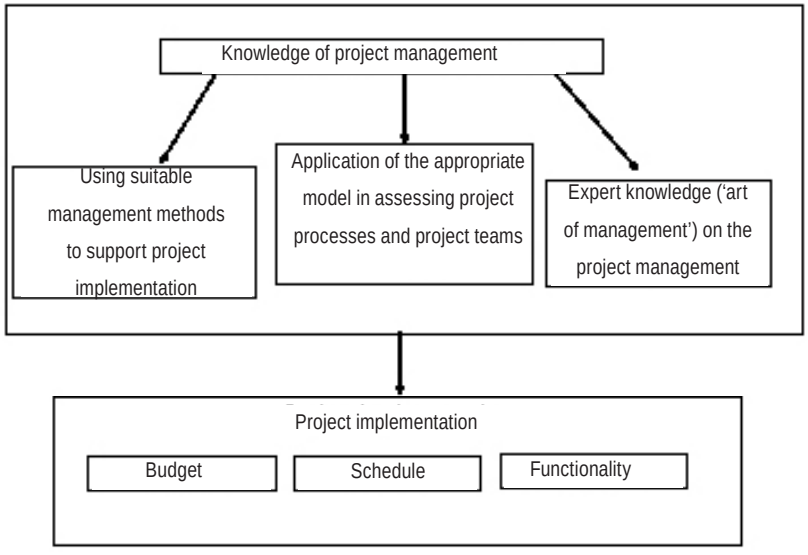
Expert Knowledge of Project Management

According to the SPM experts, scope, time, communication between team members, and risks of project changes are inter-related management knowledge factors. "Recycling" such knowledge in new projects is difficult. Those difficulties often result from an incomplete documentation of the project processes and from the lack of adequate knowledge, both on the factual project management mechanisms and way of describing SPM. They are also conditioned by the commercial nature of the projects (concerning the knowledge about implementations and their management). For given project experts (managers), the source of knowledge may be placed in the know-how of implementing previous projects. It is clear that such knowledge depends both on the specific characteristics of the performed projects and on the manager's ability to make suitable predictions, for example, to make use of this experience in implementing new projects.

Methods of Supporting Management Processes

Apart from expert knowledge, an important source of the education on project management can be found in the spectrum of the methods applied

Figure 1. Relationships between the knowledge of project management and project implementation



in project organisation. They constitute formal guides to proper behaviour in SPM. For example, *KADS* presented in the work of Hickman and Killin (1994) and *Pragmatic KADS* from the study of Kingston (1992) propose to divide projects into implementation phases and indicate solutions for these phases of the project. Adaptation of several known methods in the project management can be found in the works of Coleman and Somerland (1994) and Nerson (1992).

Individual methods created by certain large project teams can also be applied. As an example, the method of *Project Management Methodology* (*PMM*), discussed for instance by Łopaciński and Kalinowska-Iszkowska (1999), can be mentioned as being worked out by the IBM company. The main feature of PMM lies in planning the processes of the project and using a packet of *WS-DDM* (*Worldwide Solution Design and Delivery Methods*) supporting knowledge acquisition and project management.

There are also other methods which support project management in respect to costs, composition, and size of teams as well as labour intensity, as follows:

- *Estimation by analogy* — meaning an assessment of the project analysed on the basis of earlier implemented and documented projects;
- *Expert assessment* carried out by a group of independent experts;
- *Initial estimation* based on elementary work units (*Work Breakdown Structure* [*WBS*]);
- *Top down estimation* — the method of design within the limits of a given cost (*Design to Cost*) — introductory decomposition into simpler tasks (*Workpackages*) and definition of the necessary outlay of work, further decomposition to tasks, and exact processes;
- *Estimation based on parametric models* — the relationship between the productivity and the duration of the project as well as other factors directly bearing on them; and
- *Estimation in order to win* (*Price to Win*) — an assessment of the project is conducted in a way to outdo potential competitors.

For the purpose of managing time and material/human resources, the method of functional points analysis is applied. This method was worked out by Albrecht of the IBM company in 1979 and later perfected by the *IFPUG* (*International Function User Group*).

The method of functional points analysis, like *COCOMO* (Pfleeger, 2001), is characterised by the influence of subjective judgment, and not by objective indicators in the assessment of project productivity. This means the assessment of the projects productivity by the use of considerable experience and quite complicated algorithms.

Description of Project Teams

SPM demands a considerable involvement of human resources. As most beneficial solutions co-operation with an external organiser of the project and suitable definitions of the tasks of the team members are generally recommended. During team organisation, an attention should be given to the changes that may occur in the organisational structure of the consortium. New teams can be seen as complex systems in which information tools and management techniques are directly connected with each other (*Workflow Management, Groupware Process Reengineering, Computer Supported Co-operative Work*). It is also assumed that the project teams, brought together to implement a team task, can bear both formal and informal organisational structures. Such a project team is assembled by an appointed project manager. It is clear that the compiled team is usually interdisciplinary, which calls for using appropriate solutions to co-ordination problems.

Models for Assessing Team and Project Processes

Assessing teams and project processes needs appropriate models. A number of models, such as CMM (Capability Maturity Model) (Paulk, 1997), SPICE (Standard for Software Process Assessment) (Spice, 2000) and the norms of ISO 9000 have been elaborated. The models of process maturity (CMM) supply useful solutions making possible the handling of project processes. Their application supports the assessment of basic team management processes and defines their level of maturity. They also identify critical elements of the processes. The SPICE model constitutes a complement to other international standards of process assessment and other models for assessing the capacities and effectiveness of both teams and processes.

NEW POSSIBILITIES FOR CREATING THE SPM MODELS

There is an opinion in the subject literature (Pfleeger, 2001) that SPM is more an art than a systematic strategy of action. This judgment reflects the fact that, in the course of the project, the manager makes various decisions which can hardly be modelled and anticipated. They can, for instance, concern the running changes in the make-up of the team enforced by human and social factors (as best programmers may be "tempted" by a rival business). For these reasons, many managers consider the art and experience of SPM as being the principal source of knowledge about project management. Generally, the project success is connected with the manager's ability to appropriately forecast changes and react to the risks appearing in SPM. Many managers assert that it is not possible to successfully plan, organise and control a project without applying formal SPM procedures and methods.

Several methods of assessing projects in fuzzy categories (*Fuzzy Projects*) are discussed in the works of Słowiński and Hapke (1999), Węglarz (1999), and Hapke and Jaszkiewicz (1999). There are methods of scheduling projects with the use of *metaheuristic algorithms,* which are basically founded on genetic or evolutionary algorithms, simulated annealing, and taboo search. Such methods are widely used in solving problems of continuous, non-linear, multi-dimensional, multiple criteria, or combinatory characteristics. A multi-faceted version of this algorithm called *Pareto-Simulated Annealing* (*PSA*) makes it possible to search for representations of competitive (not dominated) solutions. It is also possible to use an *interactive search method* named *Light Beam Search* (Mesarovic & Takahara, 1989) to support the effort of the manager in choosing one of many solutions found by PSA.

A different group of system models are those concerning *management in uncertain conditions* as presented by Pawlak (1997) and Kacprzyk (1997). Their characteristic feature consists within incomplete or lack of information. Examples of this type can be observed within *relational, probability, game, and fuzzy models.*

Application of Fuzzy Set Theory

Fuzzy models of a linguistic type (*LM, Linguistic Models*) are based on a set of rules of type *IF-THEN* concerning undefined conditions and inference mechanism. They are profitably used in modelling social systems (Yager & Filew, 1995). Paradigms containing logical rules with fuzzy conditions and functional conclusions are marked as *Takagi-Sugeno-Kanga* (*TSK*) models (Tong, 1979). Nevertheless, the most commonly used model is the one of Mamdani (1974).

The following exemplary rule presents the process of fuzzy modelling for the case of two input variables and one output variable:

IF (u_1 is A_i) AND (u_2 is B_j) THEN (y is C_k) (1)

where the symbols represent:

A_i, B_j, C_k — fuzzy sets,
u_1, u_2, y — input and output variables, and
i, j, k — indices of fuzzy sets.

The TSK model differs from the Mamdani's one by the form of the rules.

IF (u_1 is A_i) AND (u_2 is B_j) THEN (y is $f(u_1, u_2)$) (2)

where:

$f(u_1, u_2)$ marks the function of the output (which can assume a linear or non-linear form).

The processes of fuzzy modelling for the case involving two inputs and one output are presented in Figure 2 (Zadeh, 1978).

These processes include: *fuzzification, inference*, and *defuzzification*. In the fuzzification process, the degrees of membership of crisp, model-input values (u_1, u_2) to the definite fuzzy sets (A_i, B_j) are calculated. Within the inference process, the membership function for the output value ($\mu_{Ck}(y)$) is computed based on the input degrees of membership ($\mu_{Ai}(u_1)$, $\mu_{Bj}(u_2)$). Finally, creation of the membership function for the output variable (y) takes place in the following stages:

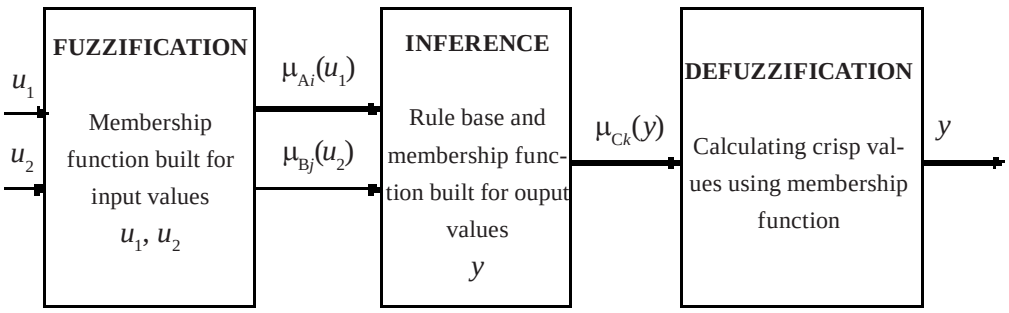
- Construction of the rule base;
- Activation of the conclusion mechanism;
- Definition of the degree of membership for the output value of the fuzzy model; and
- Calculation of the crisp value for the output value.

FUZZY *SPM* MODEL CONSTRUCTION

It is, thus, assumed that fuzzy models are constructed using the knowledge of SPM and project modelling. In the first case, the knowledge is acquired from project managers, while the latter one results from the specialists in the field of systems modelling. The knowledge of SPM can be applied in constructing formal methods of description, and, in particular, in building fuzzy models.

We also assume that a strategic milestone for the modeller lies in the selection of methods of formalising the knowledge of SPM. The famous statement of one of the originators of the fuzzy sets theory announces that "if the complexity of

Figure 2. Processes of fuzzy modelling for a case involving two inputs and one output



the system increases, then our ability to formulate accurate and also meaningful views of its behaviour decreases, until we reach a threshold value, beyond which precision and meaning become almost mutually exclusive features" (Yager & Filew, 1995, p. 167).

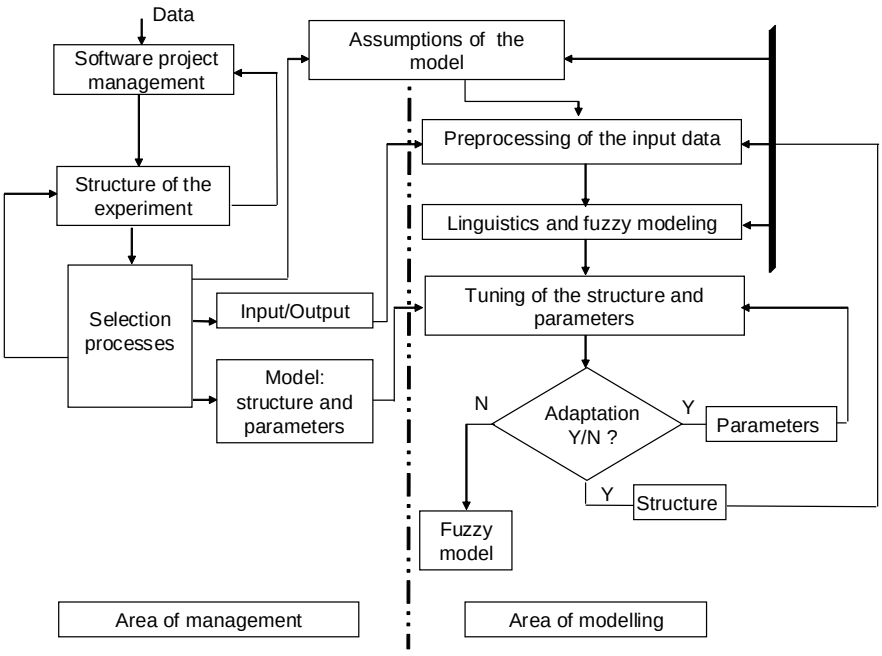
Making use of complex mathematical apparatus gives the possibility of precise description of repeatable processes mainly for technical systems (mechanical, electric). Whereas in the case of social systems, a departure from such precise descriptions is recommended by Łopaciński and Kalinowska-Iszkowska (1999), along with the use of techniques based on fuzzy logic. Therefore, also in the case of modelling socio-technical systems such as SPM, the apparatus of fuzzy set theory may be used both in the processes of formalising the knowledge and adapting the model. Then, the construction of *the fuzzy model of SPM* (Figure 3). can be treated as a process of continuous modelling with the use of fuzzy algorithms.

Keeping in mind the earlier assumptions, we have used the knowledge of the project manager (expert knowledge, methods and models) and the one of the modeller (basics of modelling and simulation, theory of fuzzy sets and regulators) in constructing a new prospective SPM model.

The aggregated knowledge of manager and modeller enables the processes of modelling SPM to be conducted along the lines of continuous modelling with the use of fuzzy sets. A detailed concept is presented in Figure 3, taking into account:

- Preparation of data concerning the structure and parameters of the model;
- Structural modelling on the linguistic level;

Figure 3. Continuous modelling of SPM with the use of fuzzy algorithms



- Basic tuning of the values of the parameters; and
- Possible adaptation of the parameters and structure.

Data Concerning the Structure of the Model

On the basis of the managerial knowledge, it is accepted that SPM can be implemented in four areas of management: knowledge, project processes, infrastructure, and supporting technologies (KPIT).

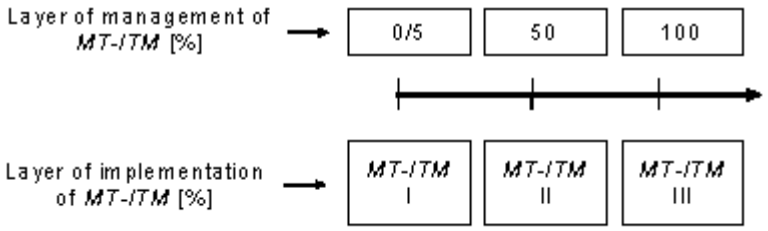
The SPM should also be analysed taking into account particular running phases of the project. Moreover, the scope of activity of managers should concern planning, selection, organisation, and monitoring of the *methods and tools of information technology and management* (*MT-ITM*). With regard to the crucial dynamic changes in MT-ITM, the use of the concept of variable states in describing those changes seems to be necessary, which needs also to be estimated by an expert of the MT-ITM according to the practice applied in SPM.

Three values of the level of exploitation of MT-ITM (in terms of management and implementation) are proposed, as shown on a linear scale in Figure 4. This distinction is applied to all the previously mentioned areas (KPIT).

1. The state of the MT-ITM management layer is described by a *scalar state variable*, which assumes one of three values. In the case of methods, they are within the range of 5% to 100%, and in the case of tools, from 0 to 100%. The choice of the scale for the methods (starting from 5%) results from the fact that the managing projects, without any method, is not possible. The choice of particular state values on the scale concerning the methods and tools of management depend accordingly:
   - On the number of team members who apply MT-ITM in relation to the number of all project team members (a composition co-factor, *ks*) at a given stage (in the case of managing infrastructure and knowledge);
   - On the number of implemented processes in which MT-ITM is applied in relation to the overall number of processes implemented (a process co-factor, *kp*) at a given stage (in the case of managing processes and supporting technologies).

On the basis of a weighted sum of the scalar states of the management layer concerning both methods and tools, we determined *a generalised scalar state of management of KPIT.* Note that

Figure 4. Defining the management states based on a linear expert scale for the two layers of KPIT



in calculating the generalised scalar states of management, we use weights, whose values are established on the basis of expert assessment.

2. Similarly, the TM-ITM implementation layer (determined by experts on the basis of the best practice in managing this type of projects) also covers the KPIT fields, but in terms of their implementation, and is assessed within three discrete values symbolised by MT-ITM I through MT-ITM II.

Using the Scale to Define the Generalised Scalar State of Knowledge Management

If the project manager achieves knowledge by means of direct talks with experts and formalises it with the use of diagrams or a rule description, the application of this method has considerable influence on planning the collection of knowledge. For the method of direct talks with experts, the scalar state of information and management methods is defined at the value of 5%, the formalisation of knowledge with the use of diagrams at 50%, and the rule description at 100%. The scalar state of information and management tools is defined similarly. Correct values are connected with the use of the co-factor *kp* resulting from the membership of the team. Next, the generalised scalar state of knowledge management (being the weighted sum of both values) is calculated.

In developing a concept of the model, several factors (such as "hand steering" of the composition of the team) have been ignored. Equally irrelevant appeared the so-far-identified phases of the project (definition and specification of demands, construction of the model, and its implementation). This could be easily justified because the manager initially chooses MT-ITM for all phases of the project and (if necessary) conducts a team training stage (Szczerbicki & Orłowski, 2003). With this, we assume that the manager is a specialist in a given field of IT and can appropriately select, exploit, and assess the information solutions applied in project management.

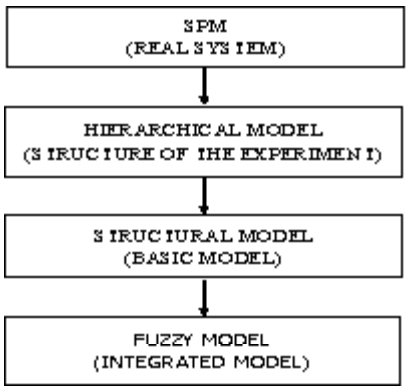
Data Concerning the Parameters of the Model

A concept of tuning the fuzzy model has been accepted, making use of SPM expert knowledge. The team manager collects the knowledge about the project management in suitable experimental knowledge base and records them in the form of rules whose structures correspond to the rules of the fuzzy model. The experimental knowledge base is classified with regard to the type of projects (e.g., successful, unsuccessful). It is clear that such a classification is performed by an expert in project management. Successful projects are defined as completed in the allotted time, with the agreed finance and aim. By using this kind of solution, we avoid misleading "averaging", which results in creating useless models.

The rules recorded in the experimental knowledge base are grouped according to the phases of the project. Such ordering influences a procedure of identifying clusters and creates the conditions for calculating the coordinates of the centres of gravity of the clusters, identified as coordinates of the peaks of membership functions. Adding new rules to the experimental knowledge base has an effect on a change of the centres of gravity of the clusters and, in consequence, on a change of the peaks of membership functions.

In the presented concept of the model construction, we also assume the processes of tuning the model parameters. The knowledge necessary for tuning should be gained from the managers and coordinators of the teams involved in the project realisation. The sources of knowledge are simply project documentations.

Figure 5. The process of constructing the fuzzy model



Structural Modelling on the Linguistic Level

A four-stage construction of the fuzzy model is proposed as shown in Figure 5. The first step is the analysis of the real system. It covers the management of the project consortium consisting of several or more project teams. In this paper, the management of the consortium is characterised by a basic structure of modelling experiments and a resulting *hierarchical model*. Next, a *structural model* is obtained, referring to the management concerning (the level of) the project team SPM_z. According to the theory of system modelling, this corresponds to a *basic model*. Next the *fuzzy model* is built (reflecting the so-called *integrated model*), which portrays the management processes based on the fuzzy rules. The applied concept of fuzzy modelling by giving a simple mathematical description of SPM_z , allows for obtaining a full hierarchy of management for the project consortium.

Tuning the Model Parameters

We have proposed a concept of tuning the membership function according to the project phases. The membership functions are tuned separately for the system input and output variables, while in the case of the state variables, the membership functions are fixed (in terms of the forms and the peak parameters). The construction of the membership function is controlled with the use of the clustering methods with the data from the concluded projects.

Adapting the Parameters

With the approach described earlier, it is easy to perform the adaptation of the model/system on two levels: (1) the management of the project and teams SPM_z and (2) the adaptation of the parameters of model SPM_z - *RFM* (*Software Project Management - Rule Fuzzy Model*). The first level of adaptation depends on the selection of "better" solutions of higher scalar values of the generalised management states, while the second level depends on the qualification of the output variables by the team manager according to the quality of team management.

As has been mentioned previously, with the proposed procedure and a great variety of the methods of implementation and various levels of quality in managing project teams and implementing information projects, we could easily finish off with "over-averaged" and, therefore, useless models. For this reason, in the concept of adaptation on the first level, we recommend a pre-selection of all projects carried out by an expert in SPM_z. The pre-selection can be done according to the quality of the project and team management, for example, leading to "successful" and "unsuccessful" projects based on the experimental knowledge base (see subsection, *Data Concerning the Parameters of the Model*). In

the second level, this influences the classification of input and output variables to appropriate models and the position of the apexes of the membership function. In conclusion, the method of modelling and adapting the model presented in this work *SPM$_z$ - RFM* allows us to:

1. Model various (established by the expert) types of processes, and
2. Create conditions for the project manager to use a *proper* model fitting to his knowledge about the level of managing the project team.

Finally, it can be worth mentioning that no adaptation of the model structure is assumed.

CONCLUSION AND ASSESSMENT OF EXISTING SOLUTIONS

The methodological solutions (methods PMM and KADS, models CMM and SPICE) provide formal approaches to supporting SPM by focusing on a collection of procedures for assessment of teams (the CMM model) or processes (the SPICE model). In the case of methodological solutions (PMM and KADS), we can use the project tools to implement IT systems, but not to manage them. The *SPM$_z$ - RFM* model indicate possibilities for managing the knowledge of IT projects and for using of IT methods and tools.

In the first case, the model *SPM$_z$ - RFM* (Kowalczuk & Orłowski, 2004) gives the possibility of selecting qualified solutions (on the basis of generalised management states) to the problems of realising project processes and functioning project teams. The latter case creates an integrated environment for ongoing assessment of both teams and processes. The criteria of MT-ITM for project teams increase the probability of selecting the best solution (on the basis of strict assessment). It also creates conditions for gaining knowledge, more effective rule-based processing, and increases the efficiency of the mechanism of cooperation between members of the project team. At the same time, it shortens production time (making group work mechanisms more efficient) and raises product quality (constant control of the produced software using *SPM$_z$ - RFM*).

REFERENCES

Coleman, D., & Somerland, P. (1994). *Object-oriented development: The fusion method*. Englewood Cliffs, London: Prentice Hall.

Hapke, M., & Jaszkiewicz, A. (1999). Integrated tools of programming project planning in uncertain conditions (in Polish). In *Proceedings of the 1st National Conference on Software Engineering* (pp. 65-76), Kazimierz Dolny, Poland.

Hickman, F. R., & Killin, K. (1994). *Analysis for knowledge-based systems. A practical guide to the KADS methodology*. London: Ellis Howard Limited.

Spice. (2000). *Software process improvement and capability determination*. Retrieved from http://www.sqi.gu .edu.au/spice

Kacprzyk, J. (1997). *Multistage fuzzy control*. New York: John Wiley Inc.

Kingston, J. (1992). Pragmatic KADS: Methodical approach to a small knowledge based systems project. *Expert Systems, 9*, 171-179.

Kowalczuk, Z., & Orłowski, C. (2004). Design of knowledge-based systems in environmental engineering. *Cybernetics and Systems, 35*(5-6), 487-498.

Łopaciński, T., & Kalinowska-Iszkowska, M. (1999). Aiding tools for planning processes and implementation of IBM software projects (in Polish). In *Proceedings of the 1st National Conference on Software Engineering* (pp. 145-149), Kazimierz Dolny, Poland.

Mamdani, E. H. (1974). Applications of fuzzy algorithms for control of a simple dynamic plant. *Proceedings of IEEE, 121*, 1585-1588.

Mesarovic, M., & Takahara, Y. (1989). Abstract systems theory. In *Lecture notes in control and information science, Vol. 116*. New York: Springer Verlag.

Nerson, J. M. (1992). Applying object-oriented analysis and design. *Communications of the ACM, 9*, 67-74.

Paulk, M. C. (1997). *Software capability maturity model, version 2* (Draft Technical Report). Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Pawlak, Z. (1997). Rough set and data mining. In *Proceedings of Intelligent Processing and Manufacturing of Materials, Vol. 1* (pp. 663-667), Gold Coast. IPMM Publisher.

Pfleeger, L. (2001). *Software engineering, theory and practice*. New York: Prentice Hall.

Słowiński, R., & Hapke, M. (Eds.) (1999). *Scheduling under fuzziness*. Heidelberg: Physica-Verlag.

Stoner, J. A. (1994). *Management* (in Polish). Warsaw, Poland: PWN.

Szczerbicki, E., & Orłowski, C. (2003). Qualitative and quantitative mechanisms in managing IT projects in concurrent-engineering environmental systems. *Analysis, Modelling, Simulation, 43*(2), 219-230.

Tong, R. M. (1979). *The construction and evaluation on fuzzy models in advances in fuzzy set theory and applications* (pp. 559-576). Amsterdam: North Holland.

W/glarz, J. (Ed.) (1999). *Project scheduling: Recent models, algorithms and applications*. Dordrecht: Kluwer.

Yager, R., & Filew, D. (1985). *Basics of fuzzy modelling and control* (in Polish). Warsaw: WNT.

Zadeh, L. A. (1978). Fuzzy sets as a basis for theory of possibility. *Fuzzy Sets and Systems*, 1, 3-28.

This work was previously published in the International Journal of Enterprise Information Systems, edited by A. Gunasekaran, Volume 2, Issue 2, pp. 1-12, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 6.9
Occurrence and Effects of Leader Delegation in Virtual Software Teams

Suling Zhang
Kean University, USA

Marilyn Tremaine
New Jersey Institute of Technology, USA

Rich Egan
New Jersey Institute of Technology, USA

Allen Milewski
Monmouth University, USA

Patrick O'Sullivan
IBM Dublin Lab, Ireland

Jerry Fjermestad
New Jersey Institute of Technology, USA

ABSTRACT

Virtual teams are an important work structure in software development projects. However, little is known about what constitutes effective virtual software team leadership, in particular, the amount of leader delegation that is appropriate in a virtual software-development environment. This study investigates virtual software team leader delegation and explores the impact of delegation strategies on virtual team performance mediated by team motivation, team flexibility and team satisfaction with the team leader. This research is a report of a pilot study run on student teams

carried out to refine and test the research constructs and research model for a larger study run in corporations. The study found that virtual team leaders delegate more to competent virtual teams and that such delegation is positively correlated with team member satisfaction with their leader and with team member motivation. Overall, the work provides important information for software-based organizations interested in developing virtual team leadership skills.

INTRODUCTION

Virtual teams are composed of geographically distributed coworkers linked though information technologies to achieve an organizational task (Townsend, DeMarie, & Hendrickson, 1998). Virtual teams are a popular structure in software development for several reasons: they provide access to lower-cost labor as well as access to a range of disciplines and technical specialties (Curtis, Krasner, & Iscoe, 1988). While software team leaders and managers are now frequently given virtual teams to manage, they have not been given clear directions on how to effectively manage such teams. One important issue regarding virtual software team management is when and how team leaders should *delegate* authority and responsibility to the team.

Delegation means that one has been empowered by one's superior to take responsibility for certain activities, which were originally reserved for the superior (Ashton & Kramer, 1980; Bass, 1990). In traditional leadership studies, delegation is widely acknowledged to be an essential element of effective management (Yukl, 2002), and effective delegation offers a number of potential benefits, both to the manager and the subordinates. However, to the authors' knowledge, only a few studies have been conducted to investigate delegation as a distinct component of global virtual team leadership. In the limited number of conceptual works and empirical studies in which delegation is not the direct focus, delegation has been a controversial issue. Some researchers argue for the benefits of delegation: Eveland and Bikson, (1988); Jarvenpaa, Knoll, and Leidner, (1998); Jarvenpaa and Leidner, (1999) report that an effective leader of a virtual team needs to be more flexible to accommodate the complexities and volatility of the virtual team environment, and to be willing to let others take the lead when necessary. Furthermore, they suggest that virtual team leadership should focus on facilitating and empowering team members to take action on their own. In contrast, Paré and Dubé, (1999) argue that, due to the distributed nature of virtual teams, management by observation is simply not possible, and that much more discipline and control is required in a virtual setting. Additionally, team effectiveness in virtual environments may be hindered by excessive autonomy coupled with exclusive reliance on electronic communication and lack of face-to-face interaction.

This article aims to address the research gap regarding delegation to a virtual software team by investigating the occurrence and effects of leader delegation in such teams. Finding out how and why leaders do or do not delegate to virtual teams and the impact of the leaders' delegation behaviors on the teams will help industry practitioners to better frame their strategy for managing distributed software teams and will also add to the field's knowledge about virtual team leadership.

The focus of the article is on software teams, in part, because it is felt that the global software team phenomenon has several unique characteristics that may not apply to virtual teams in general. Unlike other activities that have been outsourced or offshored, work activities cannot be as easily compartmentalized because of the high integration of the software product. Thus, there is a need for communication and working together in a structured fashion, which demands good leadership. In addition, software developers expect to have a high degree of independence in their work. Therefore, the degree of delegation

by a leader may differentially affect these virtual teams.

Collaboration in software development also demands the ability to communicate highly detailed specifications and questions. This requires communication skills that may not be needed in other types of virtual teams, skills which may be influenced by a team member's knowledge of English, the common language for many of these teams. Therefore, good management of the communication structure and media is likely to be an important leadership trait. Finally, the new countries that are now being included in global software development have relatively young team members. As such, the distribution of corporate knowledge and software skills is uneven. Thus, leadership and delegation in this type of environment is likely to be different for software than for other tasks carried out by virtual teams.

The article is structured as follows: first, an overview of the research model is presented to give readers a sense of the focus of the article; then, based on a review of the literature, conceptualizations of leader delegation are presented and specific research hypotheses regarding virtual team leader delegation are explained; finally, the pilot study testing the model is presented. A discussion section presents the contributions and limitations of this research, and a final section discusses the implications for virtual team management.

RESEARCH MODEL

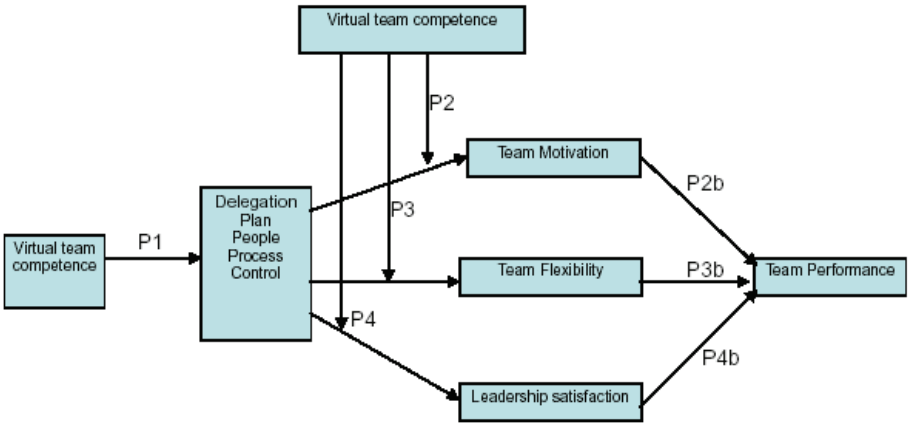
Before beginning a detailed discussion of the variables used in this study, we present the overall research model and briefly describe the relationships that are hypothesized to exist between the variables. A key focus of the model is the amount of delegation that virtual software teams receive from their leaders. It behooves us, therefore, to describe the management structures that are being studied.

When we talk about virtual teams, the structure that is typically in place does not constitute virtuality for each and every team member but, rather, distributed teams in which some subsection of the software development team is co-located and other sections are virtual. For example, a portion of the team may be located in the United States and a second and third portion in India and China, respectively. The overall team is working on the same software product, but the work has typically been compartmentalized in some way so that each co-located portion of the team has specific assignments. However, the work is such that there is a need for continued communication between each of the non-co-located portions of the team to resolve integration issues.

Management of these teams comes from several sources. First, there is typically a local manager who handles personnel issues. Then, there is a technical manager who oversees the project. Finally, there is a technical head of the particular subsection of the project who is directing the work of both virtual and co-located sub teams. Management of the work is the purview of this leader, and this is where delegation typically occurs. This management is located with one sub team and virtual to the other sub teams. Traditionally, management is located in the home of the parent company, but it also may be at a customer site.

Because at least, one part of the software team is distant from the manager, less is likely to be known about the competence of that part of the team by the manager, and therefore, he or she is likely to delegate in a manner so that majority control of the technical issues rests with that portion of the team that is co-located with the manager. The research model being put forward in this paper suggests that perceived team competence predicts the amount of delegation that will occur but that this delegation will have an effect on key team variables such as satisfaction, motivation and flexibility. It is argued that these three variables

Figure 1. Proposed model of delegation effects on global software teams



are primary in affecting team performance. Figure 1 presents our research model.

An elemental part of this model is that team competence not only affects the decision of the team leader on whether to delegate or not, but also affects the impact of the delegation. That is, teams with less competence, are likely to desire less delegation and to be unhappy and unmotivated with less direction from the leader. This occurs primarily because the team members do not know how to perform their tasks completely, yet wish to succeed in these tasks. However, the only way they can succeed is if they receive tighter direction from the leader. As we shall see with our analysis, we did not necessarily find that less competent teams wanted more delegation. The measure of perceived competence that we used had difficulties with the student teams used in this research. All student teams perceived themselves to be highly competent. Rather than treating this as a problem, it is a result that we believe has an impact on virtual team management.

Virtual subteams are not able to ascertain their competence in comparison to their distant counterparts because they lack the proximity required for an accurate comparison. They are, therefore, likely to perceive themselves as competent and desire more delegation. Culture may also have an impact in that many cultures worry about "face," that is, how they appear to others. These cultures would also perceive themselves as competent, possibly due to team members' advanced education in their country or membership in a high social class. Whatever the reason, they are likely to desire more delegation even when it is unwarranted. This issue will be further addressed in the paper's discussion section.

The next sections present prior research on the relations posited in our model and detailed definitions of each of the variables used in the model plus support from the literature on the validity of the constructs we have developed for the variables.

LITERATURE AND HYPOTHESES

Delegation

As mentioned earlier, delegation is a state or condition in which one has been empowered by one's superior to take responsibility for certain activities, which were originally reserved for the superior (Ashton & Cramer, 1980). There is a rich body of studies in traditional leadership research investigating delegation, mostly as a feature of leadership style or as a combination of related leader behaviors. Very few studies investigated delegation as a distinct management practice (Yukl & Fu, 1999; Zhang, Tremaine, Fjermestad et al., 2006). Because of this, we cannot ascertain the effect size of delegation (as a distinct leadership component) on possible variables of interest.

Another major limitation of previous delegation and leadership studies is the failure to focus on what types of responsibilities and activities a leader delegates. Only the global delegative style of the leader was assessed. This limitation seriously undermines the practical value of these studies, as their findings do not inform software team managers where they can and cannot delegate nor does it tell them what they can delegate. Leadership is a multi-faceted process, particularly so in virtual teams, given the technological, cultural and organizational complexities of a virtual team environment. Therefore, it is important to examine the leadership and managerial functions originally assigned to leaders to determine which of these can be delegated to followers. To that end, studies that describe leadership and managerial functions are reviewed in this section, and a list of leader delegation categories is generated.

For classical management theorists like Davis (1942) and Urwick (1952), the functions of the manager-leader in a formal organization were orderly planning, organizing and controlling. To address the overlapping needs of the organization, team and individuals, Coffin (1944) modified the classical functions as follows: formulation (planning), execution (organizing), and supervision (persuading).

MacKenzie (1969) proposed a well-known leader-manager model that illustrated the great variety of activities that a typical manager performs. He proposed that the central management functions relate to the management of people, ideas and things, which form the three basic components of every organization with which managers must work. Three functions (the analysis of problems, decision-making and communications) are important at all times and in all aspects of the jobs held by managers, and therefore permeate the entire work process. To carry out these functions, the leader-manager needs to execute these leader activities: planning, organizing, staffing, directing and controlling. A factor analysis reported by Dunnette (1986) of 65 managerial activities yielded seven factors: monitoring the business environment, planning and allocating resources, managing individual performances, instructing subordinates, managing the performance of groups, representing groups and coordinating groups.

Janz, Colquitt, and Noe (1997) identified four distinct facets in which a leader could give the team autonomy: planning (e.g., scheduling the team's work), product (e.g., suggesting new products or services), people (e.g., recruiting and hiring members) and process-related (e.g., specifying the development method a team should use).

Hertel, Geister, and Konradt (2005) reviewed a collection of empirical works on the management of virtual teams and summarized the management functions of virtual team leaders at various phases in the team's lifecycle. They proposed that virtual team leaders are generally engaged in such activities as *personnel selection, task design, team initiation, performance management, training and team development, and disbanding and re-integration.*

These studies categorize leader functions in different ways and use different labels for these leader functions. Integrating these different

Table 1. Four-dimensional leader delegation framework

Delegation Aspects	Key Activities
Planning-related	Scheduling the team's work Setting the team's long-term goals Setting the team's short-term objectives Setting the team's budget
People-related	Selecting team members Removing members from the team Determining team members' training needs
Process-related	Assigning work to team members Selecting the tools team members will use in their work Determining the team's operating procedures and work instructions
Control-related	Evaluating the progress of the team's work Evaluating team product quality Determining corrective actions when performance objectives are not met

categorizations, Table 1 summarizes four major leader function categories which can be delegated to virtual team members. These four areas are the overlapping important leader-manager functions identified in the mentioned studies. The first leader delegation category consists of *planning-related* team management and leadership activities that a virtual team leader can possibly delegate to the team. The second category consists of *people-related* team management and leadership activities that a virtual team leader can delegate, such as team staffing and team member training. The third category consists of *process-related* team management and leadership activities (or teamwork process management). The fourth is *control-related* team management and leadership activities. This category relates to the leader's functions and activities that control the work progress and quality of a virtual team.

To keep the categorization parsimonious, the key activities in each delegation category may incorporate more than one of the leader functions identified in the previous studies. For example, "determining operating procedures and work instructions" incorporates two virtual leader functions in MacKenzie's model: standardize methods and decide how to achieve goals. Some of the manager activities are not included because they either cannot be assigned to the team through leader delegation or are already tasks that are done by team members, for example, suggesting new products or services.

Delegation in Virtual Software Teams

The issue of productivity loss is very important when global software team leaders weigh the benefits and costs of delegation. A software team

is often formed for completing certain tasks within a limited time period. The short-term nature of the project requires avoidance of any productivity loss. In a longitudinal study using 18 student information system development (ISD) teams, Nicholson et al. (2002) confirmed the importance of avoiding productivity loss in distributed teams. The researchers attempted to identify the characteristics/behaviors of effective ISD project team leaders/managers. Their exploratory analysis revealed that face-to-face (FTF) and virtual ISD team members valued different ingredients of leadership in different phases of the ISD project. They found that a key ingredient for virtual team leadership was having a realistic leader who managed time efficiently. Global virtual teams involved in ISD are temporary structures that are very focused on the development of the information system application. These teams have tremendous time constraints. Hence, the role of a virtual project team leader is to remain practical in terms of the goals set and the deliverables promised, and to manage the time allocated to each task efficiently. Therefore, to avoid productivity loss, global software leaders are likely to delegate more to competent teams than to incompetent teams. A competent virtual software team has the knowledge, attitudes and expertise required to perform the team tasks and is more likely to plan well and manage their work efficiently (Faraj & Sambamurthy, 2006). A competent team can quickly apply their expertise to the delegated task and reduce productivity loss to a minimum. In contrast, delegation to incompetent teams means that leaders have to spend additional time giving feedback and monitoring the execution of delegated tasks to ensure that performance standards are met. Otherwise, managerial anxiety over loss of control can be overwhelming. However, in the distributed global team environment, close monitoring and timely feedback is difficult as "management by walking around" can not be used as a strategy (Paré & Dubé, 1999). Due to increased temporal distances, possible increased cultural distances and the lean nature of computer-mediated communication in virtual teams (Bell & Kozlowski, 2002), leaders will need to spend much more time and effort in coordinating, monitoring and coaching team followers in the delegated tasks.

Besides the need to manage time efficiently, the matrix organizational structure of global software teams is often another reason why team leaders delegate more to competent team members. Developing subordinates' skills and confidence is the biggest reason why leaders delegate or consult their followers, especially when followers' skill sets are still to be developed (Yukl & Fu, 1999). The potential growth of the followers is likely to be the major benefit leaders obtain from delegation to an incompetent team. However, virtual teams are often designed to cross geographical and organizational boundaries to allow dispersed organizations to maximize their expertise without having to physically relocate individuals. The required expertise for a given task or project may be dispersed at multiple locations throughout the organization; however, a global software team may facilitate the 'pooling' of this talent to provide focused attention to a particular problem without having to physically relocate individuals (Kayworth & Leidner, 2000). Therefore, global software teams are often dispersed in cities or even countries and, very commonly, the team followers do not report to the team leader in a direct organizational line. Due to the typically short-term nature of the team and the complexity of the reporting relationship, global software team managers are more likely to be evaluated on their success in achieving project goals, rather than on their development of team members. In delegating to incompetent teams, global software team managers—unlike line managers who may treat the costs of delegation as an investment to be redeemed later—are faced with the cost of sacrificing team performance, which determines the manager's own promotion and career growth. As managing a matrix structure is already chal-

lenging for virtual team leaders (Oertig & Buergi, 2006), the costs associated with delegating to an incompetent team would tend to deter virtual team leaders from this management strategy. Based on these arguments, the following hypothesis is proposed:

Hypothesis 1: *Leader delegation will be positively correlated with global software team competence.*

The Effects of Team Leader Delegation

In this section, three variables, motivation, team flexibility and team member satisfaction with their software team leader are each discussed. An argument is made as to why each of these variables is expected to be affected by team leader delegation and also why each of these variables is a primary determiner of overall team performance.

Motivation: According to Herzberg's (1968) motivation theories, recognition fulfills workers' esteem needs and can significantly improve employee's performance. A competent virtual team typically expects the team leader to recognize the team's competency by delegating more responsibility. Leader delegation will then improve the team's sense of self-worth and motivate the team to work more effectively. An empirical study found that the autonomy of virtual team members in determining work objectives and methods improved the intrinsic motivation of the team (Kirkman, Rosen, Tesluk, & Gibson, 2004). They investigated the relationship between team empowerment and virtual team performance and the moderating role of the extent of face-to-face interaction using 35 sales and service virtual teams in a high-technology organization. They found that team members are more intrinsically motivated when they believe they have high *performance capability* (**competence**), *high responsibility* and *authority* to carry out work (**delegation**), and *a meaningful task* that could impact the organization. Piccoli and Ives (2003) found that student virtual teams were more motivated and satisfied with less behavior control. Also, a team's acceptance of a decision is highest when the decision is made by the group (Curtis et al., 1988). Therefore, we posit that delegation to a competent virtual team would increase the team's motivation

On the other hand, delegation to less competent virtual teams will put the team in a difficult situation. Due to their low competence level, they need close monitoring and constant coaching from the leader or other experts, which is difficult and costly to obtain in dispersed virtual teams. Kayworth and Leidner's (2002) observation of a dichotomy between inexperienced members wanting and expecting strong directive leadership and team leaders who, faced with the practical constraints of distance and the lack of a direct management line, would prefer members to be self-managing. Such dichotomy hurts the team members' motivation and performance. In addition, the relatively short-term nature of virtual teams, which are often formed dynamically to cope with emerging projects or tasks, also means that the team has less time to learn on the job. Instead of desiring delegation, less competent virtual teams want detailed directions from the leader.

Therefore, we posit that delegation to less competent virtual teams may not improve team motivation. Based on these arguments, Hypothesis 2a is put forth:

Hypothesis 2a: *Delegation to competent virtual software teams will improve team motivation more than delegation to less competent virtual software teams.*

Team performance, as with individual performance, is a function of ability and motivation (Jarvenpaa et al, 1998). Sridar, Nath, Paul and Kapur (2007) have shown that team member motivation and trust affect performance in student teams

distributed between the United States and India. Significant improvement in team performance is therefore expected from motivated teams. Based on this argument, Hypothesis 2b is put forth:

Hypothesis 2b: *Virtual team leader delegation indirectly improves team performance through improving virtual team motivation.*

Flexibility: Team flexibility refers to how flexible a virtual team is in responding to environmental and personnel changes. Most virtual teams are knowledge-based teams which are formed to solve customer problems or to develop new products (Kirkman et al., 2004). The complex, knowledge-based tasks many virtual teams perform require behaviors such as planning and executing, managing team performance, improving team processes, and influencing organization-level direction and resource allocations (Mohrman, Cohen, & Mohrman, 1995). In conducting these activities, teams have to make sense of their tasks, improvise their work processes, and adjust how they make progress toward agreed upon goals. Therefore, flexibility is important to virtual team performance and team leaders should delegate to competent virtual teams to allow them to flexibly adapt to their immediate situations and opportunities. Remote team leaders may not be able to understand the work context or to appreciate the consequences of the changes occurring in the distributed locations. Delegation, therefore, puts this task in the hands of the competent virtual team members. These members can also make decisions in a more timely matter than the leader. Having appropriate authority delegated to them, they can proactively influence team leaders' decisions or even their own decisions instead of passively waiting for managerial permission before taking actions. Therefore, delegation to competent virtual teams will increase team flexibility. In contrast, delegation to less competent virtual teams may not improve team flexibility since less competent team members may not have the skills to make decisions and form action plans. Based on this argument, Hypothesis 3a is put forth:

Hypothesis 3a: *Delegation to competent virtual software teams will improve team flexibility more than delegation to less competent virtual software teams.*

When a virtual team flexibly adapts to its work situations and is free to respond to situations in at timely manner, it will be more risk-taking and learn from experience to continuously improve its work processes and to perform in more efficient ways. Based on this argument, Hypothesis 3b is put forth:

Hypothesis 3b: *Virtual team leader delegation indirectly improves team performance through virtual team flexibility.*

***Satisfaction with Software Team Leader*:** Delegation to competent virtual teams represents the leaders' recognition of their competence. Delegation allows the competent team to utilize their capabilities to adapt to the immediate opportunities and changes without waiting for decisions to be made by the distant leader. Therefore, delegation to a competent virtual team should improve the team's satisfaction level with the leader.

On the other hand, delegation to a less competent virtual team may decrease the team's satisfaction with the leader. In a global virtual team study consisting of undergraduates as followers and experienced MBA graduate students as team leaders, Kayworth and Leidner (2002) found that the inexperienced undergraduate followers were more satisfied with leaders who gave clear, detailed instructions and feedback. Faraj and Sambamurthy's (2006) study of 65 software development teams found similar results. Their study showed that empowering leadership has an important positive impact on team performance

but only under conditions of high task uncertainty or high team expertise. In the software teams they studied, when team members have significant levels of professional experience with software development, they are more likely to possess the relevant expertise and experience for managing their project activities. For such teams, an empowering leadership approach might be more appropriate since members may need less direct control and coordination and possibly possess as much relevant task expertise as the team leader. In contrast, when teams have low professional experience, a directive leadership might be more appropriate because the members look to the leader to provide needed directions and guidance about their work activities.

As software team distance increases, the work context exhibits increased complexities, for example, cultural misunderstandings, communication difficulties, and so forth. This makes virtual teamwork more daunting for a less competent team. Under such circumstances, the team needs to attain confidence and a sense of direction from a strong leader. Based on this argument, Hypothesis 4a is put forth:

Hypothesis 4a: *Delegation to a competent virtual team will improve the team's satisfaction with their team leader more than delegation to a less competent virtual team.*

When the virtual software team members are satisfied with the team leader, the team leader will be more able to influence the members to work towards team goals and therefore to improve team performance. This has been confirmed in empirical studies (e.g., Zeffane, 1994).

Based on this argument, Hypothesis 4b is put forth:

Hypothesis 4b: *Virtual team leader delegation indirectly improves team performance through improving virtual team's satisfaction with the team leader.*

RESEARCH STUDY

Study Design and Sample

A full-scale study with industry software development teams is currently underway to explore the presented research hypotheses. We report here on the pilot study conducted to test the validity and reliability of the constructs that were formed for this research and the viability of our research model. Although many of the questions that were used came from studies that had already tested their validity and reliability, there were modifications made to the questions to (a) fit the virtuality nature of the teams being studied and (b) to fit the software development environment. For example, the constructs of team competency and leader delegation were adapted to describe competency in terms of software skills and portions of leader delegation in terms of the assignment of software tasks

Student teams were used to pilot the research survey because the diversity of the student teams closely matched the software development team populations that the final survey is intended for. The student teams are part of a computer science and information systems program at an American, east coast university that has one of the most diverse student bodies in the United States. The students are primarily from China, India, the Middle East, Brazil, and Pakistan.

Forty-eight students in 30 software-development teams took an online survey that requested information about the variables presented in the hypotheses. Thirty-two males and 16 females; three graduate students and 45 undergraduate students participated. All participants were involved in teams that were engaged in developing a single software program for the entire semester. The software teams are part of the ABET accredited Capstone course designed to have students working in teams on real software projects before they matriculate. This is a course most students take in their last year of school. Companies are

solicited for software projects and present their projects to the class. A team leader volunteers for a software project and then interviews and accepts members for his or her team. Teams normally have four members but some teams have three or five members. Teams then meet with the company representative and develop requirements, a budget, a design and a deliverable tested product. Reports are due, including a management report with assigned team roles, at regular intervals. Thus, the teams are set up to behave as much as possible like typical software development teams. Our interviews with corporate management in software companies indicates that projects rarely last for more than six months and that when new teams form, the membership is also new, so our student teams represent this type of project assignment.

The research instrument is designed to work with virtual teams. Thus, it may be asked, how can this student population constitute a virtual team? Almost any team at the university studied is partially virtual because of the nature of the university. All capstone classes are held in the evenings because team members usually are part time students. The university is a commuter university with many classes online or partially online so that students may come to the university once a month. Thus, many of the teams meet virtually and much of the team work is done by e-mail, instant messaging and teleconferencing. It was felt that with the more mature students, seniors, the partial virtuality and member multiculturism presented a population suitable for piloting the research.

The survey was given near the end of the semester so the team members had worked together for about three months.

In a second administration of the survey, 34 graduate students from 14 report-writing teams took the online survey. The team task was to analyze an industry case study and write a team report based on the case study results. The teams consisted of five to six members with team leaders elected by the team members. The survey was given after the team finished their first case study project. These 14 teams were taking a two-month summer online management information systems course.

Only two of these teams reported meeting face-to-face once a week. The other teams did not report meeting face-to-face during the team project and may never have met each other face-to-face. This second set of teams was also different from the first in that they were working on a report rather than developing software. Thus, they met the requirement of being virtual for our pilot study but not of being a software development team. We use this second team to determine if the difference in virtuality might have an influence.

After the survey, open-ended interviews with six members from two software-development teams were conducted face-to-face. Each interview was conducted by two researchers and videotaped with the interviewee's permission. The interview results are reported in section 4.4 and 4.5.

Survey Measurement

In this section, each of the constructs used in the research is described in more detail, in particular, its source for validity and reliability verification are cited along with a sample question that presents the intent of the measure.

Delegation: The four categories of virtual team leader delegation were measured by 13 7-point Likert-scale items in the survey. Section 3.1 introduced the conceptualization of the four categories. A sample question statement is "how much is your team able to schedule team work?" (completely – not at all)

Team Competence: In the first round of the survey, six questions were used, which are adapted from the situational leadership measurement of follower ability (Hersey & Blanchard, 1988) and Hardin, Fuller, and Valacich's instrument of virtual team efficacy (2006). A sample question is

"The team has past experience related to the team job" (strongly agree – strongly disagree). In the second round, six 7-point Likert scale questions were used to assess specific skills important to team tasks. A sample question is "how do you evaluate your team on its critical analysis skills?" (extremely high – extremely low)

Team Motivation: Four 7-point Likert scale items measuring this construct are adapted from situational leadership theory (Hersey & Blanchard, 1988). A sample questions is "The team is motivated to take on additional responsibilities if needed to finish the project" (strongly agree – strongly disagree).

Team Flexibility: Three 7-point Likert-scale items were created by the research team consisting of five researchers with extensive virtual team research experience to measure this variable. A sample question is "This team quickly responds to new opportunities" (strongly agree – strongly disagree).

Team's Satisfaction with the Team Leader: Three 7-point Likert-scale items were created by the research team to measure this variable. A sample question is, "I am dissatisfied with the way the team leader manages this project" (strongly agree – strongly disagree).

Team Performance: Team performance is measured by five 7-point Likert-scale items adapted from Henderson and Lee's (1996) study. This is a composite measure which reflects five important areas of software team work outcomes including: the amount of work the team produces, the efficiency of team operations, the team's adherence to schedules, and the team's adherence to budgets, the quality of work the team produces. This measurement was examined by five experienced researchers and three IT managers with extensive virtual team experience and was considered by them to provide a valid and comprehensive view of software team outcomes. A sample question of this measurement is "Compared to other projects you have served on or observed, how do you evaluate your team's performance on adherence to schedules" (Extremely high – extremely low).

Other Contextual Variables: In addition to these constructs, the study also captured data on another two variables: trust towards other team members, and interdependence of tasks performed by team members, which were found to moderate the impact of leader delegation (Yukl & Fu, 1999) Trust is measured with four 7-point Likert scale items from Jarvenpaa et al.'s study (1998). A sample question measuring trust is "If I had my way, I would not let the other team members have any influence over issues that are important to the project" (strongly agree – strongly disagree). Task interdependence is measured by two items adapted from Campion, Medsker, and Higgs (1993). A sample question is "To what extent do the team members have to share work materials to get the project done?" (strongly agree – strongly disagree). Finally, team background information such as team member's age, year in school, how often the team met and how they met (remotely or face-to-face) was also gathered. This information is used in the post-hoc analysis to help us in our interpretation of the resulting relationships uncovered in the research model.

Preliminary Data Analysis

Delegation Construct Structure: A principle component analysis (PCA) was conducted to test if delegation is a four-dimensional variable. However, in the pilot study, the student teams did not have budget constraints and were not allowed to change their membership once the teams had been established. Therefore, the four leadership and management function measurements are not included in the data analysis done with PCA because students answered these questions as not applicable. PCA results show that all the remaining nine items measuring delegation load on one component instead of three unique components. This indicates that virtual team leader delegation

in the student teams is not multi-dimensional counter to our predictions.

Measurement Validity: The numbers in the study were too small to conduct a factor analysis, but wherever possible, the questions used for the constructs were drawn from previously validated surveys. We also checked the constructs for face validity by reviewing the questions with twenty experts from the countries where the virtual teams for the full study were located. In addition, a card sorting test was performed on the constructs using 20 respondents. Ninety-five percent of the questions were sorted correctly supporting a case for acceptable construct validity. Finally, external validity is a concern because students were used in the study. We treat this issue in the discussion on this research's limitations.

Measurement Reliability: Except for trust (α=0.409), the Cronbach Alphas of other construct measurements are above the level of 0.8. Trust is therefore not included in further data analyses.

Within-Team Agreement: Due to the small sample size, a simple measure was used to judge within-team agreement level: individual team members' responses were considered to have an adequate level of within-team agreement and were averaged to obtain a team score if the difference between the highest score and the lowest score in a team was less than 2.5 (half the scale range). There was a high level of within-group agreement in more than 85% of the 44 teams on all constructs in the research model. Therefore, individual team member's responses are averaged to get team-level data.

Test of Hypotheses

Multiple regression testing was chosen to analyze the data as the data met normality and homogeneity of variance requirements (An arcsine transformation was carried out on the team competence and performance measure to achieve these assumptions). Structured equation modeling was not used because of the small sample size and the intent of the study (pilot). Although a PLS model is more likely to have more accurate beta scores, it also has a greater chance of a Type I error (Goodhue, Lewis, & Thompson, 2006). Since this was a pilot study, we wanted to bias it against possible spurious results.

Hypothesis 1 Test: Hypothesis 1 predicts that virtual team leaders delegate more to competent virtual teams than to less competent virtual teams. A multiple regression analysis was conducted to test this hypothesis. The test results shown in Table 2 support Hypothesis 1. Delegation is positively correlated with team competence.

Hypotheses 2, 3, and 4 Tests: Hypothesis 2a predicts that leader delegation to competent virtual teams will improve team motivation more than delegation to less competent virtual teams. Hypotheses 3a and 4a predict the effects of virtual team leader delegation on team flexibility and on a team's satisfaction with team leader respectively. The three hypotheses were tested by stepwise regression with the outcome variables regressed on delegation, team competency and the interaction term of delegation and team competency.

Table 3 presents the test results in the *software-development teams. Hypotheses 2a and 3a* are not supported, as no significant interaction effects were found. Therefore, the effects of delegation on team motivation and team flexibility were not found to change as team competence varied. Regarding *Hypothesis 4a*, the results show that leader delegation improves the team's satisfaction but such effects do not change as team competence varies. In addition, team competence was found to significantly improve team motivation, flexibility, and team satisfaction with the leader (p<0.05).

Table 4 shows the Hypotheses 2a, 3a, and 4a test results in *the report-writing teams.* Regarding *Hypothesis 2a*, Team leader delegation significantly improves team motivation (p=0.006). However, the effects of leader delegation on team

Table 2. Hypothesis 1 test results in both report writing and software development teams

Delegation Regressed on Team Competency		
	Software Development Teams	**Report Writing Teams**
Standardized Coefficient	0.508***	0.706***
R-Square	0.258	0.498
F-Overall	8.706***	9.913***
**p < 0.10, ** p < 0.05, *** p < 0.01*		

Table 3. Hypotheses 2a, 3a, and 4a test results in software-development teams

	Motivation	**Flexibility**	**Satisfaction**
Delegation	0.537	0.821	1.502*
Competency	1.536***	1.423***	1.708***
Delegation X Competency	0.11	-0.153	-0.231
R-Square	0.75	0.54	0.65
F-Overall	21.93***	7.91***	9.94***
*p < 0.10, ** p < 0.05, ***p < 0.01			

Table 4. Hypotheses 2, 3 and 4 test results in report-writing teams

	Motivation	**Flexibility**	**Satisfaction**
Delegation	0.782***	0.216	1.502***
Competency	0.126	0.069	1.708***
Delegation X Competency	-0.433	0.102***	-0.231***
R-Square	0.531	0.687	0.561
F-Overall	12.263***	21.912***	11.502***
*p < 0.10, ** p < 0.05, *** p < 0.01			

motivation did not differ as team competence level varied. *Hypotheses 3a_and 4a* are supported by the regression results. The interaction effects of delegation and team competence are significant such that leader delegation to competent virtual teams improves team flexibility and team satisfaction with the leader more than delegation to less competent virtual teams.

Hypotheses 2b, 3b, and 4b Tests: Hypotheses 2b, 3b, and 4b predict leader delegation indirectly improves virtual team performance through improving team motivation, flexibility and satisfaction with leader respectively. A Sobel test determines the significance of the indirect effect of the mediator by testing the hypothesis of no difference between the total effect and the direct effect. This method is used because its superiority of reducing Type 1 error and increasing power (Mackinnon, Lockwood, Hoffman et al., 2002). Due to the small sample size of the study, a bootstrapping Sobel test was performed.

For the ***software development teams***, the tests did not find delegation improved motivation and flexibility. Consequently, the effect of delegation motivation and flexibility improvements on virtual team performance was not testable and therefore, *Hypotheses 2b* and *3b* are rejected. *Hypothesis 4b* is supported by Sobel test results, as shown in Table 5. Therefore, delegation improves team performance partly through improving team satisfaction with the leader.

For the ***report-writing teams***, *Hypothesis 2b* is not supported by regression test results, as shown in Figure 2. Regression tests did not find that motivation had a significant positive impact on performance. Therefore, delegation was not found to improve team performance indirectly through improving team motivation. Previous tests on *Hypotheses 3a* and *4a* did not find that delegation improved team flexibility and satisfaction with the leader. Consequently, delegation cannot be tested as to whether it improves virtual team performance through improving team flexibility and satisfaction with leader. Therefore, *Hypotheses 3b* and *4b* are not supported either.

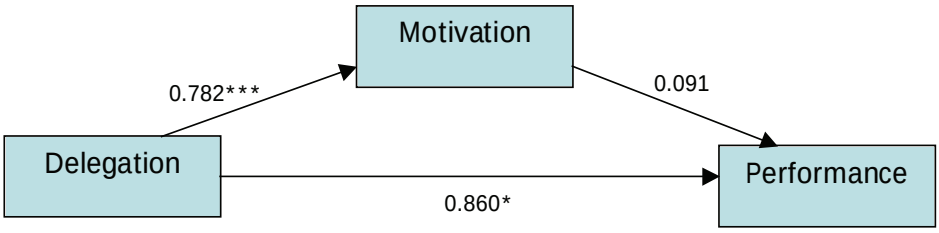
Interview Data Analysis

The interviews were analyzed in two steps. First, two PhD students and two professors with ex-

Table 5. Sobel test results on hypothesis 4b in student software-development teams

Sobel Statistic	F Value	Percentage of Total Effects that are Mediated	Ratio of the Indirect Effects to the Direct Effects
2.868	0.004	77.34	3.412

Figure 2. Hypothesis 2b test in report-writing teams



tensive research experiences watched the videos of the interviews and discussed what they found relevant to this study in a group meeting. Each of them explained his/her findings to the group and the group discussed the validity and implications of these findings. These discussion results were recorded in writing. Second, one PhD student summarized these group discussion results and combined them into a set of statements. Then, she watched the interview videos for the second time. She put down quotes from the interviewees related to each statement, examined these quotes to judge how they supported or disproved the categories of findings and modified the results of the analysis accordingly. This two-step procedure does not transcribe the interview data but roughly follows the process of creating a coding scheme, coding the data, iteratively improving the coding scheme and drawing conclusions. This procedure uses insights made by multiple researchers who separately viewed the interview videos and therefore should be adequately rigorous for a small-scale pilot study.

The interview data analysis found overall that delegation was an important part of team leadership and affected team outcomes in various ways. The following paragraphs present the detailed findings.

First, team leaders were aware of the importance of delegation, and engaged in delegation for a variety of reasons. One team leader learned from his previous military training that leaders should teach their subordinates to do their job so that they can take over if the leader is not available. The other reason both team leaders gave for delegating to the team was a belief that the team had the capabilities to perform the work. This belief in the team's competence came from knowledge about team members' past performance and their professional experiences. Also, one team leader recognized that competent team members have egos and decided that being too controlling would hurt such egos. Lastly, at least one team leader felt that delegating tasks to incompetent team members with the added support of fellow team members could be a way to build up a team member's self-confidence and skills.

Second, as the team develops, a leader's delegation style may change. In both teams the author interviewed, the leaders' style changed from one of controlling to that of delegating as team member's competence increased. In one team, members reported that at the beginning of the project, the team leader "tried to control everyone and everything in the project." When the leader was traveling, he sent team members detailed e-mails that specified the work to be done and the precise methods to use. As his team members demonstrated their capability and produced several deliverables ahead of deadlines, the leader "eased off in the middle." Near the end of the project, the team leader was very trusting and allowed the team members to self-lead. One of the most competent members in the team even led the team for two weeks when the leader was on vacation in a foreign country.

Third, delegation happened with monitoring and coaching. After certain tasks or functions were delegated, the leaders monitored how the team performed and coached the team members when needed. One team leader, for example, suggested books for the team members to read and advised the team on available development tools.

Fourth, delegation affects team motivation and a team's satisfaction with the leader. Within one team, a team member missed the first deadline and produced very low-quality deliverables. The team leader took action and became very directive setting detailed work schedules for the team, making detailed work assignments to team members and closely monitoring the work quality and progress of the team. Such non-delegation drove the team members to work harder. In this case team members appreciated the team leader's direction and effort to get their project started. In the other team, team members had adequate professional experience and the skills needed to perform the project tasks. In the beginning the

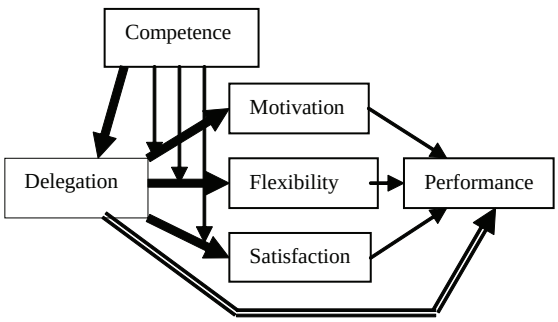
team leader was very controlling and the team members complained and even had a confrontation with the leader. However, as the team leader became aware that he was directing a competent team, he switched to negotiating with team members when making decisions. With this change, the team members felt his trust in their abilities and became more satisfied with the leader.

DISCUSSION

The following hypotheses were supported by the study. First, virtual team competence predicted leader delegation behaviors. This implies that virtual team leaders should carefully evaluate a team's competence before delegating tasks, especially before delegating important tasks. Second, leader delegation improved the satisfaction and motivation of team members. The effects of delegation on flexibility and satisfaction were more prominent in competent report-writing teams than in less competent report-writing teams but these effects were small. Other hypothesized results were not found. None of the intermediate outcome variables (flexibility, motivation, satisfaction) mediated the effects of delegation on team performance. Instead, tests results indicated that delegation directly improves virtual team performance. Figure 3 presents the model supported by the study results.

Comparing the results found in the two types of teams, one will observes that first, delegation exerted deeper influence on the report-writing teams than the software-development teams. It is suspected that the differences in delegation effects may arise from the differences in the number of times the teams met face-to-face or, as suspected, the degree of virtuality of the team. In contrast to the software development teams, which met face-to-face at least once a week, the report-writing teams barely met. Students in the report-writing teams were in an online summer course and, throughout the project, only two teams met face-to-face once a week. As collaboration and communication processes suffer from lack of face-to-face contact, the leader's role in team coordination and communication becomes more important. Therefore, leader delegation produced deeper effects in the more virtual report-writing teams. This suggests that leader delegation is likely to be a very significant factor in managing virtual teams.

Figure 3. Delegation effects on student teams. (Bold lines indicate supported hypotheses. The double line indicates a new result showing a direct link between delegation and team performance.)



Limitations of Study

One of the major limitations of this study is that the teams were student teams working on class projects. In particular, some of the questions that were designed for corporate virtual teams were not applicable. Student teams do not typically assign salaries, manage finances or hire and fire personnel. Only the planning part of the delegation construct showed differences between teams. The other parts were scored as not applicable (one of the possible answers) so that the results were pooled into one construct called Delegation which consisted mostly of items that addressed planning delegation.

In addition, the software and report writing teams did not represent the distributed teams that the survey was prepared for. There were some examples of distributed teams, for example, one-half of the team lived in the southern part of the state and the other half lived in the northern part, but, by and large, teams consisted of members who were individually virtual but also met face-to-face occasionally. The report writing teams were the most virtual with some team members never having seen each other.

It can also be argued that student teams do not give representative answers that parallel those of individuals working in companies although there is evidence that this is not always so. Hughes and Gibson (1991) found that MBA students made decisions comparable to managers in an Executive MBA program, but Ashton and Kramer (1986) in their literature review, note that attitude questions are answered differently by individuals in the workforce than by students. Briggs, Balthazard, and Dennis (1995) found students to be valid predictors of managerial technology adoption and Remus (1989) found graduate students to be more representative of industry personnel than undergraduates. In particular, studies show that students are not representative because of their lack of experience in the workforce and because of their youth. Because most of the students in our study were part time students and had full-time jobs, because the age of the students in the teams was higher than normal for university undergraduates and because the students represented the cultural mix that we wanted to assess, it was felt that this study's population was representative of the industry group our study targeted.

We did find, however, evidence that suggests that students were responding differently than a workforce population might. Student groups uniformly evaluated their team competence and team performance highly with more spread on this evaluation in the report-writing teams (Master's students). We also found that we were not able to obtain any viable reliability on what is considered a highly reliable trust measure that we borrowed from the literature. We therefore did not include trust in our models. We also remade the competence measure into a formative construct for the survey of the report writing team. This helped to fix the skew in the distribution of these results.

As mentioned earlier, the skewed evaluations of team competence and performance may be an artifact of student teams, but they also might be an artifact of virtual teams in some cultures. Thus, an additional variable to collect and compare to self report of team performance is a team leader's report of team performance in addition to other related variables such as subproject completion times. We continue to strive to obtain measures of performance from team leaders, but, to date, the descriptions of how management scores performance for their distributed teams lead us to believe that self-report of perceived performance is not any worse a measure than managerial reports. In particular, the cultural and temporal distances are likely to affect a leader's perception of performance.

The trust measure may have been highly unreliable because student team members have a different social relationship than workforce team members. One trust question asked if a team member would like to control the work of the other team members. This is socially out of the question

in a student team where fellow team members are independent individuals more than members of some greater whole such as a company. Thus, team members are likely to respond negatively to this question. A second question asked if a team member felt uncomfortable with the work of the other team members. A team member could answer positively in a socially acceptable way to this question. Thus, the reliability of a construct that works fine in a company environment falls apart in the student context. However, this failure of the trust construct could be applicable to virtual teams where their non-colocated counterpart may also feel that they have no right to control or monitor the work that the distant team performs even if they feel uncomfortable with this lack of control. This is likely to be true of teams in China and India where new hires are continually being added to the workforce. None of these younger team members would feel that they should control the work of their team members in Europe or North America, but their recent training may also make them feel uncomfortable with the work being done in these places.

These discussion notes that the literature demonstrates that student teams can provide reliable answers that represent industry situations if the groups are appropriately chosen, but it also suggests that the very nature of student teams might be more appropriate for studying virtual teams across cultures in that their responses might represent similar cultural responses.

The small sample size also limits the generalizability of the study findings. We analyzed the teams separately because of inherent differences in their virtuality and work assignments. We also performed a separate analyses so that we could examine the effects of virtuality although confounded with task and a more senior student population. This made the sample sizes small. Approximately 200 software teams were asked to fill in the survey but only 30 responded. The response rate was extremely low, in particular, because many of the students were in their final year, already had jobs that reduced the importance of the payment incentive we offered and were quite busy with class projects. The response rate was significantly higher for the report writing teams (about 50 percent) but the class size was small. We also choose to analyze all of the teams, even those with only one respondent because the response rate was low. Thus, there were 12 software teams that only had one member. For the report writing teams, only teams with two or more respondents were used in the analysis. The problem with a single team member responding constitutes another analysis problem because that single member could have been an outlier generating data unrepresentative of the team. Studies now in progress with a larger number of industry teams will yield more conclusive findings related to virtual team leader delegation.

CONTRIBUTIONS AND FUTURE WORK

This study addressed an important yet under-researched area regarding leader delegation practice in virtual software teams. It provides statistically sound conclusions which reduce the confusion arising from the conflicting findings of prior case studies. Some virtual software team managers claim that the increased distance make self-management necessary while others question whether excessive autonomy will produce negative effects in a complex virtual team environment. A pressing question for industry practitioners is when and how much authority and responsibility should be delegated to remote team members. Our findings suggest that delegation is an important virtual team management strategy that positively affects team performance. In particular, the results from the student teams imply that a team leader will delegate in response to his or her sense of how competent the team is. However, the real world situation affects this delegation with a push and pull effect, that is, management will want to

delegate more because the task of managing a global virtual team means more communication, odd hours of work scheduled for communication and care needed to avoid miscommunications. This is the push to delegate. However, the pull effect is that a team leader because of the lack of information on the global virtual team stemming from language difficulties, cultural differences and simply not being able to observe team behavior because of the distance will not want to delegate to the team. The findings suggest that team leaders need to be trained to ignore these effects and perform their delegation based on real information about the team, perhaps by visiting the virtual team or setting up viable measures for team performance.

The tendency of the student teams to give self-reports of high competence and high performance suggests leadership guidelines for industry. In particular, it would be wise to give team leaders training in the cultures they are interacting with so that they can better judge the self-reports they are obtaining.

Our findings also suggest that delegation is a good thing in that it increases a team's satisfaction with its leader, a team's flexibility to adjust the project to local needs and a team's motivation. The findings, however did not find a strong mediating relationship between these values and team performance Earlier discussion on the limitations of the study suggest that the performance measures captured were corrupted by the use of student teams. Flexibility, motivation, and satisfaction have been shown to affect performance in face-to-face teams so there is good reason to believe that obtaining better measures of performance would find moderation by these variables. This is future work that needs to be performed.

Overall, more delegation was found to be a positive behavior for a team leader, but with the youth and newness of team members joining virtual teams in many of the companies that offshore or outsource, this is likely to be a poor strategy unless measures are taken to bring up the skill sets of the offshore team members.

Due to high team member turnover in some countries, especially with the constantly increasing wages in these countries, companies are reluctant to invest in training for these team members. However, as we have been told by team managers from India, this training is precisely one of the mechanisms used to reduce turnover. This is another variable that clearly needs evaluating in future work.

Overall, the pilot study findings suggest useful recommendations for virtual software team leadership as to when and how much they should delegate to the team based on the team's degree of virtuality and competence. However, the hypotheses of this study were mainly based on software team literature and the pilot study was done mainly with student software teams. Future work with real industry teams with varied types of companies and a variety of organizational models needs to be performed to verify these findings. Currently a full-scale survey and interview has been conducted with the global software testing teams of a Fortune 100 company. Future data analysis will be performed to compare the industry team findings with the findings of this pilot study.

REFERENCES

Ashton, R. A., & Kramer, S. S., (1980). Students as surrogates in behavioral accounting research: Some evidence. *Journal of Accounting Research, 18*(1), 1-15.

Bass, B. M. (1990). *Bass and stogdill's handbook of leadership* (3rd ed.). New York: The Free Press.

Bell, B.S., & Kozlowski, S.W.J. (2002). A typology of virtual teams: Implications for effective leadership. *Group and Organization Management, 27*(1), 14-49.

Briggs, R. O., Balthazard, P. A., & Dennis, A. (1995). Graduate business students as surrogates for executives in the evaluation of technology. *Journal of End User Computing, 8*(4), 11-17.

Campion, M. A., Medsker, G. J., & Higgs, A. C. (1993). Relations between work group characteristics and effectiveness: Implications for designing effective work groups. *Personnel Psychology, 46,* 823–850.

Coffin, T. E. (1944). A three-component theory of leadership. *Journal of Abnormal and Social Psychology, 39*(2), 63-83.

Curtis, B., Krasner, H., & Iscoe, N. A., (1988). Field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Davis, R. C. (1942). *The fundamentals of top management*. New York: Harper.

Eveland, J.D., & Bikson, T.K. (1988). Work group structures and computer support: A field experiment. *Transactions on Office Information Systems, 6*(4), 354-379.

Faraj, S,. & Sambamurthy, V. (2006). Leadership of information systems development projects. *IEEE Transactions on Engineering Management, 53*(2), 238-240

Goodhue, D., Lewis, W., & Thompson, R. (2006). PLS, small sample size, and statistical power in MIS research. *Proceedings of the 39th Hawaii International Conference on System Science*, (pp. 1-10).

Hardin, A., Fuller, M., & Valacich, J. (2006). Measuring group efficacy in virtual teams: New questions in an old debate. *Small Group Research, 37,* 65-85.

Henderson, J. C., & Lee. S. (1996). Managing IS design teams: A control theories perspective. *Management Science, 6,* 757-777.

Hersey, P., & Blanchard, K. (1988). *Management of organizational behavior: Utilizing human resources* (5th ed.). Englewood Cliffs, NJ: Prentice-Hall.

Hertel, G., Geister, S., & Konradt, U. (2005). Managing virtual teams: A review of current empirical research. *Human Resource Management Review, 15(1),* 69-95.

Herzberg, F. I. (1968). One More Time: How do You Motivate Employees? *Harvard Business Review.* January-February, 109-120.

This work was previously published in the International Journal of e-Collaboration, edited by N. Kock, Volume 5, Issue 1, pp. 47-68, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 6.10
Cultural Diversity Challenges: Issues for Managing Globally Distributed Knowledge Workers in Software Development

Haiyan Huang
The Pennsylvania State University, USA

Eileen M. Trauth
The Pennsylvania State University, USA

ABSTRACT

This chapter discusses cultural diversity challenges in globally distributed software development and the implications for educating and managing the future global information technology workforce. It argues that the work practices of global software development are facing a variety of challenges associated with cultural diversity, which are manifested in and can be analyzed from three dimensions: the work environment of global software development, the globally distributed knowledge workers, and the global software development work. It further articulates how cultural diversity is manifested in these three dimensions. Furthermore, it highlights the importance of developing cultural awareness and cultural diversity understanding as important skills for the future information technology workforce.

INTRODUCTION

In this chapter, we explore the cultural diversity challenges of managing globally distributed knowledge workers who engage in global software development work practices. This topic is important to information technology personnel management and knowledge management for three reasons. First, there has been a significant increase in global software development work practices in recent years. Such work practices not only adopt the conventional characteristics of knowledge intensive work, but also generate a set

of distinct features, which call special attention to managerial researchers and practitioners. Second, in global software development, the information technology (IT) professionals are globally distributed in the forms of global virtual teams and represent a wide range of nationalities and, thus cultures. Therefore, we should not only acknowledge the existence of cultural diversity of globally distributed knowledge workers, but also explore how such cultural diversity may affect global software development work, and how to explore, assess, and manage this cultural diversity. Third, although cross-cultural issues have been one of the major concerns of the global information systems discipline, there are still on-going debates about how to assess culture and cultural diversity. As a result, different views of culture and cultural diversity will have impacts on the related human resource strategies used in managing global IT personnel. Consequently, evaluation and reflection on those issues in global software development work environments are very important.

As knowledge work is increasingly outsourced globally, we would like to take the opportunity in this book chapter to consider the cultural diversity challenges of managing globally distributed knowledge workers. The objectives of this book chapter are: (1) to propose a framework to address the cross-cultural aspects of managing IT personnel in globally distributed software development work; and (2) to discuss some managerial implications that are derived from this framework. We believe both professionals and academics working in the field of global information technology and information systems (IS) management will benefit from these discussions.

The organization of the book chapter proceeds as follows. In the Background section, we introduce the concepts of global software development and virtual teamwork. Then we present our research framework, which focuses on articulating how cultural diversity is manifested in global software development workplaces, workers, and work practices. In the following section on recommendations, we discuss how we may address the cultural diversity challenges in managing globally distributed knowledge workers who are engaged in global software development activities, particularly from the perspectives of IS/IT education and organizational human resource management.

BACKGROUND

Global Software Development

Global software development as one type of information technology offshore outsourcing activities (Lacity & Willocks, 2001), has become an established practice for software and information systems development (Carmel & Agarwal, 2002; Herbsleb & Moitra, 2001). Global software development can be defined as software and information systems development practices that are knowledge intensive and involve the work arrangements between two or more organizations across the national boundaries.

Software and information systems development has been widely conceived as knowledge-intensive work (Henninger, 1997; Swart & Kinnie, 2003) with three characteristics. First, knowledge as intellectual capital is an important input to a software development project, and an important output as well (Swart & Kinnie, 2003; van Solingena, Berghoutb, Kustersc, & Trienekensc, 2000). Second, Waterson, Clegg, and Axtell (1997) pointed out that software development work is "knowledge intensive" in the sense that building a complex software system demands selecting and coordinating multiple sources of knowledge (Shukla & Sethi, 2004). Drucker (2004) argued that the specialized knowledge in knowledge work indicates that knowledge workers need to access the organization—the collective that brings together a diversity of specialized knowledge workers to achieve a common goal. For example, a software development project may

involve a variety of IT personnel such as designer, analyst, programmer, tester, implementer, and manager. Therefore, collaborations of team work are necessary and critical for software development projects. Third, knowledge associated with software development is rapidly changing as the complexity and diversity of the application domain is increasing (Henninger, 1997). Therefore, software development knowledge is not static but, rather, is evolving with the changing needs of the customers and business environments (Henninger, 1997). Drucker (2004) pointed out that knowledge workers not only need formal education to enable them to engage in knowledge work in the first place, but also need continuous learning opportunities through the work practice to keep the knowledge up-to-date. These three characteristics of software development work usually refer to the work practices within a single organizational domain. As software and information systems development work is increasingly outsourced globally, how to manage the knowledge workers to facilitate effective software development work practice in the cross-cultural context has become a great challenge.

Since the 1990s, software development and IT services have become dominant in global sourcing, which includes application packages, contract programming, and system integration (Lee, Huynh, Kwok, & Pi, 2002). And the global IT outsourcing market is continuously growing (Sahay, Nicholson, & Krishna, 2003; Trauth, Huang, Morgan, Quesenberry, & Yeo, 2006). It was projected that the IT outsourcing revenue would reach $159.6 billion by 2005 (Laplante, Costello, Singh, Bindiganaville, & Landon, 2004). The U.S. is the primary user of the global software and systems development market, followed by Western European countries such as the UK and Germany (Sahay et al., 2003). Countries such as India, Ireland, and Israel, have dominated the offshore outsourcing supplier market (Gopal, Mukhopadhyay, & Krishnan, 2002). A news release (*InformationWeek*, June 3, 2004) indicated that India's revenues from exports of software and back-office services is at $12.5 billion in the latest fiscal year and with growth of 30% compared with $9.6 billion in the previous year. Another news release (Friedman, 2005) reported that 7 out of 10 top software designers have operations in Ireland.

When compared to the traditional characteristics of software development work, globally distributed software development knowledge work has three additional characteristics. First, it is mainly conducted through a virtual environment that is supported to a great extent by networking technologies. Such virtual space is global by nature and transcends national and organizational boundaries. Second, it is situated within different complex, multi-leveled socio-cultural contexts. Walsham (2000, 2001) argued that the distinct cultures of different local contexts are critical factors in mediating the globalization process in the specific contexts. Therefore, the globally distributed workplace has a global-local duality. Third, the work practices of global software development are facing a variety of challenges associated with the difficulties of temporal and spatial distance, and cultural diversity.

Global Virtual Team

The globally distributed virtual team is the basic unit engaged in software development work. A global virtual team can be defined as a collection of individuals who are organizationally and globally dispersed, and culturally diverse, and who communicate and coordinate work activity asynchronously or in real time primarily through information and communication technologies (ICTs) (DeSanctis & Poole, 1997; Jarvenpaa & Leidner, 1999).

A variety of strategic and catalytic factors have contributed to the increasing trend of using globally distributed virtual teams for software and information systems development (Carmel, 1999; Herbsleb & Moitra, 2001). These include:

24/7 around-the-clock development activities, the desire to reduce development costs and have access to a global resource pool, and the proximity to the customer. In addition, some authors have further emphasized the contribution of diversity of heterogeneous teams to work performance brought about by globally dispersed team members (Adair, 1986; Harrison, McKinnon, Wu, & Chow, 2000; Hartenian, 2000; Maugain, 2003; Trauth et al., 2006). For example, Maugain (2003) argued that the different thinking modes and dissimilar problem solving methods brought in by diverse team members in multicultural R&D (Research & Design) teams will stimulate novel ideas and creativity. Hartenian (2000) pointed out that diverse groups have a tendency to make higher quality decisions, to be more creatively motivated, and have a higher productivity potential than less diverse groups.

However, research also shows that the absence of regular face-to-face interactions and the breakdown of traditional communication and coordination mechanisms are negatively associated with the effectiveness of globally distributed software development teams (Cameral, 1999; Herbsleb & Mockus, 2003). Systems development tasks, particularly front-end activities, require formal and informal communication and coordination (Audy, Evaristo, & Watson-Manheim, 2004) to facilitate knowledge exchange and learning (Curtis, Krasner, & Iscoe, 1988). According to Herbsleb and Mockus (2003), the change of communication patterns and the lack of effective communication channels (formal or informal) in globally distributed software development teams can lead to delays in global software development projects. The study by Cramton and Webber (2005) shows a negative relationship between geographic dispersion and perceived team performance with respect to complex and interdependent tasks.

The cultural difference may further exacerbate the communication problems (Herbsleb & Moitra, 2001). Carmel (1999) pointed out that the barriers of time, space, and cultural distances may be detrimental to building trust and achieving team cohesiveness in global virtual teams. Nicholson and Sahay (2004) argued that the barriers of knowledge sharing among knowledge workers in offshore software development are related to the embeddedness of knowledge in the local cultural context, and should be investigated at the interconnected societal, organizational, and individual levels of analysis.

While cultural factors may influence global virtual teams engaged in a variety of activities in general, they are particularly important to software development work for three reasons. First, compared to other activities such as new product developments in manufacturing sectors, the processes of software development are more complexly interdependent and iterative, the products of software development are less tangible, and knowledge perspectives involved in software development are more tacit and fast changing in nature (Sahay et al., 2003). Second, a number of studies have shown that culture is a critical influential factor in global software development work and has impacts on a variety of issues. While some issues are general issues faced by global virtual teams engaged in other activities in general (e.g., managing conflicts—Damian & Zowghi, 2003), building trust (Zolin, Hinds, Fruchter, & Levitt, 2004), some issues are specific to software development, such as managing IT outsourcing relationships (Krishna, Sahay, & Walsham, 2004; Nicholson & Sahay, 2001; Sahay et al., 2003), preference of software development methods (Borchers, 2003; Hanisch, Thanasankit, & Corbitt, 2001), preference of computer supported collaborative technologies (Massey, Hung, Montoya-Weiss, & Ramesh, 2001), knowledge transfer and management related to software development (Baba, Gluesing, Rantner, & Wagner, 2004; Nicholson & Sahay, 2004; Sarker, 2003), and the process and performance of globally distributed software development teams (Carmel, 1999; Olson & Olson, 2003). Third, as more and more countries are now entering the IT outsourc-

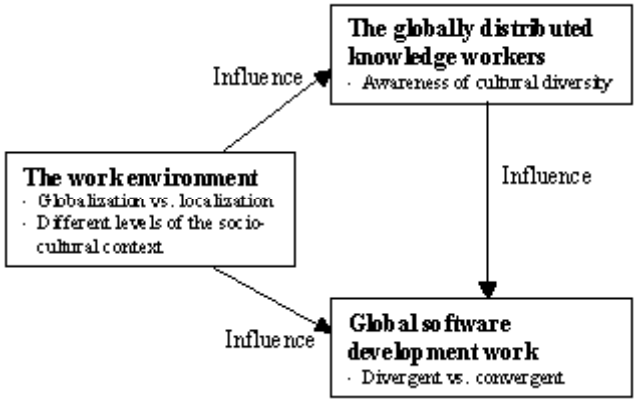
ing market, global software development work practices are facing more cultural diversity (Sahay et al., 2003; Trauth et al., 2006). Companies in Japan and Korea join those of the U.S., Canada and other western European nations in outsourcing their software or information system development and services activities to other countries. Besides the current leading outsourced countries such as India, Ireland, and Israel, Russia and China are now establishing their capabilities as outsourcing providers (Sahay et al., 2003).

Globally distributed software development efforts, thus, must deal with trade-offs between taking advantage of the global resource pool and cultural diversity while managing the cultural and distance barriers to effective communication and coordination in a geographically dispersed environment. How to make sense of cultural diversity and its impact on managing globally distributed knowledge workers who are engaged in global software development work activities are becoming the primary concerns of global IT personnel management and knowledge management.

In the research framework (Figure 1), we propose that cultural diversity is situated and manifested in three interrelated dimensions of global software development activities: the virtual workplace, the workers, and the work (Trauth, 2000). These three main constructs reflect what dimensions of global software development may be affected by cultural diversity. And the bullets under each main construct further indicate how cultural diversity is manifested in each of these dimensions. Trauth (2000) studied the information economy development in Ireland and pointed out that culture is one of the major influential factors. More specifically, she addressed the cultural influences from three perspectives: multinational workplaces, knowledge workers, and knowledge work. These three perspectives are interrelated and serve as our analytical lenses to study how the cultural factors influence IT work, and in this chapter, how cultural diversity is manifested in and affects global software development.

The virtual workplace of global software development is situated within a multi-leveled socio-cultural context with the global-local duality characteristic, which constitutes a unique work environment. Being engaged in global software development activities within such a work environment, globally distributed knowledge workers should be aware of the complexity and dynamics

Figure 1. Research framework: Situating cultural diversity in global software development



of cultural diversity, and constantly make sense of and negotiate meanings of such diversity. Global software development work, which includes both divergent and convergent perspectives is affected by the effectiveness of the sense-making processes and the management of cultural diversity. This framework adopts a situated approach and emphasizes the importance of studying globally distributed knowledge work as socially negotiated work practices by situating it within the both the global and local contexts (Avgerou, 2002; Trauth, 2000; Weisinger & Trauth, 2002, 2003). In the following sections, we discuss in detail how cultural diversity is manifested in each dimension of the framework, respectively.

THE WORK ENVIRONMENT

Globalization vs. Localization

Globally distributed software development work can be seen as a result of the globalization process—the IT industry is becoming more and more globally interconnected. According to Castells (1996), the globalization process involves the flows of capital, commodities, technology, cultural influences, and human resources across national boundaries, thereby creating a networked society. One stream of sociological and cultural research considers processes of globalization and flows of cultural elements across frontiers as a global "cultural homogenization" (Kellner, 2002; Schuerkens, 2003). Schuerkens (2003) criticized such "cultural homogenization" arguments of globalization by pointing out that they usually ignore the existence and active role of local cultural perspectives. Castells (1996) also pointed out that the globalization process is selective and segmented with many imbalances, and the networked society is both centralized and decentralized, which shows heterogeneous and global-local duality characteristics.

Sahay et al. (2003) argued against the "cultural homogenization" assumption of globalization and proposed that global software development work relationships can be seen as "models of" globalization process and "models for" globalization as well (p. 27). They emphasized the dynamic reciprocal relationships between the local cultural contexts and the globalization processes. Similarly, Walsham (2001) used Giddens' structuration theory (1990) and Castells' network society theory (1996) to study global IT development and stressed that the existing socio-cultural context of a country is a critical factor in mediating the globalization process in the specific context and, in turn, will have an impact on the complexity of globalization. They both acknowledge the uniqueness and importance of local contexts to globally distributed software development.

To illustrate the continuous interactions of local cultural elements and global cultural influences, Schuerkens (2003) cited Long's (1996) discussion: "Local situations are transformed by becoming part of wider global arenas and processes, while global dimensions are made meaningful in relation to specific local conditions and through the understandings and strategies of local actors" (p. 217).

Therefore, the local cultural context is neither a passive recipient of globalization and external cultural influences as indicated by the "global homogenization" argument, nor is it a static and deterministic factor that remains unchanged during the globalization process. The local cultural forms and meanings are constantly reconstructed (Schuerkens, 2003; Walsham, 2001).

We believe that such continuous interactions of globalization and localization processes have three implications for conceptualizing the cultural diversity of global software development work environments. First, global software development work is situated within a complex and dynamic global-local societal context. Second, cultural diversity is inherent in global software develop-

ment and is a critical influential factor affecting global software development work practices. Third, the emergent nature of both the local cultural context and the globalization process indicates that we should focus on the appropriation and transformation of local cultural elements to address the dynamic perspectives of cultural diversity of the global software development work environment.

Different Levels of the Socio-Cultural Context

Another important feature of the cultural diversity of the virtual workplace of global software development is the multi-levels of analysis ranging from societal (national) to regional, organizational, professional, and team. Different cultural factors at different levels coexist, interact with each other, and together produce different work environments of globally distributed software development work practices. However, the influence of different cultural factors at different levels is not equal and varies across work environments. Some cultural factors may be more visible than others and some may seem trivial compared to the predominant factors depending on different cases.

For example, Robey, Gupta, and Rodriguez-Diaz (1988) studied one multinational company's efforts to implement an account system in its subsidiaries in two Latin American countries: Chile and Panama. Their findings showed that cultural and political differences between Chile and Panama could not explain the significant differences in the implementation outcomes. They believed that those differences were due to the organizational cultural differences of the two subsidiaries. This case is an example of the dominance of organizational cultural influences while national and organizational cultural differences coexist. Barrett and Walsham (1995) studied the global software development relationships between a Jamaican insurance company and an Indian software company. They pointed out that although the Indian and Jamaican team members of this joint venture development shared a similar professional culture, there were major differences between the local work culture at the Indian software company and the Jamaican insurance company. This case demonstrates the dominance of national and organizational cultural influences. The case study by Kaiser and Hawk (2004) on a long-term alliance outsourcing relationship between a U.S. company and an Indian company showed that the mutual understanding of ethnic and corporate cultures was an important factor to build stable and trust relationships.

In global IT outsourcing research, the focus tends to be on the national level of analysis. Therefore in most cultural studies of global software development, the national culture is predominant while other factors such as regional, organizational, and team cultures are in the background. This is probably due to the high visibility of cross-national cultural differences. Another reason may be that the cultural elements at different levels interact with each other and somehow diffuse into some inseparable influential factors.

Some studies (Cougar, Adelsberger, Borovits, Zviran, & Motiwalla, 1990; Constantine, 1995) pointed out that while the national culture may show divergent characteristics across national boundaries, the professionalism of the knowledge workers will share certain common cultural elements that constitute the professional culture. As a result, the team culture of the global virtual team may show a different pattern from either the national culture or the organizational or the professional culture. Earley and Gibson (2002) pointed out that through communications and interactions, the highly heterogeneous global team may appear to develop a common identity over the course of a long-term project, which they referred to as the team culture.

The contemporary work environment of global software development is situated within

a complex multi-leveled socio-cultural context in which culture and its influences are emergent as the work practices evolve. The emergent perspective of cultural diversity indicates that it cannot be reduced to a set of variables and treated as unchanging inherited properties. In a sense, globally distributed knowledge workers are not passively embedded in their local context. Instead, they continuously and actively engage and negotiate with their work environment in everyday work practices.

The Globally Distributed Knowledge Workers

Brannen, Gómez, Peterson, Romani, Sagiv, and Wu (2004) pointed out that the concept of culture is by no means free of controversy. According to Worsley (1984), there are four ways of conceptualizing culture: the elitist view – culture implies superior power; the holistic view–culture implies the whole way of life; the hegemonic view – culture is a set of behaviors imposed by the majority; the relativist view – culture is localized and may bear different behaviors in different regions or communities from the same society.

There are two general doctrines of conceptualizing culture—the functionalist view and the interpretivist view (Schultz & Hatch, 1996). The functionalist view assumes that culture can be studied from several generalized dimensions and those dimensions are universal. As a result, the functionalist studies focus on categorizing cultural dimensions and predicting their influences. Hofstede's (1984) framework of five major national cultural dimensions is one example of the functionalist doctrine. The interpretivist doctrine, on the other hand, argues that culture may be ambiguous and unstable and should be studied within a specific local context instead of using general frames.

Schultz and Hatch (1996) studied the difference between the functionalist and interpretivist paradigms of cultural research. They proposed that these two paradigms can somehow interact to address the different perspectives of culture. To some extent, the mainstream of each of these doctrines can be integrated to a "multiparadigm" approach. They suggested that for example, to study national cultural patterns, the functionalist view uses predefined categories to provide a clear, generalizable and stable pattern, while the interpretivist view uses interpretation and symbolic representation to describe the ambiguous, situated and instable perspectives of culture. However, this approach may still be problematic since it assumes that there is a line between the stable and unstable elements of culture. In reality, the line itself may be ambiguous and dynamically changing.

The anthropological view of culture is a constructivist view which rejects the idea of culture as having hard and fast boundaries (Avison & Myers, 1995). On the contrary, culture is seen as contestable, temporal, and emergent, and is constantly interpreted and re-interpreted in social relations (Carrithers, 1992). Therefore, the anthropological cultural view rejects the notion of culture as a set of predefined variables peculiar to a certain society. In the information systems discipline, Walsham's notion of culture mediating the global process in specific local contexts (Walsham, 1993, 2000, 2001), Avegerou's proposal on relativism (2002), and the situated culture perspective suggested by Weisinger and Trauth (2002, 2003) are three approaches to studying culture through exploration, interpretation, and sense making, which reflect the anthropological perspective of conceptualizing culture.

We argue that when managing global software development practice, the functionalist approach may provide general guidance if cautiously adopted. However, it is lacking the capability to provide an in-depth understanding of cultural dynamics. Therefore, we take the following statement as a working definition of culture:

Culture is the sense making of different social structures and relations such as beliefs, values, and norms, attitudes, hierarchies by a group of people within a particular social context.

In this definition, we view culture as the "sense making" which actively strives for interpretation and re-interpretation of the relationships between the self-identity and the surrounding contexts. We believe that viewing culture as dynamic and emergent instead of static and predefined will provide the corresponding cross-cultural management the capability of accounting for the evolving and diversified nature of global software development phenomena.

Child (2002a, b) pointed out that the globalization trend and subsequent interconnectivity of networking technologies have pushed the traditional boundaries between nations and organizations to become somehow "borderless." At the same time, they enhance the people's awareness of their own identity and cultural distinctiveness as they have more and more opportunities to interact with a variety of cultural groups during the processes. In a sense, they interpret and reinterpret self-identity and the relationships between the self-identity and the surrounding contexts.

We argue that cultural awareness of globally distributed knowledge workers should have two levels—the self-awareness of their own identity and the mutual awareness of the existence of the cultural diversity and differences in others. Baba et al. (2004) pointed out that in order for team members of a globally distributed team to bring together and integrate the divergent knowledge, they should first develop the mutual awareness and shared cognition of the divergences. They further stressed (Baba et al., 2004) that the mutual awareness is not simply exchanging declarative or procedural knowledge—it requires: "…suspending our own judgment as we learn the cultural logic and rationality of others' divergent beliefs and values, while also allowing those others to call our own beliefs and values into question as they learn about us…" (p. 583).

The Global Software Development Work

Studies have shown that while cultural diversity may lead to advantages with respect to the divergent processes of knowledge work, it may also cause problems for the convergent processes (Miroshnik, 2002). Divergent knowledge work processes in software development refer to processes of generating and articulating different viewpoints by different team members, as well as challenging the existing assumptions in requirement analyses and systems designs, which are important for surfacing and exploring alternatives, thus promoting creativity and innovation in software development (Kryssanov, Tamaki, & Kitamura, 2001; Nickerson, 1999). Convergent knowledge work processes refer to processes of developing shared understanding and building common ground among team members with respect to different perspectives of software development, which are important to decision-making and effectiveness of teamwork (Potts & Catledge, 1996).

Knowledge intensive work, such as design and development of new software and information systems is usually characterized as highly ambiguous, uncertain, equivocal, and interdependent (Curtis et al., 1988; Herbsleb & Grinter, 1999; Hoegl & Proserpio, 2004). The analysis of systems requirements, which is a critical task at the front-end of software and information systems development, is highly dynamic, complex, fluctuating, and evolutionary in nature (Audy et al., 2004; Curtis et al., 1988; Mathiassen & Stage, 1990). Cultural diversity may provide benefit to the front-end of software development work by providing different perspectives, ideas, and approaches. Dafoulas and Macaulay (2001) pointed out that cultural diversity may be beneficial to

team performance, especially on tasks for which differing perspectives might increase team performance (Trauth et al., 2006). Miroshnik (2002) also argued that cultural diversity can be used as a resource to enhance creativity, flexibility and problem solving skills, all of which are important for knowledge-intensive work.

On the other hand, to bring the divergent perspectives into a convergent development practice, cultural diversity may become a barrier to knowledge sharing and transference since knowledge is contextually dependent and culturally contingent (Nicholson & Sahay, 2004). To a great extent, the convergent processes require both formal and informal communication and coordination mechanisms to exchange diverse knowledge perspectives and facilitate learning (Curtis et al., 1988), to surface conflicts and negotiate differences (Audy et al., 2004; Briggs & Gruenbacher, 2002; Curtis et al., 1988), and to build shared understandings and common ground regarding various issues such as how to represent the system requirements and which system development methodologies are more appropriate (Cramton & Webber, 2005; Damian & Zowghi, 2003). During these convergent processes, cultural diversity may create cultural distance and barriers to knowledge sharing and transference. Herbsleb and Moitra (2001) pointed out that while cultural diversity can be seen as an enriching factor by bringing together divergent bodies of knowledge, it can also lead to serious and chronic misunderstandings.

For example, in the case study of distributed software development between England and India, Nicholson and Sahay (2004) identified cultural difference in perceptions of time between India and England. In England, a 9 AM to 5 PM working routine and the separation of working life from personal life are encouraged. In India, the boundaries between working life and home life are less defined (Nicholson & Sahay, 2004). Thus, Indian employees may respond to personal or home needs during regular working hours and may spend extra time working later hours or on weekends (Nicholson & Sahay, 2004). Such cultural differences are implicitly embedded in each local cultural context. Without building corresponding mutual knowledge and awareness about these differences, team members from one site may have misconceptions about the availability of team members from the other site. Saunders, van Slyke, and Vogel (2004) argued that different global virtual team members may have different time visions, which may influence the management and performances of global virtual teams.

The interchange of benefits of cultural diversity and hindrance of cultural distance put forward special challenges of teaching cultural diversity to future IS/IT workforce and managing globally distributed knowledge workers. On one hand, we need to address issues related to bridging cultural distance to encourage knowledge sharing and transference across different cultures. On the other hand, we need to study how to cultivate and integrate cultural diversity in order to develop new organizational capabilities (Baba et al., 2004).

RECOMMENDATIONS FOR PRACTICE

As suggested by our research framework, it is very challenging to manage such a diverse workforce for global software development for three reasons. First, the culture of the virtual work environment is complex and dynamic. Second, the cultural diversity of globally distributed knowledge workers has the potential for both great accomplishments and great conflicts. Third, the cultural diversity of such a global workforce needs to be proactively managed, and cultivated in order to facilitate both the divergent and convergent perspectives of software development work activities.

To address these challenges, we recommend the following. First, treat cultural awareness and cultural diversity understanding as important and necessary skills for the future IS/IT workforce, provide IS/IT students opportunities to experi-

ence cultural diversity, and help them explore and develop a proper mind-set towards diversity. Second, adopt the sense-making approach in human resource practices to motivate and facilitate globally distributed knowledge workers' articulation of their self-identities and the identities of others during the social interactions of teamwork processes. Third, balance the tensions between the values and the conflicts of cultural diversity by encouraging contested, diverse thinking while building the trust and shared understanding among globally distributed team members. Finally, value cultural diversity knowledge as an important part of the organization's intellectual capital and strategic resources for competing in the global market.

IS/IT Education

The gap of critical skills and knowledge required for information technology professionals between academe and industry has been a major concern for IS/IT education (Lee, Trauth, & Farwell, 1995; Miller & Donna, 2002; Swanson, Phillips, & Head, 2003; Trauth, Farwell, & Lee, 1993). Academics and practitioners have called for assessing and expanding IT, IS, and MIS curricula to adapt to the needs of future global IS/IT workforces (Miller & Donna, 2002; Swanson et al., 2003). For example, Swanson, et al. (2003), and Noll and Wilkins (2004) discussed the growing needs for soft skills such as communication skills and teamwork skills in information technology professionals. Larsen and McInerney (2002) simulated the inter-organization virtual teamwork environment in course design to teach students certain skill sets needed in virtual work.

However, only a few of these programs specifically target the global IT environment and conceptualize diversity as one of the core elements in the global IT environment. One of those few examples is the online "IT Landscape in Nations" repository initiated by Carmel and Mann (2003) to facilitate students conducting comparative analyses of different nations and developing greater awareness of the global IT environment. Therefore, there is a great gap between current IS/IT education and the increasing demands of the global IS/IT workforce. Educators should focus on designing and implementing corresponding curricula, renovating and expanding current pedagogical approaches to bridge such a gap.

IS/IT Human Resource Management

Along with the focus shifting from capital resources to knowledge resources in modern economic development, the role of knowledge has been fundamentally changed (Drucker, 1994). As a result, the role of human resource management has become more and more important because "*people are the only sustainable asset in modern business*" (Schwarzkopf, Saunders, Jasperson, & Croes, 2004, p. 28). The strategies and implementations of human resource practices directly affect how knowledge workers will be continuously motivated and trained to perform their value creation tasks (Hill & Jones, 1998; Pfeffer, 1994). Trauth et al. (2006) pointed out that it is critical that researchers and practitioners take an active role in creating HR solutions and it is important to understand diversity issues in the global IT environment.

Kakabadse and Kakabadse (2000) pointed out that organizational outsourcing initiatives have both negative and positive effects on their employees. As more and more IT jobs shift offshore, it may hamper the employment relationship of belonging and dedication when employees feel unsafe with respect to job security (Kakabadse & Kakabadse, 2000). The cultural diversity and the lack of trust and cohesiveness of global virtual teams may influence team members' working experiences (Carmel, 1999). It is also argued that outsourcing and global software development arrangements may provide career enhancement and learning opportunities for employees and organizations provided that special expertise

and skills can be acquired and knowledge can be mutually shared and transferred across borders (Baba et al., 2004; Carmel, 1999; Kakabadse & Kakabadse, 2000). Therefore, one of the primary concerns of human resource management in global software development practices is how to mitigate the negative impacts and enhance the positive effects.

Given the complexity and dynamics of cultural diversity and its criticality in global software development work practices, it is important to emphasize the sense-making perspective in cultural training and provide employees proper and continuous cross-cultural training. When knowledge workers are involved in different virtual work environments, the stereotypically and culturally specific approach may fail to help them make sense of different cultural nuances from different cultural contexts (Goodall, 2002; Osland & Bird, 2004). Therefore, cross-cultural training should focus on how to develop and improve the cultural sense-making skills of employees.

Foster (2000) studied the cultural training for expatriates of multi-national companies and pointed out that most of those training programs focus on pre-departure training and fail to provide continuous training during the work processes. Krishna et al. (2004) pointed out that systematic cross-cultural training is less common than informal experience sharing in their case studies of global software development activities. And if in place, that cultural training is usually in one direction: for the outsourced companies to learn the culture of the outsourcing companies (Krishna et al., 2004). Osland and Bird (2004) advocated the sense-making approach for cultural training and stressed that there should be both formal and informal mechanisms for sharing cultural knowledge.

We believe that organizational human resource management together with knowledge management practices should value cultural diversity knowledge as an important part of the organizational intellectual capital and strategic resources for competing in the global IT market in the future. Cross-cultural sense making, understanding, and knowledge sharing are critical to develop flexible, competitive, and yet sustainable learning organizations (Garvin, 1998). In cross-cultural training and learning practices, we should allow distributed knowledge workers to have opportunities to continuously reflect on their cultural experiences in the course of accomplishing working processes and encourage them to take such reflections as learning opportunities.

CONCLUSION

To address the cultural diversity challenges of managing globally distributed knowledge workers in global software development, we proposed a research framework to articulate how cultural diversity is manifested in global virtual work environments and how the cultural diversity of distributed knowledge workers may influence global software development work practices. The main objective of the chapter is to promote the awareness of cultural diversity challenges to managing information technology professionals in the increasingly globalized IT environment. Our analyses show that we should critically examine the global-local context of the cross-cultural issues to overcome the obstacles of cultural diversity in convergent tasks of software development work and maximize its values in divergent tasks of the work activities.

As researchers and educators in academic settings, we believe that cultural awareness and cultural diversity understanding should be viewed as important skills for the future IS/IT workforce. We also believe that organizational human resource practices should adopt the sense-making approach for cross-cultural training and knowledge sharing. In order for organizations to compete in the global market in the future,

cultural diversity knowledge should become an important part of the organization's intellectual capital and strategic resources.

REFERENCES

Adair, J. (1986). *Effective teambuilding: How to make a winning team*. London: Pan.

Avgerou, C. (2002). *Information systems and global diversity*. Oxford; New York: Oxford University Press.

Avison, D. E., & Myers, M. D. (1995). Information systems and anthropology: An anthropological perspective on IT and organizational culture. *Information Technology & People, 8*(3), 43-56.

Audy, J., Evaristo, R., & Watson-Manheim, M. B. (2004). Distributed analysis: The last frontier? *Proceedings of the 37th Hawaii International Conference on System Sciences*. IEEE.

Baba, M. L., Gluesing, J., Rantner, H., & Wagner, K. H. (2004). The contexts of knowing: Nature history of a globally distributed team. *Journal of Organizational Behavior, 25*(5), 547-587.

Barrett, M., & Walsham, G. (1995). Managing IT for business innovation: Issues of culture, learning, and leadership in a Jamaican insurance company. *Journal of Global Information Management, 3*(3), 25-33.

Borchers, G. (2003). The software engineering impacts of cultural factors on multi-cultural software development teams. *Proceedings of 25th International Conference on Software Engineering* (pp. 540-545).

Brannen, M. Y., Gómez, G., Peterson, M. F., Romani, L., Sagiv, L., & Wu, P. C. (2004). People in global organizations: Culture, personality, and social dynamics. In H. W. Lane, M. L., Maznevski, M. E., Mendenhall, & J. McNett (Eds.), *The Blackwell handbook of global management: A guide to managing complexity* (pp. 26-54). Malden, MA: Blackwell Publishing.

Briggs, R. O., & Gruenbacher, P. (2002). Easy winwin: Managing complexity in requirements negotiation with GSS. *Proceedings of 35th Annual Hawaii International Conference on Systems Science*. IEEE.

Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Upper Saddle River, NJ: Prentice Hall PTR.

Carmel, E., & Agarwal, R. (2002). The maturation of offshore sourcing of information technology work. *MIS Quarterly Executives, 1*(2), 65-77.

Carmel, E., & Mann, J. (2003). Teaching about information technology in nations: Building and using the "landscape of it" repository. *Journal of Information Technology Education, 2*, 91-105.

Carrithers, M. (1992). *Why human have cultures*. Oxford: Oxford University Press.

Castells, M. (1996). *The rise of the network society*. Oxford: Blackwell.

Child, J. (2002a). Theorizing about organization cross-nationally: Part 1 – An introduction. In M. Warner & P. Joynt (Eds.), *Managing across cultures: Issues and perspectives* (2nd ed., pp. 26-39). London: Thomson Learning.

Child, J. (2002b). Theorizing about organization cross-nationally: Part 2 – Towards a synthesis. In M., Warner & P. Joynt (Eds.), *Managing across cultures: Issues and perspectives* (2nd ed., pp. 40-56). London: Thomson Learning.

Constantine, L. (1995). *Constantine on Peopleware*. Englewood Cliffs, NJ: Yourdon Press.

Cougar, J. D., Adelsberger, H., Borovits, I., Zviran, M., & Motiwalla, J. (1990). Commonalities in motivating environments for programmer/analysts in Austria, Israel, Singapore, and the USA. *Information and Management, 18*(1), 41-46.

Cramton, C. D., & Webber, S. S. (2005). Relationships among geographic dispersion, team processes, and effectiveness in software development work teams. *Journal of Business Research, 58*(6), 758-765.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Dafoulas, G., & Macaulay, L. (2001). Investigating cultural differences in virtual software teams. *The Electronic Journal on Information Systems in Developing Countries, 7*(4), 1-14.

Damian, D. E., & Zowghi, D. (2003). An insight into the interplay between culture, conflict, and distance in globally distributed requirements negotiations. *Proceedings of the 36th Hawaii International Conference on System Sciences.* IEEE.

DeSanctis, G., & Poole, M. S. (1997). Transitions in teamwork in new organizational forms. In B. Markovsky (Ed.), *Advances in group processes* (Vol. 14, pp. 157-176). Greenwich, CT: JAI Press.

Drucker, P. F. (1994). The age of social transformation. *The Atlantic Monthly, 274*(5), 53-80.

Drucker, P. (2004). *The next workforce*. Retrieved on February 17, 2005, from http://207.36.242.12/data/html/pop/article3print.htm

Earley, P. C., & Gibson, C. B. (2002). *Multinational work teams: A new perspective*. Mahwah, NJ: Lawrence Erlbaum Associates Publishers.

Foster, N. (2000). Expatriates and the impact of cross-cultural training. *Human Resource Management Journal, 10*(3), 63-78.

Friedman, T. L. (2005, June 29). The end of the rainbow. *New York Times.* Retrieved January 26, 2006, from http://www.nytimes.com/2005/06/29friedman.html ?ex=127769700&en=a3f1a208e2617871&ei=5088&partner=rssnyt&emc=rrs

Garvin, D. A. (1998). Building a learning organization. In *Harvard Business Review on Knowledge Management* (pp. 47-80). Boston: Harvard Business School Publishing.

Giddens, A. (1990). *The consequences of modernity.* Cambridge: Polity Press.

Goodall, K. (2002). Managing to learn: From cross-cultural theory to management education practice. In M. Warner & P. Joynt (Eds.), *Managing across cultures: Issues and perspectives* (2nd ed., pp. 256-268). London: Thomson Learning.

Gopal, A., Mukhopadhyay, T., & Krishnan, M. S. (2002). The role of software process and communication in offshore software development. *Communications of the ACM, 45*(4), 193-200.

Hanisch, J., Thanasankit, T., & Corbitt, B. (2001, June 27-29). Understanding the cultural and social impacts on requirements engineering processes–Identifying some problems challenging virtual team integration with clients. *Proceedings of the 9th European Conference on Information Systems* (pp. 11-22). Bled, Slovenia.

Harrison, G., McKinnon, J., Wu, A., & Chow, C. (2000). Cultural influences on adaptation to fluid workgroups and teams. *Journal of International Business Studies, 31*(3), 489-505.

Hartenian, L. (2000, December). Cultural diversity in small business: Implications for firm performance. *Journal of Developmental Entrepreneurship*, 209-219.

Henninger, S. (1997). Case-based knowledge management tools for software development. *Automated Software Engineering, 4*(3), 319-340.

Herbsleb, J. D., & Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway's law revisited. *Proceedings of the 21st International Conference on Software Engineering* (pp. 85-95). Los Alamitos, CA.

Herbsleb, J., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering, 29*(6), 481-494.

Herbsleb, J. D., & Moitra, D. (2001). Global software development. *IEEE Software, 18*(2), 16-20.

Hill, C. W. L., & Jones, G. R. (1998). *Strategic management: An integrated approach* (4th ed.). New York: Houghton Mifflin.

Hoegl, M., & Proserpio, L. (2004). Team member proximity and teamwork in innovative projects. *Research Policy, 33*(8), 1153-1165.

Hofstede, G. (1984). *Culture's consequences: International differences in work-related values.* Beverly Hills, CA: Sage.

InformationWeek (2004, June 3). India's software exports reach $12.5 billion. Retrieved on December 10, 2005, from http://www.informationweek.com/story/showArticle.jhtml?articleID=21401198.

Jarvenpaa, S., & Leidner, D. (1999). Communication and trust in global virtual teams. *Organization Science, 10*(6), 791-815.

Kaiser, K. M., & Hawk, J. (2004). Evolution of offshore software development: From outsourcing to cosourcing. *MIS Quarterly Executive, 3*(2), 69-81.

Kakabadse, N., & Kakabadse, A. (2000). Critical review – Outsourcing: A paradigm shift. *Journal of Management Development, 19*(8), 670-728.

Kellner, D. (2002). Theorizing globalization. *Sociological Theory, 20*(3), 285-305.

Krishna, S., Sahay, S., & Walsham, G. (2004). Managing cross-cultural issues in global software development. *Communications of the ACM, 47*(4), 62-66.

Kryssanov, V. V., Tamaki, H., & Kitamura, S. (2001). Understanding design fundamentals: how synthesis and analysis drive creativity, resulting in emergence. *Artificial Intelligence in Engineering, 15*(4), 329-342.

Lacity, M., & Willcocks, L. (2001). *Global information technology outsourcing: Search for business advantage*. Chichester, UK: John Wiley & Sons.

Laplante, P. A., Costello, T., Singh, P., Bindiganaville, S., & Landon, M. (2004). The who, what, why, where, and when of IT outsourcing. *IT Professional, 6*(1), 19-23.

Larsen, K. R., & McInerney, C. R. (2002). Preparing to work in the virtual organization. *Information & Management, 29*, 445-456.

Lee, D. M., Trauth, E. M., & Farwell, D. (1995). Critical skills and knowledge requirements of IS professionals: A joint academic/industry investigation. *MIS Quarterly, 19*(3), 313-340.

Lee, J., Huynh, M., Kwok, R., & Pi, S. (2002). Current and future directions of IS outsourcing. In R. Hirschheim, A., Heinzl, & J. Dibbern (Eds.), *Information systems outsourcing: enduring themes, emergent patterns, and future directions* (pp. 195-220). Berlin, Germany: Springer-Verlag.

Massey, A. P., Hung, Y. T. C., Montoya-Weiss, M., & Ramesh, V. (2001). Cultural perceptions of task-technology fit. *Communications of the ACM, 44*(12), 83-84.

Mathiassen, L., & Stage, J. (1990). Complexity and uncertainty in software design. *Proceedings of the 1990 IEEE Conference on Computer Systems and Software Engineering* (pp. 482-489). Los Alamitos, CA: IEEE.

Maugain, O. (2003). *Managing multicultural R&D teams: An in-depth case study of a research project at CERN*. PhD thesis. Retrieved on January 26, 2006, from http://www.unisg.ch/www/edis.nsf/wwwDisplayIdentifier/2820/$FILE/dis2820.pdf

Miller, R. A., & Donna, D. W. (2002). Advancing the IS curricula: The identification of important communication skills needed by is staff during systems development. *Journal of Information Technology Education, 1*(3), 143-156.

Miroshnik, V. (2002). Culture and international, management: A review. *Journal of Management Development, 21*(7/8), 521-544.

Nicholson, B., & Sahay, S. (2004). Embedded knowledge and offshore software development. *Information and Organization, 14*(4), 329-365.

Nickerson, R. S. (1999). Enhancing creativity. In R. E. Sternberg (Ed.), *Handbook of creativity* (pp. 392-430). Cambridge: Cambridge University Press.

Noll, C. L., & Wilkins, M. (2004). Critical skills of IS professionals: A model for curriculum development. *Journal of Information Technology Education, 3*, 117-131.

Olson, J. S., & Olson, G. M. (2003). Culture surprises in remote software development teams. *QUEUE, 1*(9), 52-59.

Osland, J. S., & Bird, A. (2004). Beyond sophisticated stereotyping: Cultural sensemaking in contex. In S. M. Puffer (Ed.), *International management: Insights from friction and practice* (pp. 56-66). Armonk, NY: M.E. Sharpe.

Pfeffer, J. (1994). *Competitive advantage through people: Unleashing the power of the work force*. Boston: Harvard Business School Press.

Potts, C., & Catledge, L. (1996). Collaborative conceptual design: A large software project case study. *Computer Supported Cooperative Work, 5*(4), 415-445.

Robey, D., Gupta, S. K., & Rodriguez-Diaz, A. (1988). Implementing information systems in developing countries: organizational and cultural considerations. In S. C. Bhatnagar & N. BjØrn-Andersen (Eds.), *Information technology in developing countries* (pp. 41-50). New York: Elsevier Science Publishers.

Sahay, S., Nicholson, B., & Krishna, S. (2003). *Global IT outsourcing: Software development across borders*. Cambridge, UK: Cambridge University Press.

Sarker, S. (2003). Knowledge transfer in virtual information systems development teams: An empirical examination of key enables. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences* (pp. 119-128).

Saunders, C., van Slyke, C., & Vogel, D. R. (2004). My time or yours? Managing time visions in global virtual teams. *Academy of Management Executive, 18*(1), 19-31.

Schuerkens, U. (2003). The sociological and anthropological study of globalization and localization. *Current Sociology, 51*(3/4), 209-222.

Schultz, M., & Hatch, M. J. (1996). Living with multiple paradigms: The case of paradigm interplay in organizational culture studies. *The Academy of Management Review, 21*(2), 529-557.

Schwarzkopf, A. B., Saunders, C., Jasperson, J., & Croes, H. (2004). Strategies for managing IS personnel: IT skills staffing. In M. Igbaria & C. Shayo (Eds.), *Strategies for managing IS/IT personnel* (pp. 37-63). Hershey, PA: Idea Group Publishing.

Shukla, M., & Sethi, V. (2004). An approach of studying knowledge worker's competencies in software development team. *Journal of Advancing Information and Management Studies, 1*(1), 49-62.

Swanson, D. A., Phillips, J., & Head, N. W. (2003, June 8-12). Developing growing need for soft-skills in IT professionals. *Proceedings of the 2003 ASCUE Conference* (pp. 263-269). Myrtle Beach, SC.

Swart, J., & Kinnie, N. (2003). Sharing knowledge in knowledge-intensive firms. *Human Resource Management Journal, 13*(2), 60-75.

Trauth, E. M. (2000). *The culture of an information economy: Influences and impacts in the Republic of Ireland*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Trauth, E. M., Farwell, D., & Lee, D. (1993). The IS expectation gap: Industry expectations versus academic preparation. *MIS Quarterly, 17*(3), 293-307.

Trauth, E. M., Huang, H., Morgan, A., Quesenberry, J., & Yeo, B. J. K. (2006). Investigating diversity in the global IT workforce: An analytical framework. In F. Niederman & T. Ferratt (Eds.), *Human resource management of IT professionals.* Hershey, PA: Idea Group Publishing.

van Solingena, R., Berghoutb, E., Kustersc, R., & Trienekensc, J. (2000). From process improvement to people improvement: Enabling learning in software development. *Information and Software Technology, 42*(14), 965-971.

Walsham, G. (1993). *Interpreting information systems in organizations.* New York: John Wiley & Sons.

Walsham, G. (2000). IT, globalization and cultural diversity. In C. Avgerous & G. Walshem (Eds.), *Information technology in context: Studies from perspective of developing countries* (pp. 291-303). Aldershot, UK: Shgate Publishing.

Walsham, G. (2001). *Making a world of difference: IT in a global context.* Chichester, UK: John Wiley & Son.

Waterson, P. E., Clegg, C. W., & Axtell, A. M. (1997). The dynamics of work organization, knowledge, and technology during software development. *International Journal of Human-Computer Studies, 46*(1), 79-101.

Weisinger, J. Y., & Trauth, E. M. (2002). Situating culture in the global information sector. *Information Technology and People, 15*(4), 306-320.

Weisinger, J. Y., & Trauth, E. M. (2003). The importance of situating culture in cross-cultural IT management. *IEEE Transactions on Engineering Management, 50*(1), 26-30.

Worsley, P. (1984). *The three worlds.* Chicago: The University of Chicago Press.

Zolin, R., Hinds, P. J., Fruchter, R., & Levitt, R. E. (2004). Interpersonal trust in cross-functional, geographically distributed work: a longitudinal study. *Information and Organization, 14*(1), 1-26.

This work was previously published in Managing IT Professionals in the Internet Age, edited by P. Yoong, pp. 254-276, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 6.11
Business Modeling in Process-Oriented Organizations for RUP-Based Software Development

Francisco J. Duarte
Blaupunkt Auto-Rádio Portugal, & Universidade do Minho, Portugal

João M. Fernandes
Universidade do Minho, Portugal

Ricardo J. Machado
Universidade do Minho, Portugal

ABSTRACT

Several organizations nowadays are not particularly comfortable with their internal structuring based on a hierarchical arrangement (sub-divided in departments), where collaborators with a limited view of the overall organization perform their activities. Those organizations recognize the need to move to a model where multi-skilled teams run horizontal business processes that cross the organization and impact suppliers and clients. To develop software systems for any organization, the development process must always be appropriate and controlled. Additionally, for organizations that want to migrate to a horizontal business processes view, it is required to model the organizational platform where the organizational processes will run. This necessity is also true when the organization under consideration is a software house. In this chapter, a proposal of a generic framework for process-oriented software houses is presented. The way of managing the process model and the instantiation of their processes with the rational unified process (RUP) disciplines, whenever they are available or with other kind of processes, is recommended as a way to control and define the software development process. To illustrate the usefulness of the proposal, the chapter presents

how the generic reference framework was executed in a real project called "Premium Wage" and shows, in some detail, the created artifacts (which include several UML models) during the development phases following the RUP disciplines, especially the artifacts produced for business modeling.

INTRODUCTION

A generic reference framework for process-oriented organizations is presented in Fernandes and Duarte (2004). Here, that framework is specialized to the specific case of organizations that develop software (software houses) and we describe its main characteristics. From now on, the term "target organization" is used to refer to those organizations where the software is deployed and installed. The term "software house" is used to refer to the organization that develops software to run in the target organizations.

The main objective of this chapter is to present a reference framework based on processes and RUP disciplines for software houses and to show its usage in a real software development project, as a demonstration case to illustrate the applicability of the proposed model.

With the proposed framework, a holistic view of any software house is straightforward to obtain, allowing a more accurate definition of those processes directly related with the software development, without disregarding the management and support processes.

Process-Oriented Organizations

The concept of a process-oriented organization is a way of focusing the activities of an organization toward the clients needs (Hammer, 1996). These activities are oriented toward and validated by the clients, whose necessities must be satisfied efficiently and with quality. Reengineering, and its process orientation, must be applied to anticipate change and not as a corrective procedure when bad business indicators occur. In process-oriented organizations, clients' needs must be continuously satisfied, which mandates an easy and fast adaptation to changes. This favors and forces the continuous improvement of every aspect of the enterprise, being it process-, product- or organizational-related.

Information technologies are among the principal factors to permit a process-based restructuring of a given organization (Spurr, 1994). The development of a software application for organizations of this kind must consider their process framework. Thus, the software engineering processes must take into account the organization structure. With this model, the application becomes more useful to the target organization, and maintenance is facilitated since no major modifications and adaptations to the process framework are needed.

A process framework inside an organization contains processes, and these can be viewed as a set of activities that has as inputs and outputs a set of services and/or materials. This view must be oriented toward the necessities of the client and to the creation of added-value. This implies that the clients' requirements must always be considered, both in the design and in the performance of the system. In an organization, there are other processes rather than those that provide added-value to the clients. The existence of different types of processes is necessary to assure, for example, the strategic planning for the organization, the recruitment of the human resources or the fiscal duties. As illustrated in Figure 1, these processes are instantiated in Management and Support Processes.

Within an organization, the management by processes requires a structure that differs from the typical functional hierarchy. It is mandatory to synchronize the processes among them and to fulfill the strategic objectives of the organization. For a process-oriented organization, a structure with the following elements should exist:

- **Process management top team:** This team includes the top managers, all process owners and, if existing, the process management structural responsible. Its mission is to revise all the processes according to the strategic objectives of the organization, to analyze the effectiveness of the process-oriented management and to decide about unsolved problems at the processes' interfaces.
- **Process sponsor:** The mission of this top manager is to help and instruct the process owner, to decide when there is a problem of interface among processes, to determine the strategic orientation of the process and to assure that the process is uniform within the organization.
- **Process owner:** For each process, its owner must have know-how on managing processes and persons and competency in the areas associated with the process. His mission is to lead the process' multi-disciplinary team.
- **Multi-disciplinary team:** This team must be created for each added-value process, since they represent the most important processes for the clients. The mission of this team is multi-fold: to monitor its process, to define and analyze the key indicators and the process objectives, to ensure that the process documentation is updated, to decide when and how to use improvement teams and to coordinate them and to manage the process execution teams.
- **Execution teams and team leaders:** These teams and their leaders represent the instances of a given process (Scheer & Nüttgens, 2000). Therefore, during the execution of a process, some teams will use it with a specific focus. For example, for a given production process, one team may be responsible for producing parts for industrial clients, while another team may produce them for individual clients.

To align a process-based organization with its strategic objectives, it is crucial that the goals are based on the organization mission and vision, and also on its principles and values. Based on those strategic objectives and in the business plan, the priority when deciding the key business processes within the organization can be perceived and included in the process landscape. This action may imply that some processes, activities or tasks can be eliminated if they do not add any value to the clients or to the organization. These eliminated (or redefined) processes, activities and tasks, and their respective consequences in terms of reorganization and impact in human resources, are the essence of re-engineering (Hammer, 1996).

Demonstration Case

The proposals made in this chapter were tested in a real industrial environment, more specifically in the first author's organization.

The project, titled "Premium Wage," consists of the development of a software application to calculate the payment of extra money to employees, based on their productivity, quality and absenteeism (Fernandes & Duarte, 2004). This project was considered critical, since it is likely to have important social and behavioral impacts on the organization, namely, if the amount is badly calculated or if it is impossible to explain how it was obtained. This premium was introduced with the aim of ameliorating the organization's overall productivity and quality, and to return the excellence to the workers.

Besides its criticality, the business process is also complex since it depends on other processes. In this case, the payment of a premium depends on three main factors: individual absenteeism, quality of the products made in the employee's line and individual performance. The first two sub-processes were extended in order to support new functionalities. For the third, a complete

reengineering was carried out. Finally, a new process was designed, modeled and implemented to the premium wage calculation.

In the project, the proposed reference framework, namely RUP's business modeling discipline, is extensively used and we evaluate the capacity of the process to cope with complex organizations.

Structure of the Chapter

This chapter is structured in five main sections. The second section presents the main characteristics of the reference framework for Process-Oriented software houses, namely the processes they are composed of. In the third section we describe RUP's business modeling core discipline, which implements an added-value process. In the fourth section we describe and discusses in detail the produced artifacts for the demonstration case during the execution of the business modeling discipline. In fifth section, future trends and work along with the conclusions are presented.

REFERENCE FRAMEWORK FOR PROCESS-ORIENTED SOFTWARE HOUSES

A reference framework, also called PSEE (process-centered software engineering environment), does not support the notion of a predefined process model that is supposed to be applied in every development project, but instead supports a wider variety of processes based on parameterization (Engels et al., 2001). SPADE (Bandinelli et al., 1994), EPOS (Conradi et al., 1994), MELMAC (Gruhn & Jegelka, 1992), OIKOS (Montangero & Ambriola, 1994), OPEN (Henderson-Sellers, 2000), ESF (Gasch et al., 1987) and APPL/A (Sutton et al., 1995) are well-known examples of PSEE systems or process modeling formalisms.

A reference framework for process-oriented organizations is presented and justified in Fernandes and Duarte (2005). In this chapter, an updated version of this particular framework (Figure 1) includes a new process, called *change management*, that allows a more explicit management of continuous improvement and changes. This framework has some of its processes consubstantiated with RUPs disciplines. Since RUP is a process meta-model, we can obtain a specific process model by tuning some of the parameters, allowing our reference framework to be tailored for each kind of project.

According to the classification described in Fettke et al. (2005), the framework in Figure 1 has the following main characteristics:

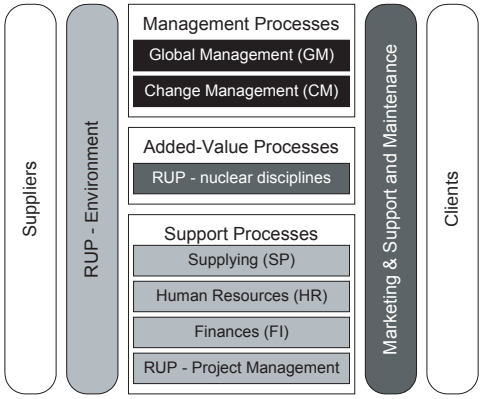
- **Construction:** domain differentiation (institution), domain description (software houses), modeling language (graphical, UML activity diagrams and RUP business use cases), modeling framework (yes) and evaluation (demonstration case)
- **Application:** method (procedure model), reuse and customization (customization of model contents)

The reference framework intends to cope with all issues related to software development processes and potentially can be used with the following purposes: documentation, analysis and improvement of software process models (Bandinelli et al., 1993), software process improvement (Avrilionis et al., 1996) and software process execution (Deiters & Gruhn, 1998).

Figure 1 represents a top-level view of the process landscape, which is useful to show and discuss with top managers and process owners. Afterwards, this view must be further refined by the process owner and the multidisciplinary team, to present the process at the appropriate level of detail for each software house professional.

It is also important to notice that Figure 1 corresponds to a specialization of the general framework presented in Fernandes and Duarte (2005) for the particular situation of organizations whose

Figure 1. Reference framework for software houses



main activity is to develop software. The execution of the TTM process by the software house produces an instance of the general framework that models the organization where the software will run. This fact makes the general framework valuable in two senses: (1) as a reference for the software house to model itself, and, (2) as a reference for the software house to model the target organization.

In software houses, we propose the business processes to be organized into two different groups: (1) the first one includes processes that exist in any organization independently of its relation to the software development business; (2) the second group includes processes that present specific characteristics due to the fact that the organization's main activity is the development of software-based solutions.

Management and Support Processes

We can observe several processes in the context of the management and support issues that are common to any type of organization.

The *global management* process includes the sub-processes *global strategy* (GS), *policy deployment* (PD) and *business plan* (BP). This process is equivalent to that of any other organization, although we must take into account the particularities of the software market, such as the rapid changes in technology and the competition in worldwide markets when defining, for instance, an organizations' business and vision.

Once stable process models have been obtained, they should be released and afterwards it is desirable to manage their changes, according to the change requests made by process stakeholders (Gruhn & Wellen, 2000). The *change management* (CM) process allows the software house to collect, organize and manage the output data and experiences that are the basis for processes self improvement. This new process should exist in any software house aiming to reach the highest CMM (*capability maturity model*) levels (Paulk et al., 1995). To reach the highest CMM levels, the concern about the constant improvement of the development processes must be part of each process, instead of being only a matter of the CM

process. CMM level 3, which is considered the minimum when discussing the software process (Henderson-Sellers, 2000) is reachable with RUP if it is extended accordingly, for example, to the proposals made in Manzoni and Price (2003). Also, rules and procedures should be defined previously to cope with the introduction of significant changes in the organization.

The *supplying* (SP) process consists essentially in creating copies of an application. In contrast with more traditional industries, where it represents probably the most important process, in software, due again to its intangible nature, this is a trivial process. The usual outsourcing of this process comes from the fact that it is considered to be secondary for an organization that develops software. Therefore in this kind of organization, this process is a support one.

The *human resources* (HR) process for software factories is the same as for other types of organizations. We must, however, point out that software development requires highly specialized people, making their hiring a critical issue for the success of the organization. It is impossible to produce quality software without skilled people.

The *finances* (FI) process is the typical fulfillment of the fiscal obligations, which is common to all types of organizations and may also include controlling activities.

The processes *marketing* and *maintenance and support*, represent the typical *customer relationship management* (CRM) process. This ensures that when a software application is delivered to the final clients, its life-cycle does not end at that time, but instead continues by incorporating changes and corrections, providing training to the users, while the application is being used by the clients. Hotline support activities can also exist as an included sub-process.

Added-Value Processes

In the context of added-value processes, the proposed reference framework is influenced by the fact that RUP constitutes a systematic approach to assign tasks and responsibilities to its members. The main aim of RUP is to construct quality software that meets the requirements of the stakeholders, within a typical engineering context (Machado et al., 2005). RUP identifies and defines the activities needed to map user requirements into a software application and is accepted to be a generic/customizable process that can be adapted for a wide range of contexts, namely organizations with distinct CMM levels, different skills and tools and unequal number of team members.

Since software is an intangible product, it is obvious that no raw materials are needed to produce it. For organizations that develop software, RUP's *environment* discipline can instantiate the *supplier relationship management* (SRM) process, since it furnishes the working environment (e.g., development tools) and the development guidelines to be followed by the teams.

The RUP's core disciplines (*business modeling, requirements, analysis and design, implementation, test and deployment*) represent the most critical activities for an organization that develops software and can be seen as the *time-to-market* (TTM) process of the organization. This set of activities, or sub-processes, run in parallel for the same development project.

The RUP's discipline *project management* implements the *data management* (DM) process. In this discipline, some activities lead to the production of indicators of the project status. Its existence is the foundation to take decisions based on facts, related to the advance of the project aiming to adjust and improve the software development process.

Business Modeling in RUP

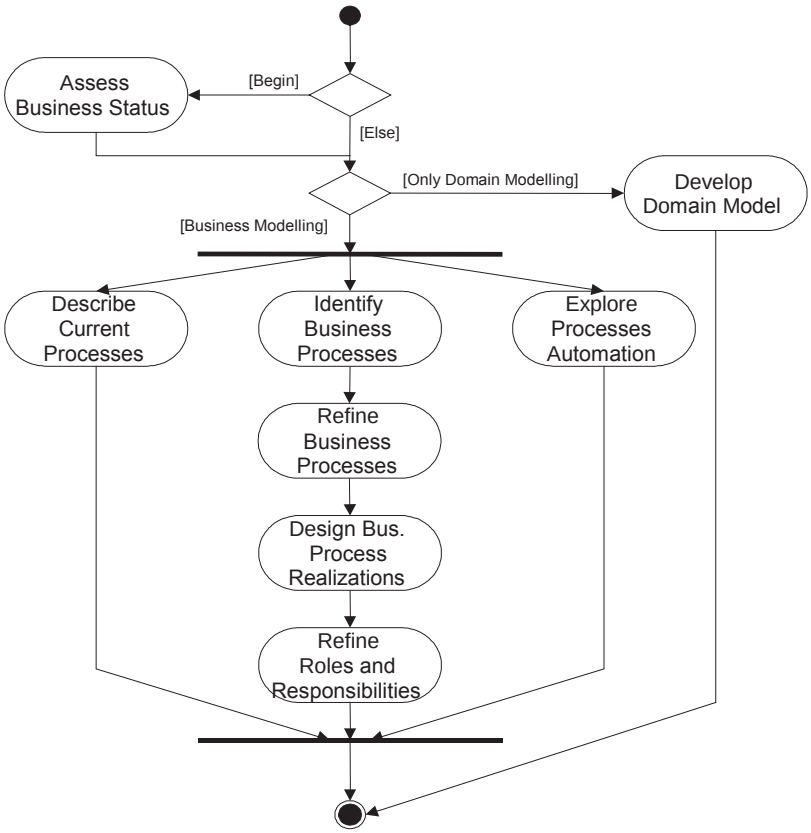
RUP's core disciplines implement added-value process. These disciplines are sub-divided in activities, which can be viewed as sub-processes. The description of those sub-processes is made with UML activity diagrams, complemented

optionally with other kinds of diagrams, such as interaction and business object diagrams.

This representation is also valid for all other processes of a generic organization. Any time a software house starts a new project, the TTM process is executed. Since we propose this process to be implemented by the six RUP's core disciplines, it implies that *business modeling* will also be executed. Among the recommended diagrams by this discipline for modeling purposes are included activity diagrams. Thus, a target organization will be modeled also by a collection of these diagrams. Additionally, within the software house, the discipline *business modeling* itself can also be modeled by activity diagrams, since it is a sub-process of the TTM process (Figure 2).

During software development, all the stakeholders must have a common understanding of the business processes that exist in the target organization. This reality is not circumscribed to the obvious organizational information systems. If the development of applications does not take into account the current business processes (or those to be implemented), the result will be probably unsuccessful. This may be caused by the fact that end users do not employ correctly the application, since it does not support directly the activities under their responsibility.

Figure 2. Activity diagram to help the execution of RUP's business modeling discipline in TTM process



The main activities of *business modeling* are centered on the identification, refinement and realization of the business processes and in the definition of the roles of people associated to the business. Each role in this RUP's discipline has under its responsibility the execution of several activities that will have as deliverables several artifacts (Table 1).

The activities of Table 1 are at a detailed level than those of Figure 2. For example, the activity *refine business processes* includes the activities *structure the business use case model*, *review the business use case model* and *detail business use cases.*

Among all the activities and their respective artifacts, only some are mandatory. This flexibility permits the configuration of RUP, so that it can be adapted to a specific project executed in a given organization.

Table 1. Roles, activities, and main artifacts for business modeling in RUP

Role	Sub-Activity	Activity (Figure 2)	Main Artifacts
Analyst of the Business Process	Assess target organization	Assess Business Status	Business rules Business use case model Business glossary Business object model Business vision Supplementary business specification Target organization verification Business architecture
	Set and adjust goals		
	Capture the business vocabulary		
	Find business actors and use cases	Describe Current Processes	
	Maintain the business rules	Assess Business Status & Identify business processes	
	Structure business use case model	Refine Business Processes	
	Define the business architecture	Identify Business Processes	
Reviewer of the Business Model	Review the business use case model	Refine Business Processes	
	Review the business object model	Refine Roles and Responsibilities	
Designer of the Business	Detail business use cases	Refine Business Processes	Organizational units
	Find business workers and entities	Describe Current Processes	
	Define the automation requirements	Explore Processes Automation	
	Detail business entities	Refine Roles and Responsibilities	
	Detail business workers		

BUSINESS ARTIFACTS FOR THE DEMONSTRATION CASE

Nowadays, the technology-planning horizon for big companies is a synthesis of software and process engineering (Smith & Fingar, 2002). Additionally, the success of a project depends heavily on the correct perception of the business process to be modeled. Taking into account these two aspects, the RUP's *business modeling* discipline assumes a critical role in the software development process. The artifacts that can be generated by this discipline have the following objectives:

- To understand the structure and dynamics of the organization where the system will be executed
- To comprehend the current problems of the target organization and to identify potential improvements
- To assure that clients, final users and developers have a common understanding about the target organization
- To capture/deduct the requirements of the system necessary to support the target organization

RUP can be parameterized and used both in small and complex projects; next we discuss what artifacts were produced for the demonstration case. For this parameterization to occur, it is necessary, during project execution, to choose which artifacts to use and their level of detail. This choice was validated by quality assessments contained in the process model as milestones between phase transitions. Thus, both the subset of used artifacts and also its degree of detail can not be anticipated with rigor, but must be selected based on experience and knowledge of the development team in relating the characteristics of each project with the functionalities offered by the artifacts. The criteria to fulfill this choice are related with:

- Characteristics of the project itself (e.g., criticality of the modeled business processes, type of target organization)
- Characteristics of the organization that develops software (e.g., team size, level of knowledge about internal rules)
- Temporal restrictions. Since resources are limited in engineering projects, it is always a necessary balance between the quantity and detail of the produced artifacts and the deadlines for implementing the project.

The produced artifacts result from a set of activities that occur inside those disciplines. In this demonstration case, we identified the need for the artifacts to represent two distinct situations in terms of business: One part of the project represents reengineering activities of some business processes, while the other part represents the introduction of a new business process. In several diagrams (e.g., business use case model), the standard UML is augmented with the stereotypes defined by RUP, thus allowing the creation of RUP-like artifacts.

Business Rules

The business rules correspond to policy statements and conditions that should be fulfilled from the business perspective. They are similar to systems requirements, but they focus on the business core, expressing rules related to business, and also its architecture and style. Its modelling must be rigorous, one possibility being the use of the object constraint language (OCL) as specified in UML. Alternatively, business rules can be modeled with structured English, using some fixed constructors (Odell, 1998).

The usage of structured natural language with fixed constructs and with a pseudo-programming language syntax was chosen due to the necessity to validate directly with key users the perception of the development team about stated business rules, and also because key users have a generic engi-

Table 2. Business rules examples for Premium Wage

Rule #	Description
1	**If** worker_has_conflict_inside_team **or** worker_is_under_disciplinary_process **then** premium(worker) = 0
2	**Switch** absenteeism(worker) **case** = 0h: premium(worker) = premium(worker) * 1 **case**]0-4h]: premium(worker) = premium(worker) * 0.75 **case**]4-8h]: premium(worker) = premium(worker) * 0.5 **case** >8h: premium(worker) = premium(worker) * 0
3	**If** line_productivity_not_available **or** absenteeism_not_available **or** quality_factor_not_available **then** premium(worker) = 0

neering background. This way, in the particular situation of the demonstration case, the usage of natural language was preferred over OCL.

Table 2 shows, as examples, three rules for Premium Wage system.

Rule 1 describes a situation where no premium payment is due when the worker has conflicts in its team or is under a disciplinary process, even though productivity, quality and absenteeism are at good levels for that worker. Rule 2 assigns a weight factor to the final premium based on the worker's absenteeism. Rule 3 states that if there is no information available from one of the three premium factors, no payment will be made for that worker.

Business Use Case Model

The main goal of this artifact is to show how the business is being perceived and run by stakeholders. This is achieved by modeling the business processes and their interactions with external parties, based on business use case diagrams (with stereotypes for business use cases and business actors) (Fernandes & Machado, 2001; Machado & Fernandes, 2002).

Business processes models should specify how added value is created for the business actors. Activity diagrams, possibly extended with the representation of organizational units interfering in the business process and with the distribution of the activities by those organizational units, can support this modelling. The knowledge about "who is doing what" should be obvious when reading this model.

Business use case model is the first description of the business functionalities and actors inside the target organization. For Premium Wage, one artifact was created to model the current situation (Figure 3) and another for the desired future situation (Figure 4).

The existence of these two models shows to all stakeholders, using the same notation and same detail level, the first perspective of effort amount

needed to reach a future situation, and more important, acts as a base for the target organization's management to decide on business re-engineering and improvement. A special emphasis should be given to explain to the target organization's management that an information system is not the complete business reality, but an abstraction of it. Real business processes, such the ones stated in *business modeling* discipline, are far more extent than the ones implemented inside the information system. This may be caused by people's activities because they do not act similarly every day, and mainly because people like to have their own special information systems (e.g., spreadsheets or personal databases) to make decisions and cause the *starvation* of the organization business information system with missing data. This is a crucial success factor at the present time because the work is no longer only individual or departmentally related, but exists through horizontal business processes that cross the entire organization and reaches external partners.

Figure 3 shows the current situation, where only three business use cases exist and only two business actors take part. The line productivity is calculated by the controller-based master data lines' team member and products maintained by the team manager. This situation shows that no information was available on how and where to collect quality data and absenteeism. Next, in Figure 4, the desired extensions and new business use cases can be seen. For the future situation, six business actors are needed, and special care should be put on this situation because some of them may not be directly related with project activities and may not understand the business value of the new activities (stated in the business use cases) if no proper involvement and education is provided.

In this situation, line productivity is *triggered* also by the controller, but to be useful, it first has to be transformed into individual performance (because one employee can work in several production lines for the premium calculation

Figure 3. Business use case model: As-Is situation

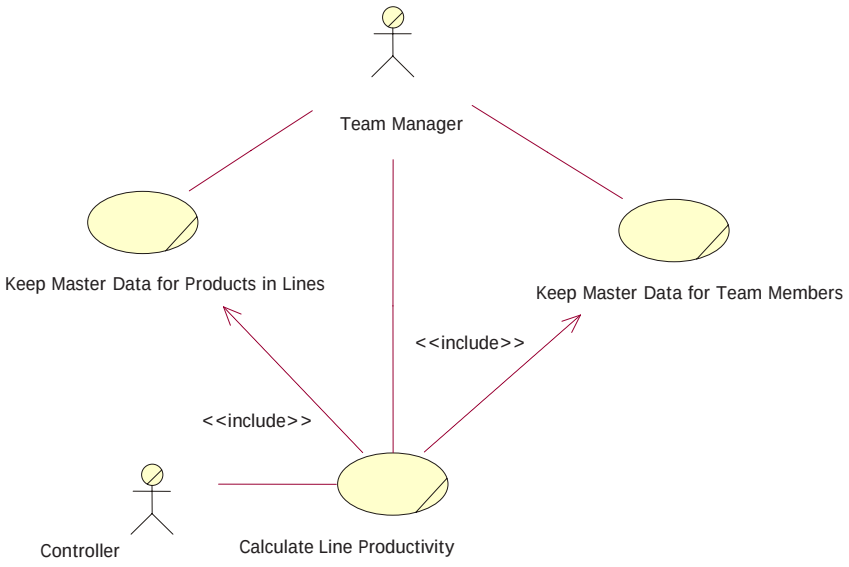
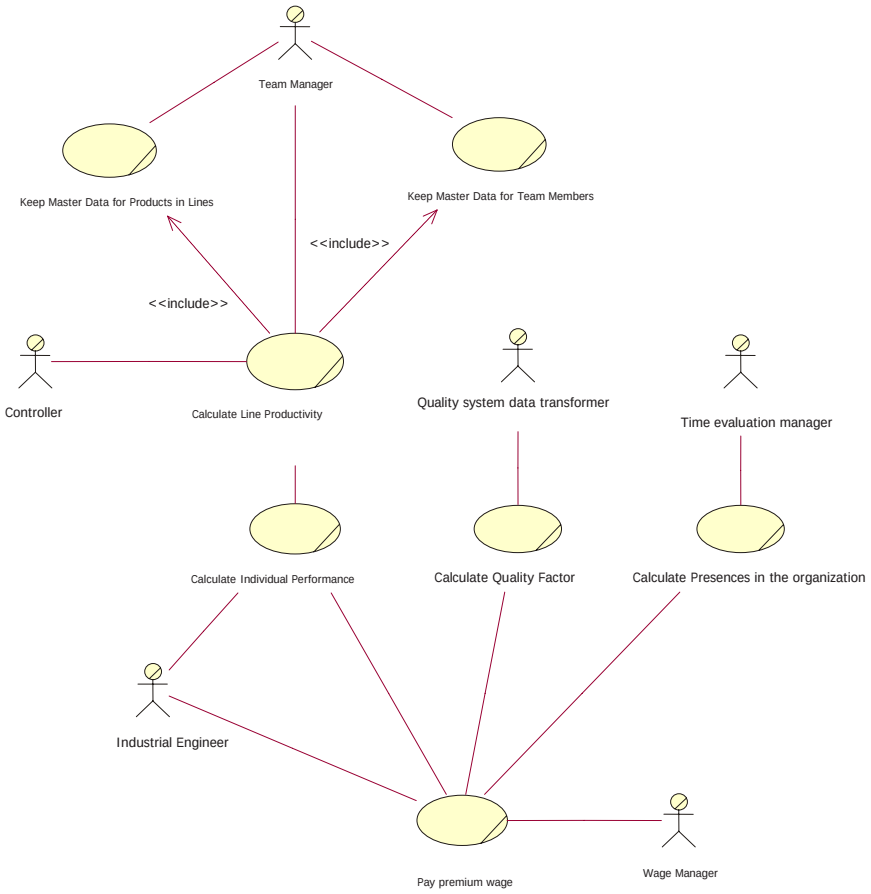


Figure 4. Business use case model: To-Be situation



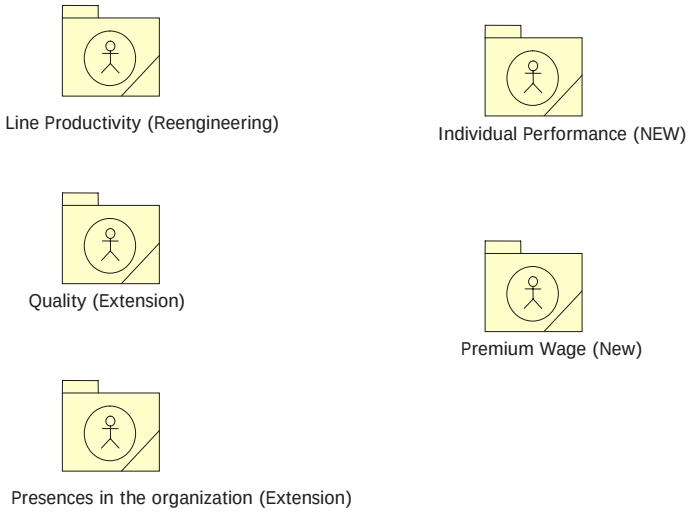
time frame). Additionally, the remaining two premium factors (quality and absenteeism) are stated, thus allowing the industrial engineer to calculate the final values and the wage manager to handle them.

Organizational Units

This artifact is used to reduce the complexity and structure of the business object model by dividing it into smaller parts. For the demonstration case, five organization units where created (Figure 5), each one representing a collection of business workers, business entities, relationships, business use-case realizations, diagrams and other organization units.

The organization unit *line productivity* includes the re-engineered activities for the productivity calculation, *quality* and *presences in the organization* where extended to cope with premium calculation. The *individual performance* and *premium wage* calculation organizational

Figure 5. Organizational units



units were newly created in the organization, because none of the existent capabilities could handle it.

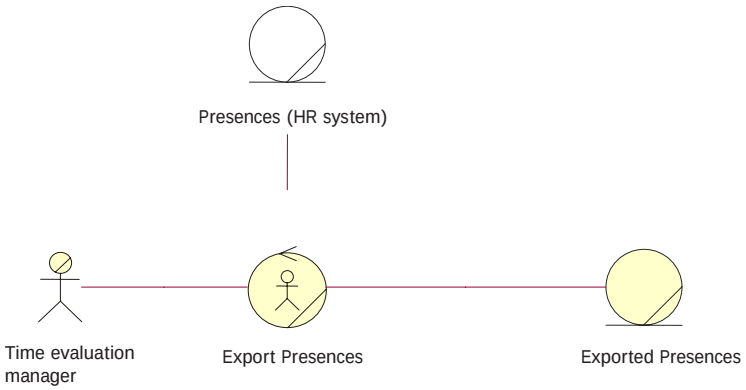
Business Object Model

This artifact is an object model describing the realization of business use cases. It serves as an abstraction of how business workers and business entities need to be related and how they need to collaborate in order to perform the business. In Figures 6 and 7 the realizations of some To-Be situation business use cases are presented, namely *Presences in the organization* and *Pay premium wage*.

The importance of this artifact comes also from it being a basis for identifying future information system actors and use cases, and in addition, to identifying classes for analysis and design models. First modelling the business and afterwards the information system is a crucial practice to correctly align the business reality with the abstraction the information system implements. If the software information system development only starts with the modelling of a software system, a quality software product can be obtained (being the quality perceived as the degree of correctly implementing requirements) but that may fail completely to cope with the business reality of the organization in which it will run. Although business information systems implement a large percentage of the business practices, there are always some parts of the organization's business processes that are not implemented in software systems. The modeling of such business practices by the software development team is a wise and safe procedure to avoid future inconsistencies when introducing the information system in the organization.

To calculate the presences in the organization (Figure 6), the business worker *Time evaluation manager* uses business entity *Export presences*

Figure 6. Business object model: Calculate presences in the organization



to pick the business object *Presences* available in the HR system and generate a new business object *Exported presences* containing the presences in the organization in a proper format to use inside the Premium Wage information system.

In Figure 7, business entity *Calculate Premium Values* uses several business objects representing final premium factors, such as *Compiled Quality System data*, *Exported presences* (generated in Figure 6, business object model) *Individual Performance* and also business objects representing levels and limits (acting as data supports to implement business rules (Table 2) to generate the business object *Premium to Pay.* Afterwards, this business object will be used and transformed by several business entities, with and without the intervention of business actors, until the final Employee Total Wage (Wage + Premium) is loaded and calculated in the HR system, as shown in Figure 7.

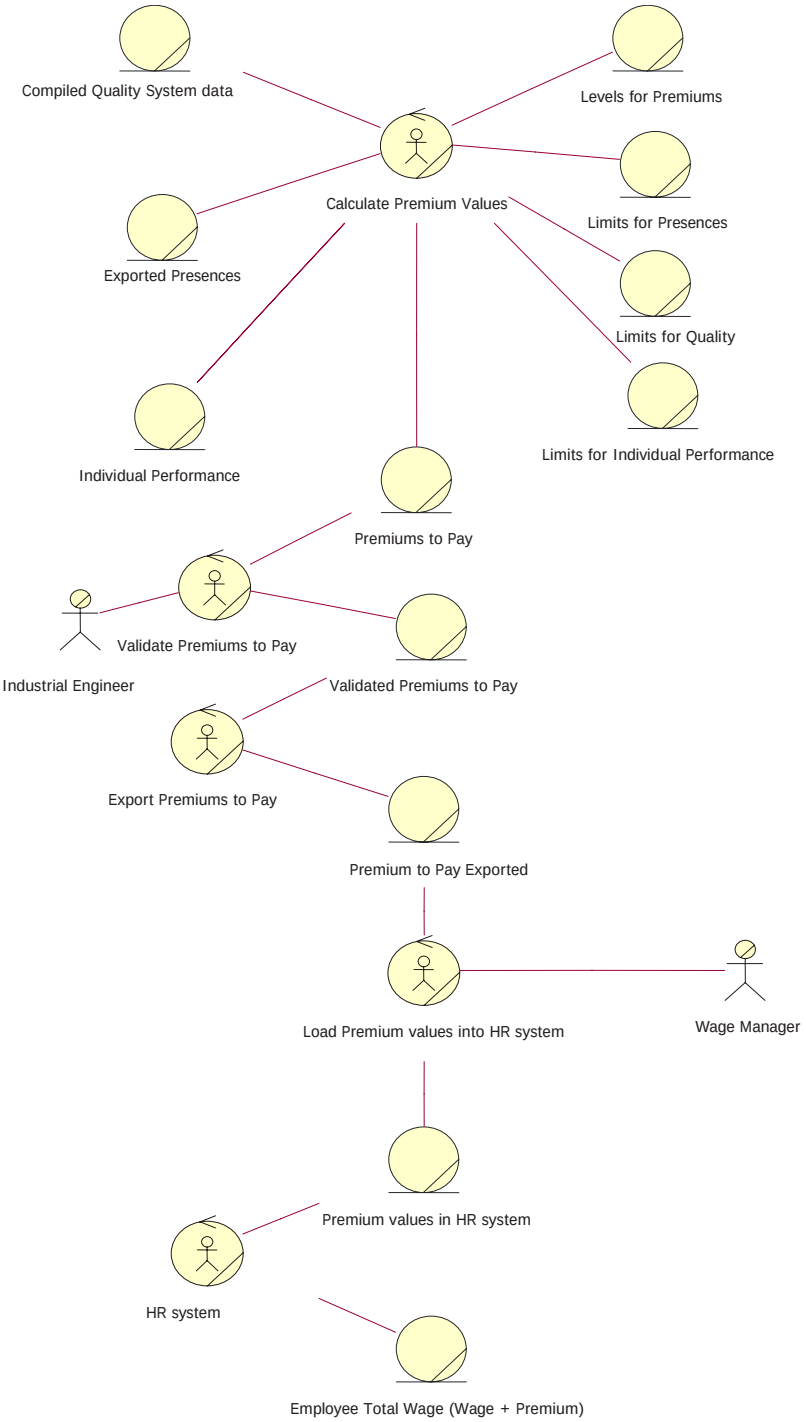
The business object model exposed in Figure 7 is not fully contained in Premium Wage information system. The business object Premium to Pay Exported is generated by Premium Wage to be used inside the organization's HR system. This is an example where some of the business actors, business entities and business objects will not be part of the information system required by the software development team, but they were modeled to guarantee a proper integration of current information systems (e.g., HR system) with newly developed ones (e.g., Premium Wage).

Other Business Artifacts

From all of the proposed artifact in RUP for business modeling, some were not used in the Premium Wage demonstration case. Next, we present the reason why they were not needed in the instance of the RUP we used for this project:

- **Business glossary:** In this artifact all business terms and expressions are kept. They are necessary for a good understanding among all project stakeholders. In our situation, the business terms are common to the target and to the software developing organization because they both are sub-organizations of the same organization.

Figure 7. Business object model: Pay premium wage



- **Business vision:** This artifact captures the goals of a particular business modeling activity, stating what is to be modeled and the reasons why. It also serves as an oracle to all future high-level decisions. We did not use business vision because it is common to the target and software development organization due to the same reasons as those of business glossary.
- **Supplementary business specification:** No need for extra-detail than that available in the business use case and the business object models.
- **Target organization verification:** The target organization is perfectly known by the software developers. The current processes are modelled by the business use case model, As-Is situation.
- **Business architecture document:** The detail level presented in the business use case model and in the business object model is sufficient to understand the business architecture.

CONCLUSION AND FUTURE TRENDS

In this chapter, we have presented a revised version of a reference framework for process-oriented software houses which serves as a foundation to model organizations. This specific framework is based on a more generic one, which is also used as a template to model the target organization in which the software product is to be executed. Additionally, we also show the way of managing the framework and the instantiation of its processes with RUP disciplines whenever feasible.

We also discuss in detail the usage of the framework within a demonstration case, and more particularly, the produced artifacts during the execution of the RUP's business modelling discipline. The modeling capabilities of a graphical modelling language (such as the UML) and the understanding that it gives to all the stakeholders was a crucial factor in the demonstration case to avoid communication and interpretation errors and to improve the solution utility and correctness.

As future work, we plan to model the reference framework with UML (Fettke et al., 2006) and formalize the processes of the framework by using colored Petri nets (Jensen, 1992). Similar approaches also based on Petri nets were also experienced with results (Gruhn & Wellen, 2001; van der Aalst, 2003).

By adopting a formal language, it is possible to model, animate, simulate and formally verify the properties of each single process. Additionally, we intend to explicitly model the interfaces between the business processes in our reference framework, which allows the complete framework to be analyzed, verified and validated.

We intend to automatically generate CPN skeletons from business requirements models. A semantic layer in the Arena environment (Kelton et al., 2002), capable of accepting CPN-based business specifications, will also be developed to allow the stochastic execution of workflow scenarios as a complement to some current validation approaches based on CPN/Tools (Beaudouin Lafon et al., 2001).

After formally describing the reference framework processes, we can use them in every organization (in this case, software houses) to compare with current processes. This comparison (based on the same Petri net formalism) should allow a quick assessment of the organization against world-class processes and consequently permit the re-engineering and improvement of its own processes. In this way, the reference framework acts as a To-Be model to be compared with the As-Is model of the software house. The detected mismatches show the improvement areas for the software house to proceed accordingly within the organization vision and mission.

REFERENCES

Avrilionis, D., Belkhatir, N., & Cunin, P.-Y. (1996). Improving software process modelling and enactment techniques. In C. Montangero (Ed.), *The 5th European Workshop on Software Process Technology*, Nancy, France (LNCS 1149, pp. 65-74).

Bandinelli, S., Fuggetta, A., & Grigolli, A. (1993). Process modelling in-the large with SLANG. In *Proceedings of the 2nd International Conference on the Software Process: Continuous Software Improvement*, Berlin, Germany (pp. 75-83). IEEE.

Bandinelli, S., Fuggetta, A., Ghezzi, C., & Lavazza, L. (1994). SPADE: An environment for software process analysis, design, and enactment. A. Finkelstein, J. Kramer, & B. Nuseibeh (Eds.), *Software process modeling and technology*. London: Research Studies Press.

Beaudouin-Lafon, M., Mackay, W. E., Andersen, P., Janecek, P., Jensen, M., Lassen, M., et al. (2001). CPN/Tools: A post-WIMP interface for editing and simulating coloured Petri nets. In *Application and Theory of Petri Nets 2001, Proceedings of the 22nd International Conference (ICATPN 2001)*, Newcastle upon Tyne, UK.

Conradi, R., Hasgaseth, M., Larsen, J.-O., Nguyên, M. N., Munch, B. P., Westby, P. H., et al. (1994). EPOS: Object-oriented cooperative process modeling. A. Finkelstein, J. Kramer, & B. Nuseibeh (Eds.), *Software process modeling and technology*. London: Research Studies Press

Deiters, W., & Gruhn, V. (1998). Process management in practice: Applying the FUNSOFT net approach to large scale processes. *Automated Software Engineering, 5*, 7-25.

Engels, G., Schäfer, W., Balzer, R., & Gruhn, V. (2001). Process-centered software engineering environments: Academic and industrial perspectives. In *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Canada (pp. 671-673). IEEE CS Press.

Fernandes, J. M., & Duarte, F. J. (2005). A reference framework for process-oriented software development organizations. *Software and Systems Modeling (SoSyM), 4*(1), 94-105. Springer-Verlag.

Fernandes, J. M., & Duarte, F. J. (2004). Using RUP for process-oriented organisations. In F. Bomarius & H. Iida (Eds.), *Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004)* (LNCS 3009, pp. 348-362). Springer-Verlag.

Fernandes, J. M., & Machado, R. J. (2001). From use cases to objects: An industrial information systems case study analysis. In *Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS '01)* (pp. 319-328). Springer-Verlag.

Fettke, P., Loos, P., & Zwicker, J. (2005). Business process reference models: Survey and classification. In *Proceedings of the Workshop on Business Process Reference Models (BPMR 2005)*, Nancy, France (pp. 1-15).

Fettke, P., Zwicker, J., & Loos, P. (2006) Using UML for reference modeling. In P. Rittgen (Ed.), *Enterprise modeling and computing with UML*. Hershey, PA, USA: Idea Group Inc.

Gasch, B., Kelter, U., Kopfer, H., & Weber, H. (1987). Reference model for the integration of tools in the EUREKA software factory. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow* (pp. 183-189). IEEE Computer Society Press.

Gruhn, V., & Jegelka, R. (1992). An evaluation of FUNSOFT nets. In *Proceedings of the 2nd European Workshop on Software Process Technology (EWSPT '92)* (LNCS). New York: Springer-Verlag.

Gruhn, V., & Wellen, U. (2000). Structuring complex software processes by "Process Landscaping." In *Proceedings of the 7th European Workshop on Software Process Technology (EWSPT 2000),* Kaprun, Austria (LNCS 1780, pp. 138-149). Springer Verlag.

Gruhn, V., & Wellen, U. (2001). Process landscaping: Modelling distributed processes and proving properties of distributed process models. *Unifying Petri nets* (LNCS 2128, pp. 103-125). Springer-Verlag.

Hammer, M. (1996). *Beyond reengineering: How the process-centered organization is changing our work and our lives.* New York: Harper Collins.

Henderson-Sellers, B. (2000). The OPEN framework for enhancing productivity. *IEEE Software, 17*(2), 53-58.

Jensen, K. (1992). Coloured Petri nets: Basic concepts, analysis methods and practical use, Vol. 1: Basic concepts. In *EATCS monographs in theoretical computer science.* Springer-Verlag.

Kelton, W. D., Sadowski, R. P., & Sadowski, D. A. (2002). *Simulation with ARENA* (2nd ed.). New York: McGraw-Hill.

Machado, R. J., & Fernandes, J. M. (2002). Heterogeneous information systems integration: Organizations and methodologies. In M. Oivo M. & S.K. Sirviö (Eds.), *Proceedings of the 4th International Conference on Product Focused Software Process Improvement—PROFES '02*, Rovaniemi, Finland (LNCS 2559, pp. 629-643). Springer-Verlag.

Machado, R. J., Ramos, I., & Fernandes, J. M. (2005). Specification of requirements models. In A. Aurum & C. Wohlin (Eds.), *Engineering and managing software requirements* (pp. 47-68). Berlin: Springer-Verlag.

Manzoni, L. V., & Price, R. T. (2003). Identifying extensions required by RUP (rational unified process) to comply with CMM (capability maturity model) levels 2 and 3. *IEEE Transactions on Software Engineering, 29*(2), 181-192.

Montangero, C., & Ambriola, V. (1994). OIKOS: Constructing process-centered SDEs. A. Finkelstein, J. Kramer, & B. Nuseibeh (Eds.), *Software process modeling and technology.* London: Research Studies Press.

Odell, J. (1998). *Advanced object-oriented analysis & design using UML*. Cambridge University Press.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (Eds.). (1995). *The capability maturity model: Guidelines for improving the software process.* Reading, MA: Addison-Wesley.

Scheer, A. W., & Nüttgens, M. (2000). ARIS architecture and reference models for business process management. In W. van der Aalst, J. Desel, & A. Oberweis (Eds.), *Business process management, models, techniques, and empirical studies* (LNCS 1806, pp. 376-389). Springer.

Smith, H., & Fingar, P. (2002). *Business process management: The third wave*. Tampa, FL: Meghan-Kiffer Press.

Spurr, K., Layzell, P., Jennison, L., & Richards, N. (1994). *Software assistance for business reengineering.* New York: John Wiley & Sons.

Sutton, S., Heimbigner, D., & Osterweil, L. (1995). APPL/A: A language for software process programming. *ACM Transactions on Software Engineering Methodology, 4*(3), 221-286.

van der Aalst, W. M. P. (2003). Challenges in business process management: Verification of business processes using Petri nets. *Bulletin of the European Association for Theoretical Computer Science, 80,* 174-198.

This work was previously published in Reference Modeling for Business Systems Analysis, edited byP. Fettke & P. Loos, pp. 98-117, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 6.12
Improvement of Software Engineering by Modeling Knowledge-Intensive Business Processes

Jane Fröming
University of Potsdam, Germany

Norbert Gronau
University of Potsdam, Germany

Simone Schmid
University of Potsdam, Germany

ABSTRACT

The Knowledge Modeling and Description Language (KMDL®) allows analysts to identify process patterns, which leads to improvements in knowledge-intensive processes. After modeling the business processes, knowledge and process potentials in daily business processes can be unleashed. The following contribution presents a specification of KMDL® for software engineering (KMDL®-SE). A real-life example is used to explain KMDL®-SE.

INTRODUCTION

Software development is a knowledge-intensive business process. Until now, no adequate methods were available to improve knowledge management in software engineering by appropriate models, analyses, and concepts. It seems useful to combine more than 10 years of experience in the modeling and analysis of information processing tasks, applying methods like event-driven process chains and establishing a new modeling paradigm focused on knowledge creation. Its application in

the area of software engineering is described in the following contribution.

The main focus of the contribution refers to two principal objects: First, knowledge-intensive business processes in software engineering can be identified and improved using an adapted modeling language. Second, the specification of the Knowledge Modeling and Description Language (KMDL®) is used to model an exemplary software engineering processes. In the section "Theoretical Foundation of KMDL®," the theoretical framework of KMDL® is presented. The modeling language is used to describe knowledge-intensive business processes, tacit and explicit knowledge, and knowledge and information flows. In the section "Analysis of Potentials with KMDL®-SE," potential improvements for software engineering processes are proposed. In the section "Real-Life Application of KMDL®-SE," a real-life example of using KMDL® is described. "Identification of Process Patterns" describes the conclusions and identified process patterns that can be found and used to reorganize the software engineering processes in software engineering companies as well as for the specification of KMDL®. The last section gives future prospects of further developments of KMDL®-SE.

QUESTIONS AND PROBLEMS OF KNOWLEDGE MANAGEMENT IN SOFTWARE ENGINEERING

The dynamic behavior of the actual business environment will gain speed and complexity. The market for software products will transform very quickly, and the pressure due to competition is expected to increase massively. Especially small and medium-sized enterprises have to cope with the high pressure in the software engineering sector consisting in the rivalry between themselves and major players (Groff & Jones, 2003). Therefore, methods and applications are needed to identify potentials in daily business processes (Hamel & Prahalad, 1990). The knowledge and use of these potentials can be a decisive competitive advantage. The management and processing of organizational knowledge increasingly are being viewed as critical to organizational success (Inkpen & Dinur, 1998).

The contribution is based on the central thesis: The productivity of software engineering will be increased using appropriate knowledge management applications.

Software engineering processes have to be improved in a way that relevant information and knowledge have to reach the appropriate employee at the right time. If so, employees reduce unnecessary waiting time for information and knowledge; therefore, tasks can be completed more quickly. Another way to increase the productivity of software engineering is a constant documentation and optimization of recurring subprocesses and a reuse of these as patterns in other projects. Knowledge management activities in software engineering can be effective only if they are implemented and applied consequently throughout the company. Even the greatest strategies will be unsuccessful without the support of employees. Staff members have to deal with knowledge management, and its advantages have to be made clear.

In the following sections, the central thesis will be discussed, applying it to a real-life example of software engineering in a small and medium-sized enterprise. A German software engineering firm was analyzed within the research project M-Wise[1]. M-Wise is based on the German federal government's software engineering research initiative 2006. The interdisciplinary organized project aims to promote knowledge management in software engineering. Existing methods and applications to model knowledge-intensive business processes were improved, and a new specification of a modeling language in software engineering could be developed and tested in a real-life environment.

MODELING OF SOFTWARE ENGINEERING PROCESSES WITH THE KNOWLEDGE MODELING AND DESCRIPTION LANGUAGE

In the following section, the Knowledge Modeling and Description Language KMDL® is introduced. KMDL® is currently under development at the University of Potsdam in Germany. The theoretical framework of KMDL® as well as a general procedural model for its implementation is described. Finally, this article will point out the use of KMDL® especially for software engineering, for which a new specification (KMDL®-SE) is introduced. Further references to the modeling method are available in Gronau and Weber (2004) and Gronau and Weber (2004a).

Knowledge-Intensive Business Processes in Software Engineering

Within process-oriented knowledge management, the knowledge-intensive business process is the primary perspective (Remus, 2002). Several attempts have been made in the literature to define knowledge-intensive business processes. Heisig (2002) points out the opportunity to schedule the knowledge demand and evaluates knowledge intensity according to the existence of variability and exceptions (Mertins et al., 2000). Other sources define a process as knowledge-intensive if an improvement with conventional methods of business reengineering is not or is only partially possible (Remus, 2002). Davenport recognizes the knowledge intensity by the diversity and uncertainty of process input and output (Davenport & Prusak, 2000). A process is knowledge-intensive if its value can be created only through the fulfillment of the knowledge requirements by the process participants. Several properties that are typical of knowledge-intensive business processes are introduced in the following list (Gronau et al., 2005):

- In knowledge-intensive processes, knowledge contributes significantly to the values added within the process. Innovation and creativity plays a major role in such processes (Eppler et al., 1999). People within the process have a large scope in the freedom of decision, meaning that they can decide autonomously.
- The event flow of knowledge-intensive business processes is not clear in advance, as it can evolve during the process (Davenport & Prusak, 2000).
- The participants in the process have different experiences and bring in knowledge from different domains with a varying level of expertise (Heisig, 2002).
- The lifetime of knowledge involved in the process is often very short (Eppler et al., 1999). It quickly becomes obsolete. It is usually very time-intensive to build up this knowledge.
- Usually knowledge-intensive business processes do not follow structured working rules and often lack metrics to evaluate the success of the process (Davenport & Prusak, 2000).
- IT-tools for knowledge-intensive business processes are generally not very sophisticated, because the knowledge usually is transferred through socialization and an informal exchange of knowledge.
- Often the costs of knowledge-intensive processes are very high.

Software engineering consists of different knowledge fields such as requirements analysis, software design, software testing, and software configuration management (Abran et al., 2004). These fields can be analyzed and evaluated in terms of their knowledge-intensive characteristics. Except for the software configuration management, all the mentioned fields of applications are affected by a high degree of innovation and

autonomy. They do not follow structured working rules, and various individuals with different expertise are involved in the process. All that the knowledge fields have in common is that there is great variety of sources and media, a high demand for communication, a short half-life period of knowledge, and high process costs; therefore, it can be defined as knowledge-intensive processes.

In order to successfully create knowledge-intensive business processes in software engineering, a method to reflect knowledge flows is necessary. This dimension cannot be brought out adequately by methods like event-driven process chains (Van der Aalst, 1998), because the ability to display the essential elements of knowledge flows are missing, especially the description of tacit knowledge. The process-based approach can provide an efficient way to capture and navigate knowledge (Kim et al., 2003).

Theoretical Foundation of KMDL®

The first version of KMDL® was developed by Professor Gronau at the University of Oldenburg. The motivation for the development was the lack of appropriate methods to model knowledge-intensive business processes. At the University of Potsdam, KMDL® is improved continuously by the Knowledge Management Group. A procedural model is developed as well as a mechanism for the process analysis. KMDL® is a description language that is suitable to model knowledge-intensive business processes. It has the ability to identify available knowledge that exists in or is necessary for the process, including its origin and application. The following section introduces the theoretical concepts that are used to define the Knowledge Modeling and Description Language. The first paragraph outlines the tacit and explicit knowledge defined by Nonaka and Takeuchi. In the second paragraph, the concept of knowledge conversion will be introduced.

Tacit and Explicit Knowledge

KMDL® uses the understanding of tacit knowledge according to Nonaka and Takeuchi (1995). They follow the thoughts of Polanyi (1958), who introduced the terms of tacit knowledge and explicit knowledge. Polanyi (1958) defines tacit knowledge as personal knowledge bound to humans. This type consists of mental models, beliefs, and perspectives (Nonaka & Takeuchi, 1995). It is partially unconscious and, therefore, difficult to be communicated and explained by the persons who possess it.

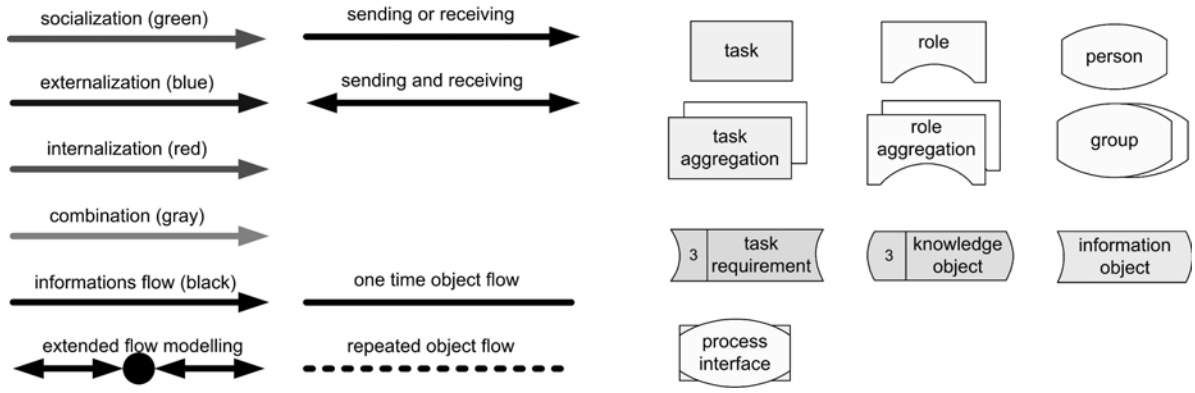
Explicit knowledge, on the other hand, easily can be expressed in handbooks, papers, patents, or software (Gronau & Weber, 2004). It is formal, codified, systematic, easy to communicate, shared, and can be articulated in writing and numbers (Schmidt et al., 1996). This also means that it can be transmitted and stored for reuse by other people.

Numerous life-cycle models adopt a similar staged view of knowledge flow (Nissen, 2002; Nissen & Levitt, 2004). Nonaka goes further still, as he introduces a model describing a spiral of five dynamic interactions between tacit and explicit knowledge along an epistemological dimension, and he characterizes four processes (socialization, externalization, combination, internalization) that enable individual knowledge to be amplified and to effect organizational knowledge crystallization along the ontological dimension (Nonaka & Takeuchi, 1995).

KMDL® Description Language

The basis for a software engineering specification of KMDL® is version 1.1 (see Figure 1). Tasks represent the frame of the KMDL® model description. They describe the logical sequence of the business process. The prerequisite for the execution of a task is certain information as

Figure 1. Associations and objects of KMDL® v1.1



input. The generated information is the result or the output of a task. The information flow connects these objects. Tasks are executed by roles. A role integrates textual or organizational-related activities of a task. For each role, requirements to execute the specific task in the desired manner in order to generate the required output using the given input are modeled. The person who performs a task is assigned to a role. The knowledge objects (tacit knowledge, see the section "Tacit and Explicit Knowledge") are attached to the person. These objects are required in order to execute a task. Each knowledge object is linked to a person and is personal. The attribute level describes the qualification of each knowledge object and task requirement. At present, the attribute level contains four degrees: 0 means no knowledge, 1 means basic knowledge, 2 means intermediate knowledge, and 3 means expert knowledge.

Attributes are used for more detailed descriptions of objects. For instance the attribute knowledge domain is provided for each knowledge and information object, assigning it to a specific topic. This enables the hierarchical assignment of knowledge and information objects and, hence, the description of used explicit and tacit knowledge within the considered process.

In addition, the description language enables the visualization of knowledge flows (see Figure 1). From the business process perspective, the flows of knowledge as well as the conversions of knowledge into other knowledge types are of significant importance. New objects of knowledge or information are created by the transformation of objects existing in the process. This transformation is performed by an interaction of knowledge and information objects. As an analogy to Nonaka/Takeuchi, KMDL® distinguishes between four types of knowledge conversion (Gronau & Weber, 2004):

- **Internalization.** Internalization means the conversion of explicit knowledge into tacit knowledge. It is related very closely to learning by doing. Experiences made through socialization, externalization, or a combination are internalized and integrated into the individual's knowledge framework.

The internalization is started by an information object and ends with a knowledge object.

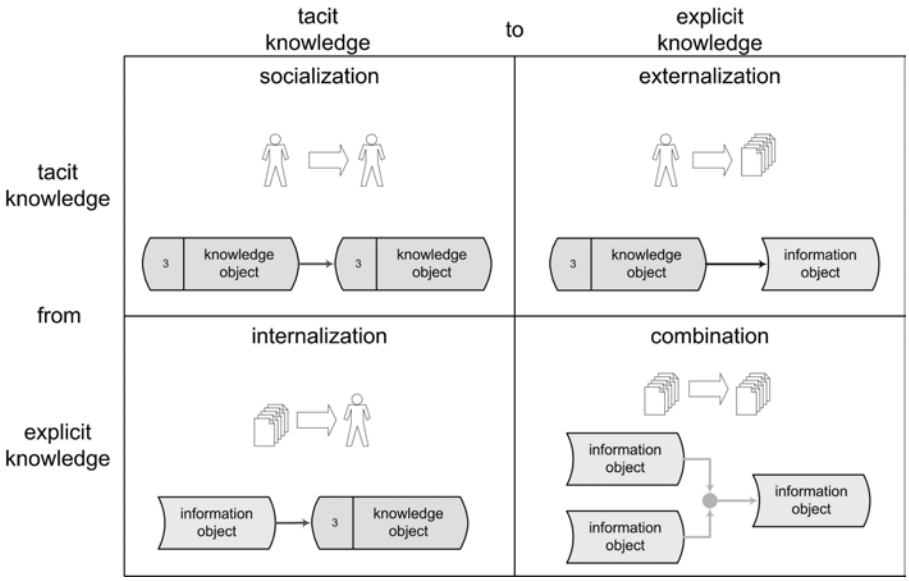
- **Externalization.** Externalization is defined as the transformation of tacit into explicit knowledge. The problematical aspect within this conversion is that important and person-bound parts will get lost, because it is difficult or, in some cases, impossible to externalize tacit knowledge. The externalization is modeled beginning with at least one knowledge object and ending with an information object.
- **Socialization.** Socialization is defined as a conversion from tacit to tacit knowledge. The most common way is by sharing experience: Just like apprentices of a craftsman learn skills by observation, a knowledge worker can learn required abilities through on-the-job training. The socialization does not demand for spoken or written words. Socialization is modeled starting with a knowledge object of one person to a knowledge object of another person.
- **Combination.** Combination is the conversion from explicit to explicit knowledge. During a combination, one or more information objects are used to create a new information object.

Unlike other process modeling tools, the main focus of KMDL® is to support all four expressions of knowledge conversion (see Figure 2). It allows a systematical description of process knowledge and the identification of all containing information. Origin and usage of knowledge and information can be defined as well. A conversion is either once (continuous line) or repeated (dashed line).

Specification of KMDL® for Software Engineering (KMDL®-SE)

Software engineering is a field of activity with a high percentage of knowledge-intensive business

Figure 2. Model of the dynamics of knowledge creation



processes. Competitive as well as sustainable software solutions have to be developed. In addition, there are factors like a fast-changing technical basis and a dynamic personnel situation. Modern software engineering processes have to find a way between these demands and conditions. An effective and efficient knowledge transfer within and between projects is an important factor for success, especially for an early identification and estimation of risks.

Besides the description of classical business processes, KMDL® provides the systematical identification and analysis of knowledge flows and transformations. This allows identifying knowledge monopolies, unused competencies, or unsatisfied demands. Thereby, actions to enhance knowledge-intensive processes can be taken. The development of a new knowledge management approach specifically for software engineering is based on the previous version of KMDL®, which was enhanced toward KMDL®-SE. It enables users to configure knowledge acquisition, storage, and transfer in software engineering effectively and efficiently. Starting points of the KMDL®-SE development were the complex and, to a high degree, knowledge-intensive software engineering processes of several business partners that accompanied the project. The project partners gave significant input in order to determine requirements and to evaluate the KMDL® methodology (Rombach et al., 1993). Simultaneously, the K-Modeler tool that supports the modeling of knowledge-intensive SE processes was developed and a prototype implemented.

Analysis of Potentials With KMDL®-SE

The K-Modeler automatically identifies and evaluates various design patterns in the modeled processes, which hence helps to analyze the process. These process patterns are derived from known disadvantageous process elements and structures found in knowledge-intensive processes (Brown et al., 1998). The concept of patterns was coined originally by the architect and mathematician Christopher Alexander. Patterns can be used to find solutions for recurring problems, reusing existing knowledge (Greenfield, 2004; Kirchner & Jain, 2004). During the 1990s, the concept of patterns and best-practice solutions was transferred in subareas of software engineering (Gamma et al., 1995). The principle of patterns is used in KMDL® to analyze knowledge-intensive business processes (KMDL, 2006). Thereby, a single process pattern describes a specific situation that repeatedly occurs during these processes. It is an indicator of hidden process potentials and points out opportunities for an alternative process design. During several projects, a multitude of patterns could be identified:

- **Occurrence Patterns.** The pattern of occurrence shows where specific objects appear with exceptionally high frequency in the business processes. If a specific person frequently occurs in a process, a knowledge monopoly might be present. The pattern can show that one person holds knowledge of high relevance for the process; hence, problems can surge when this person resigns. A better responsibility assignment can be a valid improvement. The occurrence pattern also can be used to determine the occurrence of information, knowledge, roles, tasks, and task requirements objects.
- **Multi-Step Patterns.** The multi-step pattern category describes a combination of two conversions, whereby transitions from tacit to the explicit process level and vice versa will be analyzed. There is also an examination of conversion doubling on the same level. Twelve different combinations of knowledge conversions are imaginable, but only some of the combinations can be used to improve the process design. The multi-step socialization pattern is one example of a multi-step pattern in which information gets

lost during a double socialization (Chinese Whisper).

- **Relevance Patterns.** Relevance patterns refer to tasks with a high degree of complexity and knowledge intensity. There are four types of relevance patterns that indicate tasks with a great amount of input (knowledge objects, information objects), output (knowledge objects, information objects), integrated persons, and task requirements. A suggestion for improvement can be task reorganization and, therefore, splitting up tasks.
- **Exclusive Patterns.** Two types of the exclusive pattern category are distinguished: exclusive information and exclusive knowledge pattern. Certain information or knowledge objects are requested very frequently in the business process. The loss of these information or knowledge objects can lead to a process disruption. Therefore, the information or knowledge has to be secured, for example, through externalization of specific knowledge objects.
- **Prerequisite Pattern.** The prerequisite pattern describes a person involved in the process who has the ability to fulfill a task only after the generation of knowledge through socialization with a noninvolved person. The informal acquisition of knowledge depends on personal preferences and the personal social network. A way of improvement can be the institutionalization of the knowledge transfer.

Usually, the mentioned patterns cannot be identified without modeling the business process. Therefore, modeling knowledge-intensive business processes with KMDL®-SE is an important step to detect strengths and weaknesses. As soon as the patterns are identified, proposals to improve the process can be made. This is an effective way to identify and unleash potentials systematically.

REAL-LIFE APPLICATION OF KMDL®-SE

In this section, the application of KMDL®-SE in a software developing company will be described. First, the company will be introduced, and then the main objectives of the real-life example will be explained. This will provide a better understanding of usage of KMDL® in software engineering.

The real-life application was carried out in a small software developing company with 25 employees based in Berlin, Germany. The company, founded in April 1999, today attends the entire middle European market. Since its establishment, the company has offered two browser-based content- and knowledge-management products based on the IBM Lotus Domino platform. These systems enable the maintenance of complex Internet, intranet, and extranet sites. Flexible integration modules provide interfaces and import and export functionalities. Beneath the permanent improvements of their standard software products, the firm also offers professional consulting, training, and individual development in conjunction with the product implementation. To cope with the changing market and client requirements, the company decided in 2003 to migrate the development environment of its content management system from IBM Lotus Domino Platform to IBM WebSphere.

Principal Objects of the Practical Example

For a reasonable design and analysis of software engineering processes, an adequate modeling is fundamental. Therefore, it is of great importance to consider the knowledge-intensive processes that occur during the development of new software products and even to build the main part of it. Based on the already existing Knowledge Modeling and Description Language (KMDL®), the practical example had two intentions:

- **Analysis of the Existing Software Engineering Processes**
  The complete analysis of the existing software engineering processes is the first step in order for a company to reorganize and, therefore, to improve these processes. All knowledge conversions and knowledge carriers have to be identified. Furthermore, the existent knowledge in the company should be identified, which could be useful for future developments. In this real-life example, the main focus was directed to the analysis of standard software engineering processes. As a result, the complete model of standard software engineering processes with the existing version of KMDL® could be presented. By mapping these standard development processes to one meta-model, critical points and risks as well as potentials in the existing software engineering processes could be identified. With the help of KMDL®, an effective redesign for the post migration processes became feasible.
- **Expansion of the Knowledge Modeling and Description Language (KMDL®) to a New Specification for Software Engineering Processes**
  The customization of the KMDL® for software engineering processes is the starting point for process reorganization. The existing version of KMDL® had to be adapted to the theoretical and practical requirements of the software engineering processes. The implementation in practice identified problems, which led to extensions and revisions of KMDL®. Putting KMDL® into practice allowed discovering critical points and insufficiencies in general. A further intention was to point out opportunities for improvement and to check the KMDL® approach for its relevance in software engineering. Thus, the drawing of profound conclusions for a new specification for KMDL® in software engineering became possible.

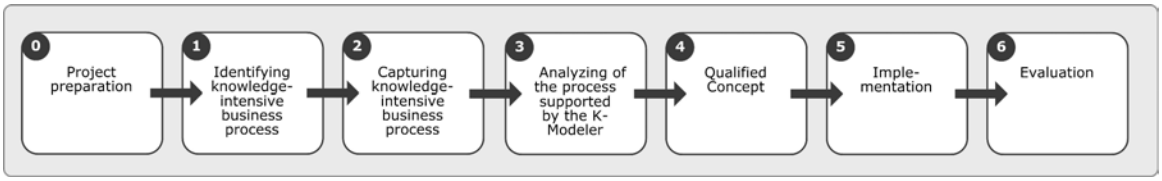
KMDL® Procedural Model

In order to determine the potentials for process improvements, a detailed analysis of knowledge-intensive business processes is required. The procedural model ensures the correct collection of data and information (see Figure 3). It is the basis for KMDL® process models and it was used in the following real-life example:

After project acquisition and preparation, it is necessary to identify knowledge-intensive processes. For this selection, a criteria catalogue can be used. It consists of more than 30 properties that are typical of knowledge-intensive business processes. In the next phase, current knowledge-intensive business processes are mapped. KMDL® offers a subprocedure within the six steps: definition of tasks associated with the process, identification of the information input and output, assignment of the persons to the specific roles, executing the task, specification of the role requirements, and assignment of the knowledge objects to the accompanying person. During the third phase, the modeling is realized with the K-Modeler tool. After going through the six steps, possible process improvements can be generated. The next step is to implement the recommendations in practice.

Detailed mapping of the knowledge-intensive business process is a requirement for the analysis and evaluation of potential improvements of the process. The analysis of the process contains the identification of knowledge intensity, the process scheme, and process potentials (weak spots). As-is models illustrate the ownership, demand, development, and use of knowledge. Further on, it is possible to visualize the knowledge intensity on a knowledge map for the entire process, a process part or for a single activity of the respective tasks. This procedure allows a classification of single tasks or the weighting of their relevance. The results are used for recommendations of technical and organizational improvements. The comparison of as-is models of different in-

Figure 3. KMDL® procedural model

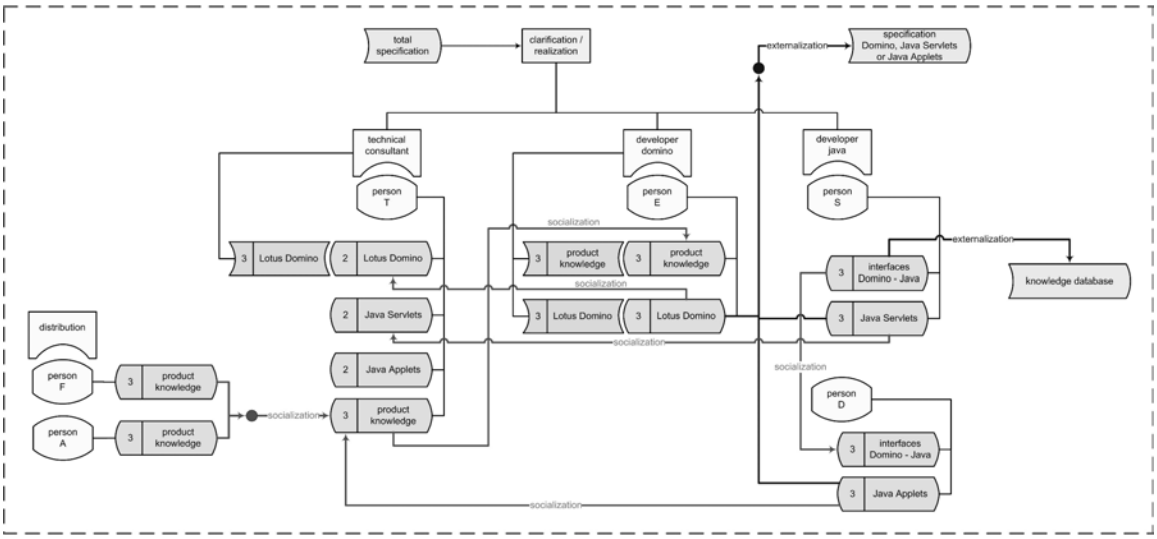


stances from the same process is useful in order to generate universally valid statements about process elements and element relations. Selected knowledge-based activities should be examined in order to identify specific patterns. It is recommended to extend the existing reference processes with this information in order to support future participants of the process.

Process Modeling in Software Engineering

In this case, three process instances of software engineering were modeled following the KMDL® procedural model. The individual process instances are used to derivate an organization-specific process model for the company. An organization-

Figure 4. Extended organization-specific process model



specific process model describes the approximated process flow and contains only all information objects and tasks, which can be found in most of the process instances.

In the next step, the organization-specific process model was enhanced with roles, persons, task requirements, and knowledge objects from the process instances in order to derivate conclusions for future software engineering processes. The resulting model is called extended organization-specific process model (see Figure 4). This model shows the process flow in the company containing all knowledge conversions. With ease, existing conversions within the software engineering process can be recognized. With these conclusions, the intended conversions can be supported directly.

Identification of Process Patterns

The following section describes the most important patterns that were identified several times in the business processes of the described software engineering company. They were used to optimize these processes as well as to improve the software engineering specification of KMDL®.

Person Occurrence Pattern

A frequent occurrence of a certain person in one process can be interpreted in two ways: First, frequent occurrence might indicate an unwanted knowledge monopoly. Second, frequent occurrence might classify the person as a specialist who supports his colleagues efficiently. Anyway, problems may surge if the specialist leaves the company and new employees are hired, because the new employees now might have the same expertise. A way to solve this issue can be the distribution of knowledge within the company. As a conclusion for the real-life example, newly hired employees socialized tacit knowledge from the specialist. In the future, the new employees are expected to take over the specialist's tasks, thereby replacing him. A major conclusion for modeling knowledge-intensive processes in software engineering is that important enterprise knowledge is frequently based on individual employees.

Figure 5 shows the KMDL® person occurrence pattern: person A is needed for all tasks within a subprocess. The person occurrence pattern (see Table 1) belongs to the family of occurrence pat-

Figure 5. Person occurrence pattern

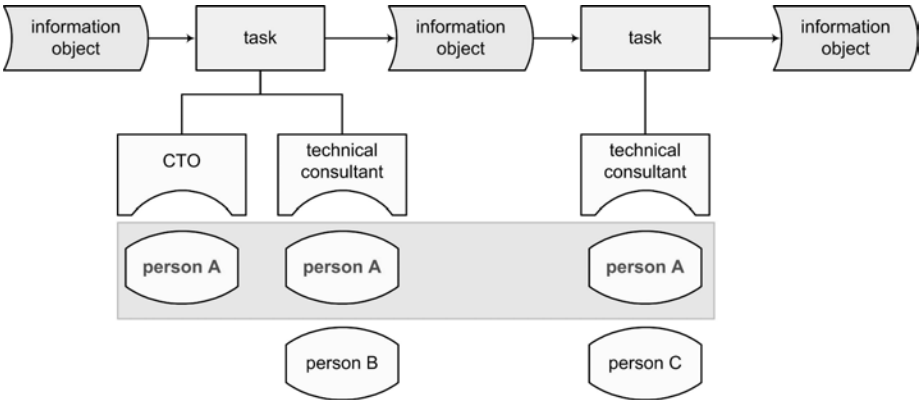


Table 1. Person occurrence pattern description

Purpose	Identification of monopolies and specialists
Description	A person is involved in many tasks within the process
Involved KMDL®-Objects	Person
Problem	Process-specific knowledge is held only by a single person. Problems can surge if this person is absent
Solution	Improved responsibility assignment

terns, such as task-, information-, knowledge-, role- and requirements-occurrence pattern.

Two-Step Socialization Pattern

The next pattern that could be identified within process modeling is a two-step socialization pattern (see Figure 6 and Table 2). This pattern is also known as Chinese whisper and belongs to the multi-step pattern family. It describes a transitive conversion between two persons (person A socializes person B, and person B socializes person C). Instead of a direct communication between person A and C, two socializations take place. Results of transitive conversion can be lost or modified information. This problem can be solved through a direct socialization. In the real-life example, German and English language skills were of significant importance. The software developers spoke only English, while the product

Figure 6. Two-step socialization pattern

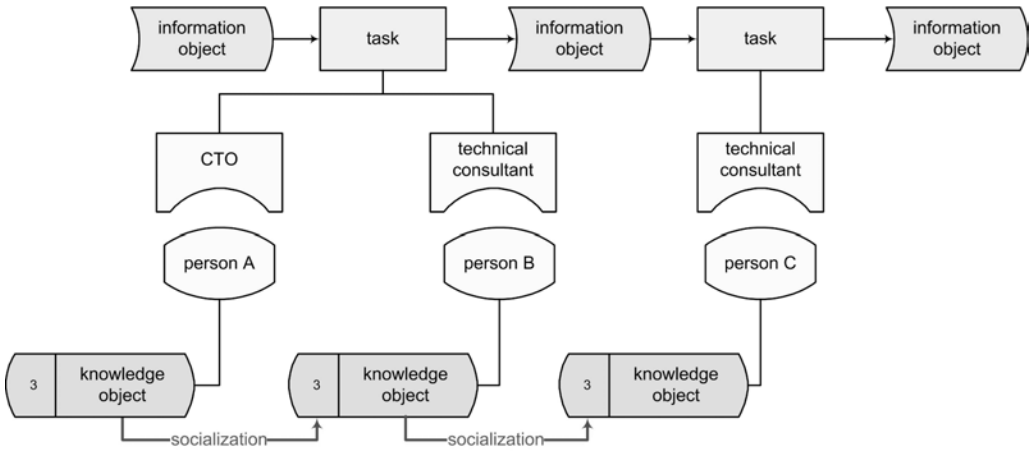


Table 2. Two-step socialization pattern description

Purpose	Identification of knowledge deformations
Description	Identical knowledge objects are transferred more than once on the tacit process level
Involved KMDL®-Objects	Persons, knowledge objects, socializations
Problem	Lost information due to indirect communication (Chinese whisper)
Solution	Support for direct communication, knowledge externalization

consultants spoke German. The consultants were the middlemen between customers and developers because the software developers could not communicate with the customers due to not being able to speak the German language. The software developers were instructed what consultants believed the customer wanted. This was one of the main sources of errors. To solve the issue, a new developer who spoke German and English was recruited. This enabled a correct translation of customer wishes into technical requirements, bypassing the consultants.

Team Relevance Pattern

The team relevance pattern (see Figure 7 and Table 3) describes the complexity of many tasks that cannot be completed by a single person. Therefore, the task requirements are very extensive and diversified. It is necessary that several persons

Figure 7. Team relevance pattern

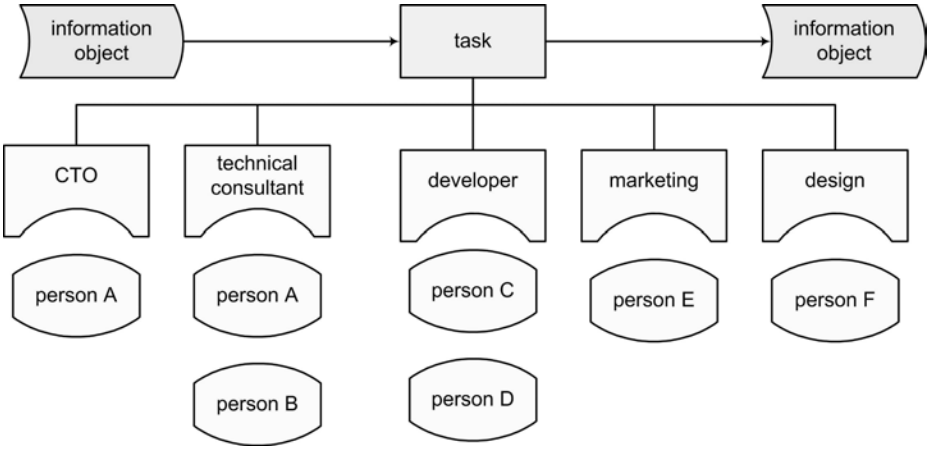


Table 3. Team relevance pattern description

Purpose	Identification of important teams
Description	Many persons are involved within one task
Involved KMDL®-Objects	Task, persons
Problem	A task is too complex to be worked out by a single person
Solution	A number of subtasks can reduce complexity

and roles be put together in order to fulfill the task successfully. It is not always a competitive advantage to reduce the number of persons within a process step. Especially knowledge-intensive business processes are characterized though complexity and often require a high degree of creativity and teamwork. Nevertheless, it can be useful to divide one task into subtasks, which can reduce complexity and make it easier to manage the task within the process.

Requirement-Oriented Team Building

Requirement-oriented team building can be realized through KMDL®. In order to guarantee a successful task realization, it is essential to build teams according to the requirements that have to be fulfilled. The team-building component is based on a term: taxonomy (see Figure 8). To clarify the functionality of the algorithm, Table 5 shows a repository including all the involved persons and their knowledge objects. These objects can be recovered in the taxonomy. Only red-marked terms are important for the real-life example; the remaining terms are relevant only in order to understand the full software engineering context.

A basic condition for the team-building component is a task. A task is defined by a couple of requirements that have to be fulfilled by a number of persons. In the practical example, a Linux server was purchased, and a Java-based application for Lotus Notes/Domino had to be developed. Therefore, the following requirements (see Table 4) had to be fulfilled by the team.

First of all, a matrix has to be set up consisting of task requirements and available knowledge objects (see Table 6). It can be recognized that Peter will not be considered, because he lacks the required knowledge object Java. Skills that are essential for all of the team members are marked (*). Requirements of particular importance and a high weighting will be indicated with (**).

Three teams were proposed to accomplish the task (see Table 7):

1. Mary (as a single person because she holds all the required knowledge objects)
2. Steve und Mary
3. Jane und Mary

The three proposed teams do not completely fulfill the requirements. More specialized or more general knowledge is available to close the gap.

Figure 8. Knowledge taxonomy (extract)

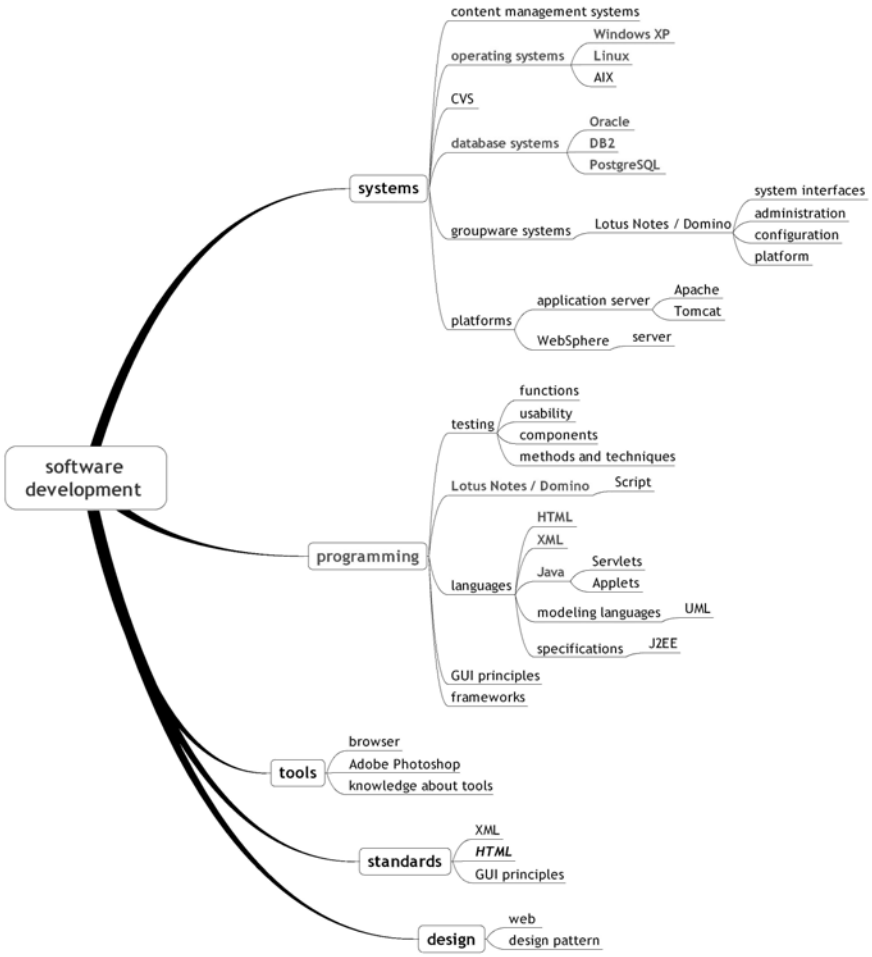


Table 4. Necessary requirements

	Necessary Knowledge (amount of persons)	Weight 8/8
Operating systems	at least one person	1/8
Linux	at least one person	1/8
Database systems	at least one person	1/8
PostgreSQL	at least one person	1/8
Lotus Notes/Domino	at least one person	2/8
XML	at least one person	1/8
Java	every team member	1/8

Table 5. Repository of persons

Steve	Peter	George	Jane	Mary
Operating systems, Windows XP, Linux, Lotus Notes/Domino, Java	Database systems, Oracle, DB2	HTML, XML, Java, DB2	Programming, Lotus Notes/ Domino, Java, Windows XP	Linux, database systems, PostgreSQL, Java, programming

Table 6. Team-building matrix

	Operating systems	Linux	Database-systems	PostgreSQL	Lotus Notes/ Domino**	XML	Java*
Steve	0	0	-5	-5	0	-5	0
~~Peter~~	~~-5~~	~~-5~~	~~0~~	~~-1~~	~~-5~~	~~-5~~	~~-5~~
George	-5	-5	+1	-5	-5	0	0
Jane	+1	-5	-5	-5	0	-2	0
Mary	+1	0	0	0	-1	-2	0

Table 7. Staffing algorithm results and team choice

	Expert Search	Team Search	Covering rate	Efficiency (covering rate/ persons)
1 person		Mary	4/8	**0,50**
2 persons		Steve, Mary	7/8	**0,43**
		George, Mary	5/8	0,31
		Jane, Mary	6/8	**0,37**
3 persons	Steve, Mary, George	Steve, Mary, George	8/8	0,33
		Jane, Mary, George	7/8	0,29
		Jane, Steve, Mary	7/8	0,29
4 persons	Steve, Mary, George, Jane	Steve, Mary, George, Jane	8/8	0,25

Within the expert search, only the team of Steve, Mary, and George are proposed.

GENERAL CONCLUSION AND FUTURE PROSPECTS

Applied to knowledge-intensive software engineering processes, KMDL® has proven to be a working concept. Current research focuses on the analysis of KMDL® process models in order to identify possible process improvements. The real-life example showed that the application of knowledge management can increase productivity in software engineering. By pointing out process patterns, potential improvements were identified. Therefore, long-winded coordination processes could be accelerated and optimized. But it has to be considered that knowledge management activities will be accepted by employees only if the advantages are clearly shown and the required effort is kept within a limit.

Based on the research results as well as the successful commercial implementation, KMDL® v1.1 now builds the backbone for the current development of KMDL® v2.0. The focus on knowledge conversions and the extension of the theory of Nonaka et al. leads toward a more powerful method that represents the actual conducted knowledge transfer, application, and creation. By introducing methods for the knowledge conversions, it is easy to distinguish the different knowledge conversions and to classify them. Therefore, they contribute to the expressiveness of the language. On the other hand, this can be used to identify best practice methods in the analysis of knowledge-intensive processes. In KMDL® v2.0 the introduction of information systems will extend the opportunities of the method and, thereby, consider technical aspects of the knowledge-intensive process. This provides an integrated view on the subject matter. The knowledge management strategy, therefore, can be arranged by considering the organizational, cultural, and technical aspects. The KMDL® v1.1 has been applied successfully in commercial and research projects. The next step with KMDL® v2.0 is the application of the method in real-life projects to measure the benefits of the improvements and to draw conclusions from the application of the method. Besides this, the analysis methods of KMDL® v1.1 that have been applied successfully in different projects have to test for KMDL® v2.0 now. Actual research in the field of KMDL® copes with skill management and staffing support for KMDL® v2.0, simulation of knowledge-intensive business processes, process-model-based configuration of knowledge management tools, interorganisational knowledge management, and semantic annotation of process models.

REFERENCES

Abran, A., Moore, J.W., Bourque, P., Dupuis, R., & Tripp, L.L. (2004). *Guide to the software engineering body of knowledge. Swebok®: A project of the software engineering coordination committee* (1st ed.). IEEE Computer Society Press.

Brown, W.J., Malveau, R.C., McCormick, H.W., & Mowbray, T.J. (1998). *AntiPatterns: Refactoring software, architectures, and projects in crisis.* John Wiley & Sons.

Davenport, T.H., & Prusak, L. (2000). *Working knowledge*. Harvard Business School Press.

Eppler, M., Seifried, P., & Röpnack, A. (1999). Improving knowledge intensive processes through an enterprise knowledge medium. In *Proceedings of the 1999 Conference on Managing Organizational Knowledge for Strategic Advantage: The Key Role of Information Technology and Personnel*, New Orleans, Louisiana.

Gamma, E., Helm, R., Johnson R., & Vlissides, J. (1995). *Design patterns.* Addison-Wesley.

Greenfield, J., Short, K., Cook, S., Kent, S., & Crupi, J. (2004). *Software factories: Assembling application with patterns, frameworks, and tools* (1st ed.). Wiley.

Groff, T.R., and Jones, T.P. (2003). *Introduction to knowledge management: KM in business.* Butterworth-Heinemann.

Gronau, N., & Weber, E. (2004). Modeling of knowledge intensive business processes with the declaration language KMDL®. In *Proceedings of the Information Resources Management Association International Conference*.

Gronau, N., & Weber, E. (2004a). Management of knowledge intensive business processes. In J. Desel, B. Pernici, & M. Weske (Eds.), *Business process management* (pp. 284-287). Heidelberg: Springer.

Gronau, N., Müller, C., & Uslar, M. (2004). The KMDL® knowledge management approach: Integrating knowledge conversions and business process modeling. In D. Karagiannis, & U. Reimer (Eds.), *Practical aspects of knowledge management* (pp. 1–11). Berlin: Springer-Verlag.

Gronau, N., Müller, C., & Korf, R. (2005). KMDL®—Capturing, analysing and improving knowledge-intensive business processes. *Journal of Computer Science, 4*, 452–472.

Hamel, G., & Prahalad, C.K. (1990). The core competence of the corporation. *Harvard Business Review 68*(3), 79-91

Heisig, P. (2002). GPW-WM: Methoden und werkzeuge zum geschäftsprozessorientierten wissensmanagement. In A. Abecker, K. Hinkelmann, & M. Heiko (Eds.), *Geschäftsprozessorientiertes wissensmanagement* (pp. 47–64). Berlin: Springer.

Inkpen, A.C., & Dinur. A. (1998). Knowledge management processes and international joint ventures. *Organization Science, 9*(4), 454–468.

Kim, S., Hwang, H. & Suh, E. (2003). A process-based approach to knowledge-flow analysis: A case study of a manufacturing firm. *Knowledge and Process Management, 10*(4), 260–276.

Kirchner, M., & Jain, P. (2004). *Pattern-oriented software architecture: Pattern for resource management.* John Wiley & Sons.

KMDL. (2006). *Knowledge modeling and description language*. Retrieved March 30, 2006, from http://www.kmdl.de

Mertins, K., Heisig, P., Vorbeck, J., and Mertens, K. (Eds.). (2000). *Knowledge management: Best practices in Europe.* Springer.

Nissen, M.E. (2002). An extended model of knowledge-flow dynamics. *Communications of the Association for Information Systems, 8*, 251–266.

Nissen, M.E., & Levitt, R.E. (2004). Agent-based modeling of knowledge dynamics. *Knowledge Management Research & Practice, 2*(3), 169–183.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation.* New York: Oxford University Press.

Polanyi, M. (1958). *Personal knowledge—Towards a post-critical philosophy.* Chicago: The University of Chicago Press.

Remus, U. (2002). *Process oriented knowledge management: Concepts and modeling.* Doctoral thesis, University of Regensburg, Germany.

Rombach, H.D., Basil, V.R., & Selbey, R.W. (1993). Experimental software engineering issues: Critical assessment and future directions. In *Proceedings of the International Workshop,* Dagstuhl Castle, Germany.

Schmidt, S.R., Kiemele, M.J., and Berdine, R.J. (1996). *Knowledge based management: Un-*

leashing the power of quality improvement. Air Academy Press.

Van der Aalst, W.M.P (1998). *Formalization and verification of event-driven process chains.* Eindhoven: Computer Science Reports.

ENDNOTE

[1] http://www.m-wise.de

This work was previously published in the International Journal of Knowledge Management, edited by M. Jennex, Volume 2, Issue 4, pp. 32-51, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 6.13
A Relative Comparison of Leading Supply Chain Management Software Packages

Zhongxian Wang
Montclair State University, USA

Ruiliang Yan
Indiana University Northwest, USA

Kimberly Hollister
Montclair State University, USA

Ruben Xing
Montclair State University, USA

ABSTRACT

Supply Chain Management (SCM) has proven to be an effective tool that aids companies in the development of competitive advantages. SCM Systems are relied on to manage warehouses, transportation, trade logistics and various other issues concerning the coordinated movement of products and services from suppliers to customers. Although in today's fast paced business environment, numerous supply chain solution tools are readily available to companies, choosing the right SCM software is not an easy task. The complexity of SCM systems creates a multifaceted issue when selecting the right software, particularly in light of the speed at which technology evolves. In this paper, we use the approach of Analytic Hierarchy Process (AHP) to determine which SCM software best meets the needs of a company. The AHP approach outlined in this paper can be easily transferred to the comparison of other SCM software packages.

INTRODUCTION

A supply chain represents the veins of a business; it is a network of facilities and distribution options that perform the functions of material procurement, the transformation of materials into intermediate and finished products, and finally the distribution of finished products to customers. Supply chains are not specific to any one industry; they are inherent in both manufacturing and service based organizations. Supply chains do however vary in complexity from industry to industry and even firm to firm. The process of managing supply chains is a multi-billion dollar software industry; the worldwide market for SCM software topped an estimated $6 billion in 2006 and is expected to reach $10 billion by 2010 (a compound annual growth rate of 8.6%) (Trebilcock, 2007).

Supply chains are evolving to meet the changing requirements of the companies trying to manage them. A few years ago simply having full visibility of your own supply chain was seen as extraordinary. Now that visibility is no longer enough; companies need to be agile in respect to their supply chain (Croom, Romano, & Giannakis, 2000; Bartels, 2006). Companies need to make educated business decisions based upon the information captured in their information systems.

SCM systems are used to coordinate the movement of products and services from suppliers to customers (including manufacturers, wholesalers, and retailers). The system's main objective is to manage warehouses, transportation, trade logistics and various other issues concerning facilities and the movement and transformation of materials en-route to customers.

The components of SCM include (but are not limited to) supply chain event management and optimization, warehouse management, radio frequency identification (RFID), transportation management, demand management, supplier relationship management, and service parts planning. Beyond the traditional elements, SCM software has also incorporated modules for international management; this is the direct result of the growing need for businesses to manage supply chains that include a mix of global suppliers, manufacturers, and company owned plants. In fact, the bursting demand for global SCM has led the upsurge in the worldwide market for SCM systems (Aksoy & Derbez, 2003; Das & Buddress, 2007; Hill, 2007).

Why Compare?

Research has found that the typical U.S. manufacturer is managing an average of more than 30 contract relationships (Trebilcock, 2007). Wholesalers are distributing to worldwide retailers and jobbers for resale; and retailers now staff virtual storefronts that service customers globally. The growing supply chain requires a management system that is efficient and caters to the needs of each enterprise. The benefits of implementing an appropriate SCM system include: Increased top-line profit growth through supplier teamwork; Reduced inventory carrying costs and stock-outs; Increased customer service; Supply chain visibility; Optimization of the value chain respective to cost reduction and bottom-line improvement; Reduced corporate-wide operating costs; Increased competitiveness; and Quick adaptation to changing markets without detriment to customers.

However, since SCM system implementation is typically not a small scale operation, there are inherent managerial risks. For example, within businesses with several facilities, partners, and departments etc., a legacy or manual SCM system can lead to bottlenecks. There are cases where the appropriate SCM application is chosen but it does not sufficiently integrate with the rest of the enterprise software applications. In some cases, the wrong SCM application is chosen (perhaps to cut costs or due to poor information); the result is that the whole business from sourcing to distribu-

tion is negatively affected. Efficient SCM provides immense benefits; a well-run value chain should positively impact an organization's profitability and success.

SUPPLY CHAIN MANAGEMENT SOFTWARE

While there are a number of SCM software providers, the major players have maintained their top positions. For example, in 2005 the top 5 ranked providers were Manhattan, RedPraire, SSA Global, Swisslog, and SAP AG (O'Neill, 2005); in 2007 the top 5 spots were manned by Manhattan, RedPraire, SAP, Oracle and Infor (who swallowed up SSA) (Trebilcock, 2007). In selecting SCM software vendors to compare for this study, the following criteria were utilized:

- Limited to those providers offering world-wide solutions
- Limited to vendors whose SCM systems include the following minimum components: Warehouse Management Systems (WMS), Transportation Management Systems (TMS) and Warehouse Control Systems (WCS),
- Limited to 7 software vendors in the study (use of Expert Choice limited us to 7 alternatives).

Based on criteria outlined above we have elected to compare the following 7 software vendors:

- Aldata – Aldata SCM
- HighJump - HighJump SCM
- Infor – Infor SCM
- Manhattan Associates – Integrated Logistics Solutions
- Oracle – Oracle E-Business Suite Supply Chain Management - R12
- RedPrairie – E2e
- SAP – SAP SCM

Decision Tool

To aid in the comparison of our selected SCM systems, we have relied on Expert Choice 11.5 (EC11.5). The key functions of EC11.5 are: to structure by identifying objectives and criteria for evaluating the decision at hand and the potential alternatives; to evaluate the objectives and alternatives; to synthesize by combining hard numbers and intuitive judgments (math and psychology) to value the alternatives via sensitivity analyses and exploring "what if" scenarios (Expert Choice Inc., 2007). By relying on EC11.5 we can understand the trade-off of weighing certain choice criteria differently.

It is possible to yield the best alternative via EC11.5 using the Analytical Hierarchical Process (AHP). AHP is based on mathematics and human psychology; the process deals with complex decision making by providing a framework for arranging the criteria, quantifying them, and relating the elements to the overall goal. The AHP method breaks down the decision into a hierarchy of more clearly stated sub-issues (where each issue is treated independently); once the hierarchy is built, the numerous alternatives are reduced to a series of pair-wise comparisons for synthesis. Those judgments are converted to numerical values that are processed, evaluated and compared over the whole scope of the issue. Because a numerical priority (weight between 0 and 1) is assigned to each element, AHP allows non-comparative elements to be compared in a consistent way. Finally, AHP produces numerical priorities and the choice of the best alternative simply becomes ranking the software packages in order of preference (Saaty & Vargas, 2006; Saaty, 1980; 1996; 2001; 2005).

In the following section, we briefly discuss features offered in each of our seven chosen soft-

ware alternatives. Based upon this information, each alternative software package will be scored with respect to our evaluation criteria; these scores form the basis for pair-wise comparisons used in the AHP.

Aldata (Aldata SCM)

Aldata is one of the global leaders in supply chain software for retail, wholesale and logistics companies. The company's comprehensive range of SCM and In-Store solutions enable more than 300 customers across 50 countries to enhance productivity, profitability, performance and competitiveness. The majority of Aldata's customers are located in Western Europe; they primarily service small and medium size supermarket chains but also provide service to larger companies including Bosch and Merck. Aldata has won the IT Europa's European IT excellence award (General Business News, 2008).

Aldata invests heavily in research and development within the SCM unit. The G.O.L.D. product family is being further developed and the current version six of the software will remain the core platform for the coming years. Major launches were the new G.O.L.D. Track modules, a federation module for providing integrated traceability across business networks, and G.O.L.D. Mobile, a module providing mobility in the retail store and enabling store operations such as stocktaking, receiving and price control using PDAs or radio frequency terminals (IHL Group, 2006). The company does not provide any other enterprise management solutions.

HighJump (HighJump SCM)

HighJump, a 3M Company, offers standard functionality but leverages best practices in order to meet the clients' immediate operational disciplines while increasing efficiencies. HighJump software highlights its vertical-specific adaptability which enables solutions to fit a variety of customer requirements in industries that include aerospace, automotive, consumer goods, direct store delivery (DSD), discrete manufacturing, food and beverage, wholesale distribution/industrial production, document management, and publishing.

HighJump implants a best-practice advantage implementation methodology which focuses on budgeting and aligning clients' interests. HighJump offers in-depth training courses aimed at preparing clients to administer their software solutions and 24/7 staffed customer support. Furthermore, HighJump organizes an annual user conference where HighJump industry analysts, employees, partners and customers meet to brainstorm the latest trends in execution; customers get the opportunity to learn how to leverage SCM solutions to achieve increased efficiencies and maintain competitive advantage.

The software architecture and hardware platforms include the following: Main Languages: C++; .net; C-sharp; DMBS; SQL Server; Oracle; and a 4GL: HighJump adaptability tool set. The software pricing ranges from $100,000 to $250,000 and is dependent on the number of concurrent users. The target market for HighJump includes logistics/distribution, batch, repetitive, job shop, discrete, process, continuous flow, and project manufacturing which translates to industries that include health care, pharmaceuticals, automotive, grocery, food, apparel, 3PL, and audio.

HighJump integrates source-to-consumption solutions that contain four critical elements including rapid return on investment, a global execution platform that allows all applicants to work together seamlessly, ease of configuration to empower strategic competitive advantage, and best practices based functionality to solve core logistic challenges

Infor (Infor SCM)

Infor is a large size software developer that provides very strong management resource solutions. The company offers its products as separate

modules for various enterprise functions, including: Manufacturing, Supply Chain Management, Financials, Project Management (PM), Human Resources, and Customer Relationship Management. Infor also offers an all-inclusive Enterprise Resource Planning Suite (ERP). Their products are implemented worldwide.

Built on Open SOA (service-oriented architecture), Infor's logistics software provides advanced customization, which is not limited to any specific platform.

Due to high levels of customization, there are high setup costs in switching to the Infor's software. Due to high setup costs, Infor has historically targeted medium and large size businesses with sales in excess of $50,000,000. Recently, Infor announced a new ERP solution targeting small to medium size distributors (ERP FACTS). In targeting smaller firms, Infor has developed numerous industry specific basic modules that do not need costly customization; this has put them in a very cost advantage position when compared with their industry competition. Infor's SCM solutions range from $2,000 to $100,000+; solutions at the higher end of the price range tend to be solutions that have been extensively customized.

Manhattan Associates (Integrated Logistics Solutions)

Manhattan Associates is a leading supply chain solutions provider. The company's supply chain planning, supply chain execution, business intelligence, and business process platform capabilities enable its more than 1200 customers worldwide to enhance profitability, performance and competitive advantage. Unlike some of the other companies that provide SCM tools in addition to other non-SCM solutions, Manhattan Associates is engaged almost exclusively in the SCM solutions field. Much of their operational results company acquisitions.

Manhattan Associates targets companies in the retail, distribution, transportation, and manufacturing industries; their modules include warehouse, transportation, trading partner, distributed order, and reverse logistics management applications. Manhattan also offers performance management and radio-frequency identification tools designed to enhance the functionality of its other products. Manhattan's "Atlanta facility lets customers evaluate technology and equipment before adding RFID to supply-chain operations" (Malykhina, 2005, P.1). The company sells third-party hardware, including bar code scanners and printers, and also provides professional services. Manhattan Associates has been expanding their operations through new product offerings. According to Trebilcock (2007), "In 2001 Manhattan was the No.1 provider of warehouse management systems, with just more than $100 million in revenue. Today, Manhattan is a nearly $300 million company, offering transportation management, supplier collaboration and supply chain planning" (P. 47). Manhattan offers customer service on a 24 hour /7 days a week basis.

Oracle (Oracle E-Business Suite Supply Chain Management - R12)

With Oracle SCM (OSCM), companies can build and operate world class value-chains for profitable growth. The Oracle E-Business Suite Supply Chain Management (R12) family of applications integrates and automates all key supply chain processes, from design, planning and procurement to manufacturing and fulfillment, providing a complete solution set to enable companies to power information-driven value chains. Companies can anticipate market requirements and risks, adapt and innovate to respond to volatile market conditions, and align operations across global networks. A unified data model provides a single, accurate view of your entire supply chain. Companies can implement lean, demand driven principles and manage their increasingly complex, global supply chains.

OSCM consists of a variety of separate applications which are categorized by supply chain segments. Depending on a company's needs, a wide variety of applications are available. Some of the basic benefits of OSCM include real time supply chain measurements as a result of a direct connection with suppliers and customers, expense management for all categories of goods and services, analytical support to monitor the performance of a company's supply and the ability to make adjustments.

Oracle has been rapidly expanding its SCM software business, primarily through the acquisition of smaller, more specialized businesses. Oracle has adopted an acquisition strategy in order to accelerate its software innovations. Previous acquisitions include People Soft and Demantra. As a result of the acquisitions, Oracle is focusing its next generation of products on integration and the ability for these programs to communicate and share information with each other.

RedPrairie (E2e TM Suite)

E2e offers customers supply chain execution, store management, logistics, and warehouse management software that can assist or manage all facets of their business. E2e allows for monitoring and control from inbound logistics and inventory management to order fulfillment and transportation. Collaboration tools are included to assist in a company's daily efforts to collaborate or interact with trading partners.

Every industry is being challenged with increasingly complex multi-channel demands, especially from the end consumer of their products. The ability to respond to create perfect, customized, and timely orders is a critical competitive advantage to meet consumer expectations, reduce inventory and storage costs, and streamline operations (Report, 2006).

RedPrairie considers themselves unique in the SCM software industry in their effort to incorporate change management, learning management, interactive training, comprehensive online help, and customized learning and reference materials into their offerings. Companies implementing their suite of tools can leverage real ROI, minimize downtime due to obstacles, and move toward near 100% efficiency which increases companies core advantage. RedPrairie's support centers are located globally and offer full language capabilities in addition to leading-edge call tracking capabilities for reliability.

RedPrairie's ability to configure their software suites into practical groupings and components allows them to offer build-to-order manufacturing solutions that include sophisticated in-line sequencing which can lead to reduced cost and increased efficiency. The result is that all component levels can be tracked, revised and/or updated keeping all elements in synchronization. This capability is enhanced when used in conjunction with RedPrairie's warehouse management system (WMS).

SAP (SAP SCM)

SAP is the world's largest business software company and the world's third – largest independent software vendor. By building the strongest technology, services and development resources, SAP is positioned to deliver a superior business platform that can access valuable information resources, while improving overall process efficiencies and strong customer relationships including end users, suppliers and vendors. SAP's integrated packages allow customers' needs to be identified quickly and precisely while comprehensive and personal solutions are developed and rolled out.

SAP's services assists companies in maximizing their success through a combination of SAP experts, methodologies, tools, and certified partners. Users of SAP SCM can benefit with the following: Faster response to changes in supply and demand that will give customers the chance to quickly capitalize on new opportunity; Increased customer satisfaction- SAP SCM enables clients

to better adapt to changes and meet customer demand; Compliance with regulatory requirements; Improved cash flow; and Higher margins- SAP SCM helps companies lower operational expenses with more timely planning for procurement, manufacturing and transportation. Using SAP SCM companies can also improve their overall performance and quality through better order, product, and execution synchronization.

SAP SCM delivers a complete set of futures and functions for building adaptive supply chain networks. SAP SCM includes features and functions to support collaborative supply chain planning processes, including strategic, tactical, and operational planning as well as service parts planning. By using SAP SCM, a company can optimize a full range of planning activities including: demand planning, safety stock planning, supply network planning, distribution planning, and supply network collaboration. The company can also handle service parts planning activities, which includes: parts demand planning, parts inventory planning, parts supply planning, parts distribution planning, parts monitoring. With SAP SCM the company can manage order fulfillment activities, support end-to-end procurement, manage key transportation processes, manage warehouse activities, support all production processes including engineer-to-order, make-to-order, and make-to-stock manufacturing. SAP SCM also supports supply chain visibility design and analytics with features and functions that enable supply chain design and analytics processes. Planners and key decision makers can perform strategic and tactical business planning.

RELATIVE COMPARISON

Criteria Revisited

In order to conduct our analysis we will use selected quality criteria to assess software characteristics. With the help of Expert Choice software, we will first compare the relative importance of each of the criteria against each other.

- Ease of Integration: the ability to integrate with any third party software platforms (vendors, government clearance computers, ocean carriers, etc.) and any other proprietary software or legacy systems.
- Reliability and Stability: any warranties provided by the vendor in addition to the degree of completeness, accuracy, and consistency of the package. The availability of any templates or custom models available for specific aspects of the supply chain.
- Efficiency: the level of accessibility and efficiency; how well the software functions are aligned with the general business objectives as well as the number of tools available.
- Customization and Expansion potential: the degree to which the product supports the specific business goal assumptions and the tools available for SCM respective to the specific needs of the client company. Also, the degree of augmentation ability and the ability to evolve over time and expand as well as any expert options or limitations.
- Service and Support: the availability of support services coupled with the time it takes to have a technician to be available on site or on the network. The availability of technicians that are specialized in the particular industry the SCM is being utilized (transportation versus warehouse management, retail versus wholesale etc.). Any extra perquisites such as annual conferences.
- Mobility and Portability: is a measure of platform independence; the number of support platforms and supported architecture as well as any software requirements needed to run the software.
- Ease of Interface: shows how well the software communicates with the outside world, the quality of human machine interface, and how results are displayed.

- Pricing: the base price of the product, and/or range of the price for "basic" packaging respective to the SCM applications.

Evaluation Model

Our evaluation criteria, as entered in Expert Choice, are as follows:

- Ease of Integration – evaluated in terms of:
  - Time
  - Number of platforms supported
  - Support for open source developers
  - Reliability and Stability – evaluated in terms of:
  - What classes of models does the application support?
  - If the application allows custom model creation, templates or both.
  - The reputation of the vendor supplying the tool
- Efficiency – measured in relation to:
  - How well the software supports the general business objectives?
  - Data processing capacity and speed.
- Customization – evaluated in terms of:
  - How well the product supports the general business goal assumption?
  - Specific tools available respective to the specific needs of the client company.
- Expansion – evaluated in terms of:
  - The degree of augment ability
  - The ability to evolve over time and expand (i.e. available upgrades)
  - Any expert options or limitations
- Service and Support – evaluated in terms of:
  - Any available demos
  - Turnaround time for on site or network tech availability
  - Specialization of the techs in the respective industry
  - Any additional perks (i.e. annual conferences, 24/7 service etc.)
- Mobility and Portability – evaluated in terms of:
  - Hardware platform
  - Software architecture
  - Software requirements
- Ease of Interface – evaluated in terms of:
  - Simplicity of human machine interface
  - Result displays
  - Graphical layout
- Price (where available) – evaluated in terms of:
  - The price range provided – the lower of the range and the mean served as the rating criteria.

PROCEDURE OF EXPERT CHOICE: SHORT EXAMPLE

Providing an example of Expert Choice on a small scale helps to describe the method behind pair-wising and making the best decision regarding which SCM Software to choose contingent on our criteria. To provide a small scale example we implemented five criteria against our objective and compared three SCM software applications.

The pair–wise weights were assigned to the criteria initially chosen; the decisions were based on available information and which criteria outweighed their pair. The result is the prioritized listing of criteria respective to the objective – in this case selecting the best SCM software. Expert Choice allows for normalization in order to better understand the weighting scheme – in other words these small scale results recommend that Pricing is more than twofold the importance of Portability. In addition, the inconsistency is very low at only 2%; the logic behind assigning weights (importance) to each criterion remained consistent within each pair-wise comparison.

Figure 1. Presents the prioritized criteria after the initial weight assignments

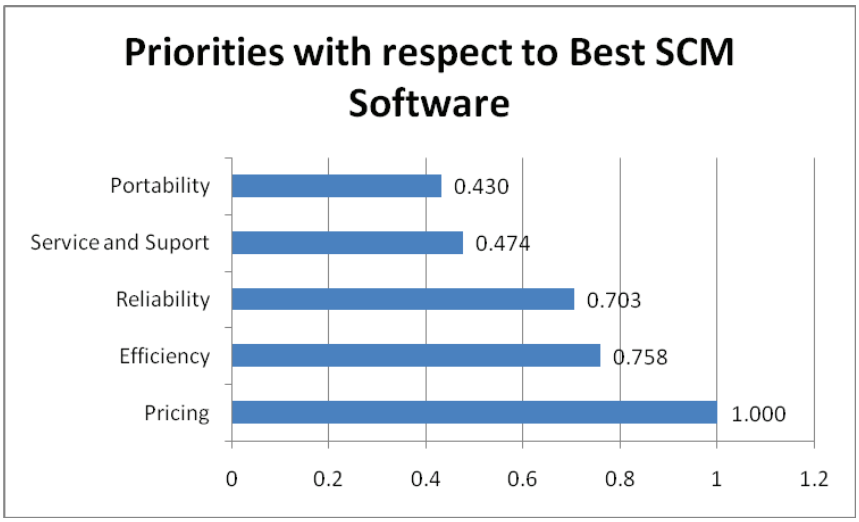
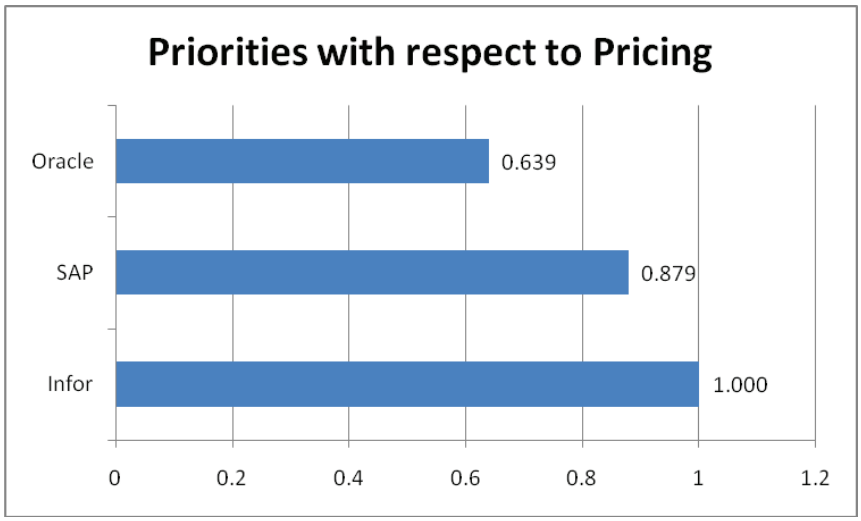


Figure 2. Presents the introduction of alternative SCM software packages analyzing pricing



We chose three alternative SCM Software packages (Infor, SAP & Oracle) for the short example to exhibit the functions of AHP relied on by Expert Choice. We proceeded to perform a pair-wise comparison of all our software solution alternatives based on each established criteria. Based on strengths and weaknesses determined about each software; we analyzed the components

of each criterion on a case by case basis. The detailed level of analysis allowed us to obtain informative results about each software solution tool. The logic provided a prioritized listing of the software packages according to the criteria which held the highest weight. As an example, Figure 2 displays results obtained from Expert Choice when analyzing Pricing criteria.

However, this is minimal information when making a decision – sensitivity analysis provides a technique for determining the outcome of a decision if a key prediction turns out to be wrong. The analysis is a tool for analyzing the impact of key criteria; sensitivity blocks are used to generate tables and/or plots of simulation results as functions of feed stream, block input, or other input variables. Since there are various criteria, the following charts demonstrate the outcome of each SCM package against the chosen criteria. Expert Choice offers a variety of alternatives which facilitates the decision making process and offers alternatives for assessing the outcomes according to user preference.

We found that the dynamic sensitivity analysis tool can prove to be very useful when trying to estimate overall impact of each criterion on the final decision. The program allows users to graphically manipulate the relative weight of each criterion against one another by simply clicking and dragging. Furthermore, the program would simultaneously change the graphically presented outcome. Thus, if for the purposes of presentation, we assigned an unrealistically high weight to the price criterion in the example above the overall outcome would change from SAP being the best option to Infor software as the ultimate solution.

SCENARIO ANALYSIS

In this section, we compare the seven chosen SCM applications based on the seven criteria previously defined. The decision of optimal software choice involves multiple-objectives and will vary among customers based upon individual needs. It is not often that one SCM application suits the expectations of every industry, institution, or customer; therefore the integration of scenarios is an important tool of the decision making process.

In order to make the simulation realistic, various scenarios were examined that altered the size, needs, and global presence (amongst other aspects) of potential customers for the available SCM applications. We proceed with the hypothetical situations and demonstrate techniques and procedures to establish the best available alternative based on our set of defined criteria. In examining the importance of various criteria, size stood out as a decision making factor. In order to emphasize the importance of size as a decision making factor, we went further to implement three specific scenarios that visit opposite ends of the spectrum; large global presence versus small regional existence. Note, however, that when the size was manipulated, only certain criteria proved to be dependent on that factor, therefore the results below exhibit how other criteria were weighted similarly, despite the variations in size.

The number of SCM applications compared coupled with the number of evaluation criteria results in a significant number of pair-wise comparisons used in the AHP process. The following table summarizes the relative weights of each criterion in addition to the direct relationship between the synthesized weights in each column with their respective criterion. The higher the synthesized weight, the more a particular sized company (Large vs. Small) views that particular SCM software alternative.

When comparing above scenarios, the notable changes were the relative weights of each criterion when the size of the company is accounted for in the scenario. It is important to take into consideration that a real business environment comprises many different industries, as well as an array of different company types with different needs, goals and business objectives; all of which

would impact relative weights and the ultimate SCM software decision.

The selection of the best software for a specific company should be based on the individual needs of the organization making the choice. The same software package will not be the best choice for every buyer. Different SCM solutions will provide the best fit depending on the applicable situation or

Table 1. Synthesized weights with respect to criteria or goal

Synthesized Weights with Respect to Criteria	Pricing		Service & Support		Reliability & Stability	
	Large	Small	Large	Small	Large	Small
Manhattan Associates	0.138	0.162	0.140	0.090	0.143	0.143
RedPrairie	0.165	0.103	0.159	0.186	0.119	0.119
SAP	0.140	0.103	0.161	0.163	0.209	0.209
Oracle	0.143	0.228	0.161	0.156	0.211	0.211
Infor	0.089	0.078	0.105	0.119	0.146	0.146
Aldata	0.114	0.051	0.090	0.065	0.071	0.071
HighJump	0.212	0.275	0.183	0.221	0.100	0.100

Customization & Expansion		Easiness of Interface		Mobility & Portability		Easiness of Integration	
Large	Small	Large	Small	Large	Small	Large	Small
0.181	0.173	0.223	0.223	0.205	0.205	0.215	0.215
0.114	0.123	0.139	0.139	0.145	0.145	0.137	0.137
0.198	0.204	0.141	0.141	0.205	0.205	0.143	0.143
0.199	0.202	0.165	0.165	0.212	0.212	0.171	0.171
0.143	0.151	0.167	0.167	0.073	0.073	0.166	0.166
0.097	0.086	0.122	0.122	0.043	0.043	0.124	0.124
0.068	0.062	0.042	0.042	0.116	0.116	0.044	0.044

Overall Synthesized Weights with Respect to Goal		
	Large	Small
Manhattan Associates	0.179	0.156
RedPrairie	0.135	0.133
SAP	0.176	0.169
Oracle	0.186	0.199
Infor	0.135	0.122
Aldata	0.094	0.070
HighJump	0.096	0.152

scenario. Creating different hypothetical scenarios can be useful in the selection process.

To illustrate this point, we have created different scenarios which demonstrate the relevance of the individual organization's environment and objectives in the selection process. In addition to size, the best SCM software package for an organization can differ based on characteristics like industry or sector, geographic diversity of operations and vertical or horizontal integration of the supply chain. We found that sector can be a crucial factor in the decision making process; we believe that an organization's sector will drive the decision for an optimal SCM software package.

In addition to a total of 144 basic pair-wise comparisons in order to compare all alternatives with respect to all of the criteria, each scenario also requires an additional 21 pair-wise comparisons. Once all pair-wise comparisons are made, Expert Choice is used to synthesize the weights of all the criteria with the weights of all the alternatives to determine the best solution for each scenario. In illustrating the impact of each scenario of the final decision, we have chosen different sets of SCM software packages for a more complete comparison.

Table 2 lists the weights of the pair-wise comparisons for government versus business entities. There is a direct relationship between the individual criterion and the weights displayed in each column. The higher the number displayed, the greater the weight placed on the criterion for that type of organization.

Table 2. Weights assigned to alternatives for both business and government use

Synthesized Weights - with respect to criteria	Portability		Reliability		Efficiency		User Friendliness	
	Business	Gov't	Business	Gov't	Business	Gov't	Business	Gov't
i2 Solutions	0.149	0.147	0.167	0.167	0.154	0.154	0.159	0.153
Logility	0.138	0.134	0.149	0.149	0.140	0.140	0.135	0.141
SYSPRO 6.0	0.114	0.116	0.085	0.085	0.112	0.112	0.138	0.138
Picaso	0.069	0.073	0.081	0.081	0.064	0.064	0.103	0.102
Manhattan Assoc	0.215	0.207	0.177	0.177	0.209	0.209	0.162	0.172
Oracle	0.238	0.182	0.201	0.201	0.251	0.251	0.154	0.131
ILOG	0.077	0.141	0.141	0.141	0.071	0.071	0.150	0.163

Synthesized Weights – Continued	Report Interpretation Simplicity		Customization Flexibility		Training & Support	
	Business	Gov't	Business	Gov't	Business	Gov't
i2 Solutions	0.164	0.164	0.153	0.153	0.143	0.143
Logility	0.117	0.117	0.165	0.167	0.153	0.153
SYSPRO 6.0	0.124	0.124	0.124	0.123	0.142	0.142
Picaso	0.097	0.097	0.105	0.098	0.118	0.118
Manhattan Assoc	0.215	0.215	0.150	0.150	0.206	0.206
Oracle	0.175	0.175	0.197	0.200	0.133	0.133
ILOG	0.107	0.107	0.106	0.110	0.105	0.105

Based on the results of the weighted criteria calculations done by Expert Choice, the top three alternatives for a business entity would be Oracle, Manhattan Associates and i2 Solutions. This was in line with our expectations. We had expected Oracle and Manhattan Associates to be prime solutions for business and government operations, since they were the software solution tools that excelled in the areas of Efficiency and Reliability.

Table 3 summarizes the overall results obtained through Expert Choice for our case scenario. We previously placed emphasis on efficiency and reliability for which the weights obtained were very close to each other when comparing the three top alternatives. However, when the rest of the criteria are considered, the weights obtained under each business entity change influencing the type of software solution that best suit each type of organization. For example: under a business entity Oracle obtained the highest weight of .202 overall, as opposed to .180 under a government entity. Picasso on the other hand, although obtained the lowest weight for both type of entities, it obtained a better rating from the government sector with a weight of .092 as opposed to .086 from the business sector.

As demonstrated by the tables previously shown above, different entities have different preferences and priorities which leads to differences in optimal software selection. The following scenarios will further support this conclusion.

Scenario 1: A&D Wholesale Distributors, Inc

Let us assume this is a mid-size distribution company that operates throughout the United States, with 550 employees and operations in 20 different states. A&D is looking for SCM software that will support a distribution intensive type of business and assist them in reducing transportation and inventory retention costs leading to increased revenue and customer satisfaction. Based on this company's goals and objectives, we decided that the criteria they would focus on would be: Customization Flexibility, they need a software solution tool that would be able to customize to support their specific needs and Efficiency, their main objectives are to reduce transportation costs and inventory retention time.

Table 3. Summary of synthesized results for government vs. business entities

Synthesized Weights -- with respect to goal	Business	Government
i2 Solutions	0.157	0.156
Logility	0.145	0.144
SYSPRO 6.0	0.113	0.117
Picaso	0.086	0.092
Manhattan Associates	0.189	0.189
Oracle	0.202	0.180
ILOG	0.109	0.121
Overall inconsistency ratio	**0.03**	**0.05**

Scenario 1.

Criteria	Weights
Efficiency	0.232
Customization Flexibility	0.228
Reliability	0.138
Report Interpretation Ease	0.126
User Friendliness	0.117
Training and Support	0.091
Portability	0.069

Alternative	Ranking
Logility	0.192
i2 Solutions	0.186
Manhattan Associates	0.156
Oracle	0.156
Syspro 6.0	0.115
ILOG	0.102
Picaso	0.093

Scenario 2: Start Up Online Company

Let us assume this is a small retail oriented start up internet company with 10 partners, no fixed location, no fixed relationship with outside parties and limited knowledge on the industry. This is a company that would need a software solution alternative that would offer them a high level of support with relation to hardware platform and software architecture, and one that would be able to provide a high level of training and support, since they are new in the industry and have a flexible SCM structure. Based on this company's needs, we decided that the criteria they would focus on would be: Portability, because they need a software solution that would support their internet based business, across different platforms and operating systems and Training and Support, because they need a software solution that will provide them with intensive training about the software as well as with aids to gain a better understanding of their flexible supply chain structure and demands.

Scenario 2.

Criteria	Weights
Portability	0.239
Training and Support	0.183
Customization Flexibility	0.176
Efficiency	0.138
Reliability	0.103
Report Interpretation Ease	0.094
User Friendliness	0.067

Alternative	Ranking
Manhattan Assoc	0.195
Oracle	0.191
I2 Solutions	0.166
Logility	0.159
Syspro 6.0	0.109
ILOG	0.095
Picaso	0.085

In the following tables, results for a number of additional scenarios are presented.

CONCLUSION

The SCM software industry is gaining an increasing amount of attention as companies try to maximize return on investment and gain a competitive edge in their markets. The increasing focus on the industry is resulting in greater investment in SCM software and fueling innovation. In order to choose the best alternative among all of the choices available, potential users must clearly identify and prioritize their needs and preferences.

Expert Choice's technology, which utilizes AHP analysis, allowed us to compare seven SCM software alternatives according to seven select criteria in order to determine which software best meets the needs of each scenario. All of the potential factors involved in the selection process must be determined by the organization making a decision on an individual basis. We expect continuous improvements and competition from

Table 4. Summary of a large scale retailer

Alternative	Total	Pricing (L:0.49)	Service/ Support (L:0.131)	Reliability/ Stability (L:0.203)	Customization/ Exapansion (L: 0.183)	Ease of Interface (L: 0.107)	Mobility/ Portability (L: 0.110)	Ease of Integration
Manhattan	0.866	0.650	0.762	0.679	0.906	1.000	0.965	1.000
RedPrairie	0.661	0.781	0.868	0.564	0.571	0.624	0.685	0.637
SAP	0.848	0.659	0.878	0.991	0.991	0.635	0.968	0.663
Oracle	0.897	0.674	0.880	1.000	1.000	0.742	1.000	0.797
Infor	0.653	0.422	0.575	0.692	0.717	0.750	0.343	0.771
Aldata	0.454	0.537	0.492	0.335	0.487	0.548	0.204	0.578
HighJump	0.463	1.000	1.000	0.473	0.343	0.188	0.548	0.204

Table 5. Summary of a regional grocery chain

Alternative	Total	Pricing (L:0.267)	Service/ Support (L:0.177)	Reliability/ Stability (L:0.238)	Customization/ Exapansion (L: 0.060)	Ease of Interface (L: 0.108)	Mobility/ Portability (L: 0.092)	Ease of Integration (0.059)
Manhattan	0.687	0.589	0.406	0.679	0.849	1.000	0.965	0.835
RedPrairie	0.585	0.374	0.842	0.564	0.605	0.624	0.685	0.607
SAP	0.743	0.376	0.738	0.991	1.000	0.635	0.968	1.000
Oracle	0.874	0.831	0.706	1.000	0.991	0.742	1.000	0.996
Infor	0.537	0.286	0.537	0.692	0.739	0.750	0.343	0.756
Aldata	0.309	0.186	0.296	0.335	0.424	0.548	0.204	0.401
HighJump	0.668	1.000	1.000	0.473	0.305	0.188	0.548	0.391

Table 6. Summary of an auto part distributor

Alternative	Total	Pricing (L:0.092)	Service/ Support (L:0.031)	Reliability/ Stability (L:0.271)	Customization/ Exapansion (L: 0.241)	Ease of Interface (L: 0.162)	Mobility/ Portability (L: 0.168)	Ease of Integration (0.034)
Manhattan	0.845	0.650	0.762	0.679	0.906	1.000	0.965	1.000
RedPrairie	0.628	0.781	0.868	0.564	0.571	0.624	0.685	0.637
SAP	0.884	0.659	0.878	0.991	0.991	0.635	0.968	0.663
Oracle	0.917	0.674	0.880	1.000	1.000	0.742	1.000	0.797
Infor	0.623	0.422	0.575	0.692	0.717	0.750	0.343	0.771
Aldata	0.416	0.537	0.492	0.335	0.487	0.548	0.204	0.578
HighJump	0.464	1.000	1.000	0.473	0.343	0.188	0.548	0.204

the companies we have examined as well as new entrants into the marketplace looking to fill niches. The natural caveat to all this software is from the human side; the software is only as good as the users who truly understand how to properly use the application. Most logistics professionals and senior level management lack the knowledge or training to fully exploit the potential of their systems (Hannon, 2005). This ties in to a recent emphasis in moving away from pure planning and focusing on the execution aspects of managing a supply chain (Parker, 2007).

Since problems, criteria, needs, alternatives and other variables will vary from one entity to the next, there is no universal solution. In order to support an optimal choice, all of the key factors in the decision process must be identified and quantified. The methods and processes relied on in our research transfer easily to the comparison of other SCM software packages. The future for SCM software solutions is endless.

REFERENCES

Adata Optimises its G.O.L.D. Supply Chain Management (SCM) Software. (2006, September 30). IT Toolbox.

Aksoy, Y., & Derbez, A. (2003). Supply chain management: despite persistent implementation problems, SCM remains a top priority for companies eager to optimize operations (2003 Software Survey). *OR/MS Today, 30*(3), 30-31.

Bartels, N. (2006). Agile more important than lean. *Manufacturing Business Technology,* May 1, 46-48.

Croom S., Romano P., & Giannakis, M. (2000). Supply chain management: an analytical framework for critical literature review. *European Journal of Purchasing & Supply Management, 6*(1), 67-83.

Das, A., & Buddress, L. (2007). Evaluating prospective e-providers: an empirical study. *The Journal of Supply Chain Management, 43*(4), 31-46.

Expert Choice Inc. (2007). Expert Choice 11.5. Retrieved July 1, 2008, from http://www.expertchoice.com/products/ec11.html

General Business News. (2008). Aldata wins at European IT excellence awards. *Manufacturing & Logistics IT*, March 7.

Hill Jr., S. (2007). The new rules for global supply chain management. *Manufacturing Business Technology*, April 1, 2007, 22.

IHL Group. (2006, June 15). Hard data, smart decisions. *The Eye on Retail Information Systems*, 11 (11), 1.

Malykhina, E. (2005). Minimize supply-chain risk. *Information Week*, February 28, 72.

O'Neill, J. (2005). Top 20 SCE suppliers. *Modern Materials Handling* (Warehousing Management Edition), 60(6), 32.

Parker, K. (2007). Current trends in supply chain management. *Manufacturing Business Technology*, September 1, 2.

Report. (2006). RedPrairie Corporation announces a supply chain revolution. *Manufacturing & Logistics IT*, May 16, 1.

Saaty, T.L. (1980). Multicriteria decision making: the Analytic Hierarchy Process, RWS Publications.

Saaty, T.L. (1996). Decision making with dependence and feedback: the Analytic Network Process, RWS Publications.

Saaty, T.L. (2001). The Analytic Network Process, 2e, RWS Publications, 4922 Ellsworth Ave., Pittsburgh, PA 15213.

Saaty, T.L. (2005). Theory and applications of the Analytic Network Process, RWS Publications, 4922 Ellsworth Ave., Pittsburgh, PA 15213.

Saaty, T.L., & Vargas, L.G. (2006). Decision making with the analytic network process: economic, political, social and technological applications with benefits, opportunities, costs and risks. New York: Springer.

Trebilcock, B. (2007). Top 20 supply chain management software providers. *Modern Materials Handling* (Warehousing Management Edition), 62(5), 47.

This work was previously published in the International Journal of Information Systems and Supply Chain Management, edited by J. Wang, Volume 2, Issue 1, pp. 81-96, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 6.14
How E-Entrepreneurs Operate in the Context of Open Source Software

Ambika Zutshi
Deakin University, Australia

Samar Zutshi
Monash University, Australia

Amrik Sohal
Monash University, Australia

ABSTRACT

The Internet has become an integral part of our everyday lives and it is often difficult to imagine how we ever functioned without it. This chapter presents experiences of two entrepreneurial companies, one of which has survived the 'dot-com bubble burst.' The chapter identifies current and future online business environments especially in light of open source software (OSS) being accepted globally. Unlike proprietary software (such as Windows), OSS comes with its internal implementation details (source code) visible both to its developers and users, along with the freedom to change and redistribute this source. The significant implications of this unique style of software distribution for e-entrepreneurs are examined. Having a flexible strategic plan; possessing management skills; providing excellent service; and having patience are some of the recommendations provided by interviewed e-entrepreneurs. When made part of the decision-making process, these recommendations would enhance current and future e-entrepreneurs in sustaining their business.

INTRODUCTION

The aim of this chapter is to explore the usage of OSS in e-entrepreneurship and to identify the attributes and skills necessary for an e-entrepre-

neur. **E-entrepreneurship** is defined as the notion which principally uses the Internet to strategically and competitively achieve vision, business goals, and objectives. **E-entrepreneurs** use the World Wide Web (WWW) to interact and complete virtual transactions both with other businesses (B2B) and their consumers/customers (B2C).

The notion of an e-entrepreneur has recently gained recognition amongst both academics and practitioners. An e-entrepreneur has many similarities with that of an 'entrepreneur,' especially with respect to the attributes and traits required to be successful. Concurrently, the major differences between the two are primarily in the resources (such as infrastructure and setup costs) required to start the business.

Over the last two decades, most businesses have experienced substantial change brought about as a result of globalisation and the Internet. Maintaining a competitive advantage to simply survive is a continued battle for many businesses. The Internet, however, has provided companies with numerous opportunities irrespective of the nature of the products and services offered to customers. Many companies now make use of the Internet and provide customers through their Web site information such as store opening hours, store locations, contact details, and listing of their products and services. However, for a majority of these businesses a large proportion of the sales revenue is still generated through activities conducted at the physical stores. One example is **Telstra**, which in addition to having nationwide physical stores also does sales and online billing (Telstra, 2004).

The number of companies performing their business activities through the Internet is increasing rapidly while still maintaining a physical store presence to enable customers to 'see and feel' their products before making a purchasing decision. Satisfying the needs of conventional customers who prefer to complete 'face-to-face' transactions is recognised by many businesses. One example being the Borders bookstores (Borders, 2004). Then, there also are companies who only have a virtual presence and complete all their advertising, marketing, and transactions through the Internet. Amazon.com (Amazon.com, 2004) is a perfect example of this type of organisation. Brand recognition, customer service, and customer satisfaction are the main ingredients for any company, whether operating solely as 'bricks and mortar,' 'online,' or a mix of the two (Mottl, 2000).

The concept of '**entrepreneurship**' has been in existence and researched by academics for some time. Due to the lack of literature in the area of e-entrepreneurship, the authors have sought guidance and direction from the entrepreneurship literature to realise the following objectives:

- Identify attributes of e-entrepreneurs.
- Identify the similarities and differences between entrepreneurship and e- entrepreneurship.
- Identify if being an e-entrepreneur is more advantageous than simply being an entrepreneur.
- Examine open source software in the context of e-entrepreneurship.

Chief executives of two e-entrepreneurial organisations were interviewed to obtain insights into the concept of e-entrepreneurship. Some of the issues explored in the interviews included:

- attributes of an e-entrepreneur,
- role played by **open source software (OSS)** in the information technology sector,
- impact of OSS on existing and future e-entrepreneurs, and
- role played (if any) by government in supporting e-entrepreneurs.

The next section presents an overview of the literature examining the dot-com crash, entrepreneurship, and open source software (OSS). This is

followed by the section that describes the research methodology used to conduct the interviews. Case studies of the two companies interviewed is then presented identifying the various aspects of being an e-entrepreneur with respect to the current technological environment including OSS. The last section presents the conclusions and future research directions.

THE DOT-COM CRASH: DID IT CHANGE THE WORLD?

The arrival of the 21st century was accompanied by the 'dot-com crash' with hundreds of companies around the world laying off thousands of employees and filing for bankruptcies. Competition amongst the remaining companies, however, has not diminished. Companies that have survived have merely "shifted their value propositions to meet (or chase) new marketplace needs" (Spiegel, 2002, p. 30). Success stories of companies that have survived and moreover thrived following the crash are no less amazing (see Anonymous, 2003; Spiegel, 2002).

A number of parties have been blamed for the 'dot-com crash' that include but are not restricted to venture capitalists, investment banks and brokerages, and the Federal Reserve Bank (Mills, 2001). Another factor that has been attributed to the dot-com crash was that the majority of these businesses were established and run by young entrepreneurs who lacked the "essential experience in planning, organising, and managing businesses" (Foster & Lin, 2003, p. 456). These arguments also have been substantiated by *The New York Times* and *The Industry Standard* studies (Infante, 2001) where lack of human resource planning has been noted as a contributing factor leading to sexual harassment and legal suits against the companies, further crippling them following the crash (see also Dvorak, 2001). Duck (2004, p.14) listed seven mistakes that resulted in the crash: "too many competitors; short-term mentality; undisciplined growth; unrealistic revenue projects; inexperienced management; underestimated costs of establishing a national brand; and lack of customer-centered focus." It is the authors' view that as opposed to the traditional models, entrepreneurs and investors alike failed to foresee long-term funds allocation and put in place contingency plans.

The '**dot-com crash**' has not meant that the Internet is no longer used for undertaking business transactions, rather it is being used more than ever before. Whether a company started using e-technologies before or after the crash, it is imperative that they offer security to their customers and avoid hackers from their Web sites. Conry-Murray (2001) and Dvorak (2001) have highlighted security issues that companies should address to protect their customers such as debugging their sites regularly, minimum use of cookies, and not putting too many advertisements on their Web sites.

Over the past decade there has been a substantial increase in the uptake of the Internet by businesses primarily as a marketing tool. The Internet has enabled even small businesses with limited resources to instantly communicate their products and services to their target markets and audiences globally. Worldwide companies are being encouraged to join this phenomenon. For instance, speakers at the Dubai Strategy Forum mentioned a number of attributes required to improve economic performance. This included the need for accepting information technologies and a "strategic structure that wipes out bureaucracy and encourages entrepreneurialism, where managers manage, innovators innovate, and the teams are rewarded for their successes" (Anonymous, 2002, p. 1).

The next section identifies the attributes of entrepreneurs and reviews the relevant literature in the area of e-entrepreneur(ship).

ENTREPRENEURSHIP AND ITS RELATIONSHIP TO E-ENTREPRENEURSHIP

The concept of entrepreneurship has been evident in economics and sociology studies since the early 18th century (Becker & Knudsen, 2004). A number of entrepreneurship definitions have been mentioned in the literature. Mulcahy (2003, p.165), while citing the *Oxford Dictionary* defines an entrepreneur as "a person who undertakes or controls a business or enterprise and bears the risk of profit or loss." Thompson and Randall (2001, p. 290) describe entrepreneurs as those individuals who "sense opportunities and take risks in the face of uncertainty to open new markets, design and develop new and improved products and processes" (see also Legge & Hindle, 1997; Kuratko & Hodgetts, 2001).

A number of traits and skills that entrepreneurs possess are cited in the literature. According to Chris Dyson, a business analyst, there are nine traits that depict a person's entrepreneurial characters. These traits include: "personality, integrity, initiative, commitment, drive and determination, directiveness, confidence, self-direction, selling, and leadership" (cited in Tams 2002, p. 399). Cherwitz and Sullivan (2002, pp. 24-25) similarly comment that an "intellectual entrepreneur" is depicted by having attributes such as "realistic and attainable vision, taking risks and seizing opportunities, using available resources to achieve the vision by using collaboration, teamwork, and innovative strategies" (see also Jablecka, 2001, p. 376). From these definitions, it can be inferred that successful entrepreneurs need to possess attributes such as vision, opportunity-seeking, leadership, and management skills.

As highlighted earlier, for the purpose of this chapter, the authors have defined e-entrepreneurship as a concept which principally uses the Internet to strategically and competitively achieve vision, business goals, and objectives. e-entrepreneurs have been defined as those individuals who use the World Wide Web (WWW) to interact and complete virtual transactions both with other businesses (B2B) and customer (B2C) (see Thompson & Stickland, 2003). E-entrepreneurs have come under focus after the dot-com crash that resulted in the closing of hundreds of businesses and thousands of people left unemployed. Practitioners, consultants, academics, and governments are investigating the causes behind this crash that left many other industries dependent on information technology crippled for months. Contingency measures are now being put in place to avoid similar crashes in the future. A study of 42 entrepreneurs based in the Greater London Business area who survived the dot-com crash was conducted during the last quarter of 2002 by the London School of Economics and Political Science (Steinberg, 2004). Using a triangulation method, the study found that businesspeople were "in the process of jointly developing a new [under]standing of what success and decision-making means via e-business networks" (Steinberg, 2004, p. 4) and, accordingly, developing coping strategies to avoid similar downfalls in the future.

One of the factors that contributed toward the demise of many e-entrepreneurial companies was the lack of human resources and communication between sellers and customers. To address such issues and provide potential e-entrepreneurs with an understanding and practicalities of the business world, many multinational organisations are now working with their prospective employees with the aim of providing them with an in-depth understanding of business operations. "Media entrepreneurship" is one such program that has been launched by Hewlett-Packard (Canada) Ltd. (Bolan, 2002). The program primarily uses Linux as being open sourced, allowing users (students) to acknowledge that there are no limitations in software development. Robert Miller, national business development manager responsible for education and healthcare at HP, commenting on the program said (Bolan, 2002, p. 19):

*The **dot-com boom**/bust saw a lot of technologically astute people become empowered with vast amounts of capital funding, but they lacked the business sense or financial management skills to fully exercise their plans. Some of them were smart enough to bring in business people that had that kind of savvy, but it was a very awkward culture mix because there were two totally different kinds of mindsets.*

Globally, companies and individuals are being encouraged to embrace the Internet as a means of developing a business advantage. For instance, the e-Business Forum Working Group D5 (WG D5) in June 2003 identified the key challenges (including those encountered in communication and policy formulation) to encourage Greek companies to enter the area of e-business (Neofotistos & Yagoulis, 2003). WG D5 consulted with a number of Greek private and public sector companies involved in e-business and provided a number of recommendations to smooth the process of conducting business through the Internet. These included being aware of issues of privacy, protection of personal information, promoting communication, and the training the e-entrepreneurs (Neofotistos & Yagoulis, 2003).

An individual's prior understanding and knowledge in business studies and cultural background affects how much new knowledge and information is required to develop a collaborative business plan. This finding was realised by Foster and Lin (2003) when exploring the impact of individual students' learning in e-business and e-commerce environments. By using cognitive perspective in the study of students from different cultural backgrounds, Busenitz and Lau (1996) found that people from some cultures produced more entrepreneurs than others (see also Thornton 1999). Similar results have been found in a recently completed study across eight countries including Australia, Slovenia, Mexico, North America, Finland, Scotland, South Africa, and Kenya (Morrison, 2000). Business plans of new ventures in New Zealand in 2000 were compared to identify the percentage of Internet usage as part of the e-entrepreneurship competition based on the Mckinsey model (McQueen, 2004). At the end of the phase two of the competition, it was found that individuals with pervious IT background, education, business experience, or personal interest had a much higher Internet component in their business plans than participants with traditional business experience such as those for the fields of accountancy, retail, entertainment, and games.

OPEN SOURCE SOFTWARE (OSS) VS. PROPRIETARY SOFTWARE

Proprietary Software Model

In the recent past, much high-profile software (including Microsoft products such as Word and Windows XP) have been distributed under a license that treats the software as a 'black box.' The software is supplied in 'compiled' or 'binary' form, meaning that a computer can read and execute it directly. However, programmers are unable to study the internals of the program. They are forbidden to understand in detail how the program works, they are not permitted to modify its working and they can redistribute neither the software in its original form, nor in any derived or modified form. Typically, a single company or an individual holds copyrights on **proprietary software** (Anonymous, 2004). These copyrights are used in conjunction with licensing agreements to deny the "freedom" or "openness" to modify and redistribute the software. "Proprietary software is software that is not free or semi-free. Its use, redistribution, or modification is prohibited, or requires you to ask for permission, or is restricted so much that you effectively can't do it freely" (FSF 2004).

From the point of view of the software vendor, the proprietary software model utilises restrictive licensing and secrecy to safeguard **intellectual**

property (IP). It is possible that the development of the software could be regarded as entrepreneurship.

However, from the point of view of an e-entrepreneur looking to leverage existing technology, proprietary software may not seem like an attractive option, since modification and redistribution of existing proprietary software is forbidden. Furthermore, providing key services related to deployed proprietary software may not be possible due to the unavailability of the internal source code. Another problem is what is commonly referred to as 'vendor lock-in.' A proprietary software vendor by definition is the only organisation with the legal capacity to improve and enhance their proprietary software products. Hence, an e-entrepreneur wishing to deploy proprietary software is "locked in" to the vendor. No other organisation or individual (including the e-entrepreneur) can provide improvements or custom modifications. For instance, Microsoft is the only organisation that can provide security updates and bug fixes for the proprietary Windows operating system. In effect, any user of Microsoft Windows faces vendor lock-in. Unless and until Microsoft decides to issue a security update or a bug fix, users must helplessly use the software in whatever condition it is in. This argument is developed further under the discussion of OSS below.

Free and Open Source Software Model

When referring to OSS, the authors have used the Open Source Initiative (OSI) definition (OSI, 2004a). OSS involves access to the underlying source code. In addition, for a license under which software distributed is to be considered open source, it must permit redistribution of the software without requiring a royalty. Redistribution must be permitted in source as well as compiled (ready-to-run) form. Modification of the software and creation of derived works must be permitted. There are some other clauses that must be satisfied for a particular software package to qualify as OSS (OSI, 2004b). However, the criteria are arguably the most fundamental and, to someone not familiar with the OSS paradigm, perhaps the most revolutionary. Many organisations and Web sites use the term "free software" (FSF 2004) whose meaning and interpretation is very similar to OSS, with 'free' implying freedom to access and modify the source as well as redistribute unmodified and modified versions. Strictly speaking, the definition of 'free software' might preclude certain software from being considered "free" even though it might be considered OSS. Since all 'free' software would be considered OSS, we will use that term for simplicity and to avoid the confusion that comes from 'free,' meaning 'at no charge.' Interestingly, while it is possible that OSS and 'free' software can be obtained for no or very little cost, e-entrepreneurs should note that it is entirely possible for OSS and 'free' software to be 'commercial' (i.e., a source of revenue). For instance, Red Hat produces an open source product called Red Hat Enterprise Linux, an open source operating system that is sold by annual subscription. Subscribed customers are entitled to receive ongoing security updates, errata fixes, and new features as they become available for the duration of their subscription.

RESEARCH METHODOLOGY

In this chapter, we have adopted the exploratory methodology (see Peil et al., 1982; Spencer, 1982) to identify the trends of how OSS has been and would impact the entrepreneurs as the usage of Internet and other technological methods to conduct business continues to increase. Conducting interviews as a method of exploratory research has been accepted in academia. For instance, Murray (1996) used case study methodology to identify the role of venture capital investments in newly established technological firms. Conduct-

ing interviews as a research methodology offers a numbers of advantages: giving flexibility to both interviewers and interviewees in setting up a mutual time; increasing the interviewers' control on the direction of the questions and an opportunity to further explore issues; providing undivided attention of the interviewees; and, last but not the least, providing insight into non-verbal observations such as body language (see May, 1993; Burns, 1998; Peil et al., 1982; Spencer, 1982; Reddy, 1987; McNiff, 1988; Yin, 1994).

As previously mentioned, e-entrepreneurship is a new and under-researched area, hence, the authors were working in unfamiliar terrains. Case study as a research methodology has been accepted when attempting to overcome the uncertainty of having clear measuring instrument (see Wallace, 1984; McCutcheon & Meredith, 1993; McGuire, 1995; Palmer & France, 1999; Corbett & Cutler, 2000).

Chief executives from two entrepreneurial organisations were interviewed in September 2004 for their experiences of setting up, running, and maintaining their businesses in light of growing technological changes. According to the Australian Bureau of Statistics (ABS) classification, Company A can be classified as "micro" with only four employees, while Company B can be classified as "small" with 25 full-time employees (see Steinberg, 2004). The focus of the interviews was on the role of OSS in today's entrepreneurial world where considerable focus is being placed on functions of the Internet for completing business transactions. After receiving consent from the interviewees, the interviews were tape-recoded and subsequently transcribed and written up as case studies. These were then sent back to the interviewees for verification of the content, and any changes as required, were accordingly made. This step was undertaken to reduce limitations (e.g., generalisation, reliability, information overload, validity, rigour) accompanied by the case study methodology (see McNiff, 1988; McGuire, 1995; Burns, 1998; Kitazawa & Sarkis, 2000).

Please note that to protect the confidentiality of the interviewees and their respective organisations, their names have not been disclosed and are referred to here as Company A and Company B, respectively. Nonetheless, as far as possible, direct quotes from the interviewees have been incorporated in the following sections.

CASE STUDY FINDINGS

Company A

With its head office currently based in regional New South Wales (NSW), Australia, the company was established and registered as a partnership business in early 2003 and then become a proprietary limited company in January 2004. The company is "focused on developing and deploying Web commerce and Linux-based network solutions" (Company A Web site) and has successfully secured and completed projects in both the open source area and commercial world projects for both Australian and foreign-based companies including in the UK and the United States. The company's open source content management product has been ranked in the top 2% of the active projects at the SourceForge dot-net site which has over 8,000 projects and downloads listed on its Web site (Company A Web site). Even though the company and its members have a strong background and focus on Linux, it also provides software solutions for pocket PCs, the Palm Operating System (Palm OS), and Microsoft Outlook.

The mission and vision of Company A, in addition to generating and increasing its revenue, is to move toward the area of "embedded media." Interviewee A considers embedded media to "employ devices and solutions on single chip computers running on open source software." Each of the Company A directors have expertise in areas of programming, administration, and management, respectively, and are on the path of expanding the company.

The motivation and flexibility offered by working for oneself was one of the driving forces for the interviewee to establish his own company. The interviewee also wanted to have the flexibility to adjust quickly as changes in the external environment and technology took place without going through the bureaucratic levels often found in a large organisation. Technology itself is also the passion of the company. This is still a motivating factor for all the personnel involved, which is steadily pushing the company forward. The interviewee believes that this is true of other companies such as Adobe and Apple, where he feels that the vision of the company and the passion of its technologists had kept them going despite management changes.

One of the themes that intrigued the interviewers was how the concept of OSS that involves freely distributing your knowledge can result in generating business for the company. Under an open source license the 'source code' is distributed along with the ready-to-run version of the software product. The interviewers were keen to ascertain how this apparent giving away of intellectual capital could result in profit for the person/organisation involved. It appears that OSS is gaining momentum and acceptance around the world, and these issues are becoming more relevant, especially for e-entrepreneurs.

To answer the query, interviewee A commented that the writers of a program are generally accepted as having the authoritative knowledge. To elucidate his point he gave the following example: If a program is released as OSS, the writer not only shares, but also demonstrates, his or her knowledge in a manner that can be subject to scrutiny by experts. In addition, other organisations that require tailoring of the program to their specific needs may contact the writer to do the customisation for them.

This is where dollars come into the picture. The interviewee has had similar experiences. A London-based company contacted the interviewee when they wanted him to make changes to their program source code so that it was compatible with the company's accounting system. Since the company's experts had the source code of the product available to them, they could, in theory, do the customisation themselves. However, this would involve them first becoming familiar with the internal details of the software and then modifying it. Cost-benefit analysis by the company showed that it was easier and more economical for them to ask the interviewee to utilise his knowledge and expertise to deliver the modified code. The interviewee estimates that the work took him approximately 20 hours to complete while his customers might have had to spend several man-days to achieve the same result. So, the interviewee was able to acquire highly specialised, lucrative business without having invested in marketing or publicity services. The client, on the other hand, was able to procure a software system that fitted their needs in less time and for less money than if they had done it by themselves. So, it was a win-win situation for both parties involved.

In interviewee A's view, the Internet, due to its ubiquitousness and near universal accessibility, can be very effectively used as a marketing medium and MySQL AB, the popular open source database product vendor, is a classic example. In less than a decade, the MySQL database server has become internationally recognised and widely used, including in customised forms. High-profile clients include Sony, Suzuki, and Sabre Holdings (MySQL, 2004).

It should be noted that not all the software produced by Company A is OSS. Some software is released under the "general public license" (GPL) (Derekgnu, 2004) and qualifies as OSS. In other cases, clients may purchase software under a "commercial licence agreement" from Company A. This agreement allows the client to use the product and to view the source code and covers the provision of regular service by Company A such as providing further customisation and enhancements. Under this license, the clients are

not allowed to modify the source. Essentially, this is Company A's strategy to be able to effectively support their clients. If too many modifications are made to the code, Company A would have to extensively study the modified version before being able to provide enhancements. It also can be seen as a precaution taken by Company A to avoid legal repercussions arising from claims of failing to provide adequate support as per the license agreement of the customised program. However, if the buying organisation changes the source code without obtaining prior consent from Company A, the latter is under no legal obligation to be able to support the changed version of the code. Of course, the client is free to approach Company A and/or other software solution providers to collaborate on customised versions subject to additional costs.

During discussions with Interviewee A, an interesting point emerged: Company A does produce open source software but also utilises open source software tools. Company A has obtained commercial services from Red Hat Linux related to their open source Linux-based operating system. They also are developing some software for embedded systems which may turn out to be a derived and open modification of existing open source software.

If an individual is contemplating to become an entrepreneur or change himself or herself from an entrepreneur to an e-entrepreneur, they have to first consider a number of alternatives and subsequently take appropriate decisions. One needs to decide whether they would be deploying new software or leverage the existing software. Further considerations regarding licensing agreements (OSS or proprietary or a mix of both) also would be required.

In views of Interviewee A, "Open source [should be considered] as a serious alternative for people [who] are trying to do [something new]. Statistically, more than 50% of the Web servers in the world run open source software, which is generally […] Apache, […] the most popular Web server in the world.[1]MySQL is the best or the most popular database for Web-based projects." Company A does still utilise proprietary software, such as MYOB for its accounting needs since it helps them conform to the appropriate standards and legislation. MYOB runs on the proprietary Microsoft Windows operating system. Except in instances where the clients request that supplied programs remain closed source, Company A generally licenses its software as OSS and believes that other organisations should do the same.

Interviewee A commented that the decision whether or not to go OSS for their software is a business decision and dependent on its vision, current position in the market, current/existing new code development, and future plans. One needs to keep in mind that like any other material product, software and code have their own life cycle and the business decision should incorporate the potential life of the code, accordingly.

Interviewee A also made two points of direct relevance to e-entrepreneurship. First, as an e-entrepreneur, if you are trying to develop a novel solution, you can focus on the entrepreneurial aspects by using existing, reliable, open source software to avoid "re-inventing the wheel." Second, as a provider of innovative IT solutions, an e-entrepreneur faces a more level playing field since organisations are not "locked-in." Hence, they can turn to the e-entrepreneurs to provide support, maintenance, and enhancement of OSS.

The interviewers also were interested to know the support, if any, provided by the government to Company A and whether being based in a regional area it was eligible for any specific government funds. Interviewee A indicated that he had approached the state government for assistance and there had been some progress. The response, however, has not always been very speedy which sometimes is a challenge for small, struggling firms looking for assistance as they may not be operating after a few months. The problem is sometimes further compounded by the bureaucratic structure of the governments.

The difficulty experienced by regionally based organisations is convincing the officials of their innovative ideas who are sometime reluctant to provide capital for new ideas that may be regarded as being too risky. A classic example is trying to get funds for OSS projects as the question raised by government officials is the same as the authors: How can one make money by giving away their knowledge and expertise?

Company B

The company has been providing innovative, competitive solutions based on open systems and open source technology to its customers since the late 1980s. The company aims to "develop strong, ongoing relationships with its clients and long-term partnerships, based on mutual growth and respect with industry vendors" (Company B Web site). Services provided by the company fulfil customers' needs in areas of: consulting; application development; and training in software programs such as Unix, Linux, Windows systems administration and network management, and Web-based solutions to name a few. In addition to serving a number of small and medium-sized Australian-based customers, Company B also has successfully completed projects and provided training to a number of large organisations including Hitachi, Telecom Australia, Kodak Australasia, University of Melbourne, Mobil Oil Australia, CSIRO, RACV Insurance, Rockwell Areospace, ANZ Bank, Ericsson Data Australia, and VDO Instruments (Company B Web site).

When the interviewee first started working in the computer industry, not only was the industry in its infancy with huge-sized computers, a much smaller percentage of people had access to computers as compared to today. The majority of people involved in the industry at the time were young males generally categorised as 'geeks.' Only large professional organisations such as insurance companies and banks were using computers. The interviewee's introduction to the potential for online collaboration and the spirit of OSS occurred in the late 1980s. At the time, only a small team of professionals had access to the Internet. He recalls participating in an online newsgroup where he would ask questions about the C++ programming language and on occasion receiving advice from Bjarne Stroustrup — the creator of C++. However, the state of the technology at the time meant that only technically skilled people could take advantage of this online community and near-instantaneous communication. Interviewee B realised that there was a "great business opportunity" in this area if people at large could access the Internet using tools that they could learn to work with relatively easily. Unfortunately, initial feasibility studies indicated that the level of capitalisation available was not sufficient to fund the infrastructure needed to realise such an opportunity. The way to make an entry into the field was by doing consulting work based on the emerging Internet technologies and the related open standards and software.

One of the areas in which Company B has competitive advantage is in the area of OSS as it was one of the pioneering companies. The company also has a very high reputation in providing superior client service and catering to clients' specific needs. Hence, the company receives many of its projects through referrals as has happened in one of its recent projects when an Australian University on recommendation from another university contacted the company to tailor its student database to comply with the federal government's reporting guidelines by using the ERP system. In this instance, the company made use of existing codes from "open source framework called Open for Business," along with their expertise in programming to successfully complete the project in less than half of the time and cost than if the company had to write the source code from scratch. By using existing codes on the OSS, the company can reduce the price of their products and accordingly is more competitive than its counterparts. By having

access to codes and research and development (R&D) at their disposal, the company also is able to provide prompt service as compared to other large software companies who may not have their respective service offices in Australia. The company does not bind its clients into a lifetime contract and the latter have full access to their codes that they can decide to move to another vendor/company if they wished without being penalised/disadvantaged in any way.

Working toward the "betterment of the mankind" by sharing his knowledge and expertise with others while operating in an exciting, dynamic sector are the motivations for Interviewee B to remain as an e-entrepreneur. One of the challenges encountered by the company and others in the information technology sector is when trying to market their products to third parties and businesses. It has been noted that most technology experts do not have marketing and business skills that can often disadvantage them in the marketplace.

Interviewee B and his company had different experiences while interacting with the government sector. At the time of the interview, Interviewee B had been working with the federal government to create a document/database that would provide access to all government "agencies on the procurement of open source software." The document would explain legal ramifications if the third party decides to take up the OSS modules from the document. The database also would act as a networking site for individuals and organisations who wish to safely use OSS modules. The federal government is consequently working to "remove impediments towards the adoption of open source." At the state government level, the focus is still at industry development. The New South Wales (NSW) government recently announced a US$40m Linux project which is one of the largest in the world.

Company B had been in operation long before the dot-com crash, and the authors were interested in understanding how the company had survived it as opposed to many other unfortunate competitors. Interviewee B noted that unlike other new companies emerging at the time with hundreds of people being employed in the company within weeks, the number of employees had remained more or less the same in Company B. Many people contemplating to expand their wealth also had invested huge funds in their newly established companies. Company B, however, did not receive any such funds. This does not imply that Company B's products and services were any less reliable or competitive. Nonetheless, its experience had cautioned them against investing or accepting impulsive projects and funds alike. Thinking and operating strategically as well as employing experienced staff saved the company while other businesses vanished within days after the dot-com crash. In an attempt to capture the already saturated market, new information technology companies spent huge amount of resources and was another reason for their failing: not conducting sufficient market and competitive analysis, a prerequisite for establishing and running any type of business.

Interviewee B cautioned existing and new entrepreneurs of being aware of globally existing patents for various programs and software codes as even without their knowledge the programmer could be held liable for potentially plagiarising other patented softwares. He proposed that for emerging economies and businesses to be successful, it was essential that the software patent system be either made redundant or more flexible with clear guidelines with a database for searching all the existing patents.

When asked about the future of e-entrepreneurship, Interviewee B commented that this was going to expand in the coming years. To emphasise his point, he gave the example of the music industry. Until very recently, popularity in the music industry was gained by singing face-to-face to a wider audience and generally it took years to get a reputation and make money. In this current era, however, by using the technology

and the Internet, the singer can make hundreds of copies of the music on CD and simultaneously distribute it worldwide capturing the global music market. This would not have been possible using the traditional manufacturing and distribution system.

Interviewee B strongly believes that for existing and future e-entrepreneurs and information technology companies it is essential that laws relating to patents should be changed, otherwise the progress could come to a standstill. Entrepreneurs also need to be aware and cautious of the situation and take comprehensive legal consultation and protection.

DISCUSSION

For an e-entrepreneur, the software tools used are likely to be the enabling factor of the novel service being provided. In fact, the entrepreneurial product may be software or a combination of hardware equipment and software. Given that such is the case, how should various entrepreneurs decide whether to use software solutions and/or which model to use for development?

To become a successful entrepreneur, it is essential that a person learns from the experience of others and avoids making the same mistakes. The reoccurring themes within the literature and interviewees complement each other. Halloran (1991), for example, discussed the 20 commonly experienced pitfalls which should be avoided, including: having unrealistic expectations; shortsighted financing arrangements; missing the target market; buying costly and ineffective advertising; and inconsistent and chaotic management.

Explaining the similarities and differences between an entrepreneur and an e-entrepreneur, Interviewee B viewed that both have similar attributes and skills. Both need to be able to "visualise future potential [that is] above and beyond just the vision for making money." One major difference between the two is that while working in the information technology sector, an e-entrepreneur requires comparatively less funds and infrastructure when starting a business and, consequently, less total investment dollars. Once a comprehensive market and competitor analysis has been undertaken and the service that would be delivered has been finalised, only access to the Internet is required to start the business, which can be done from any location.

Andal and Yip (2002) postulate that companies should combine traditional and new-economy bases of competitive advantage into their business models in order to be successful in e-business. The generally accepted "e-bases" (Andal & Yip, 2002, p.1) include community effects, first mover advantage, fulfilment/delivery, technology, teamwork, and scalability. They also suggest that some e-business start-ups failed to implement these advantages effectively or found that they needed to be augmented with traditional bases of competitive advantage. For instance, the e-base first mover advantage should be combined with traditional product/service advantages. Getting to the market first with a novel product or service can result in significant benefits such as in the case of Amazon.com and Yahoo. Also, while the use of new and emerging technologies is considered an e-base of competitive advantage, realistically, most technology can be easily replicated. Despite this, some companies, notably Google, have been able to convert technology into an asset and sell it.

The interviewees' comments indicate that they are at least intuitively aware of such implications. Both Interviewees A and B perceived a business opportunity in connection with an emerging technology, namely, embedded devices and the Internet, respectively. At the same time, they also realised that over-committing themselves merely on the basis of new technology did not make business sense, and they relied on other sources of revenue such as consulting work and Web development to acquire the infrastructure and capital to develop their e-entrepreneurial ideas.

Interviewee B mentioned that the fact that they were the pioneers in the industry of open source solutions was a major source of competitive advantage, thus, underscoring the first-mover e-base of competitive advantage. However, Interviewee B regards their use of OSS as another — and perhaps less traditional — source of competitive advantage. By candidly disclosing to their clients the fact that a solution is based on open source software, the clients are reassured that they can, should the need arise, go to other vendors for maintenance, support, and development. There is also an undercurrent of transparency at work; when a company agrees to provide an OSS solution, their entire system is potentially subject to scrutiny by their clients. This may give the clients a sense of confidence; a vendor supplying a completely open solution that can be verified by independent technical staff must surely believe in the technical quality of their product.

A possible interpretation of the comparative ease with which certain technological functionality can be replicated is that the intrinsic value of the software that provides such functionality does not amount to much. In cases like these, OSS offers the opportunity for an e-entrepreneur to focus on services that are enabled by or based on technology rather than wasting resources developing technology which will soon be replicated and widely available anyway. Certainly, it is still possible to try and sell technology, as Google has done. But this involves ensuring that one's technology is constantly evolving at a rapid enough pace to consistently stay ahead. As pointed out by Interviewee B, such research and development (R&D) can be prohibitively expensive for e-entrepreneurs, particularly in the Australian market where capitalisation can be harder to come by than, say, in the United States.

Teamwork amongst a diverse mix of people with varied skill sets and experience is another commonly cited e-base of competitive advantage (Andal & Yip, 2002). Apart from the contributions from team members within the organization, making software available in open source form allows participation from the wider community. One of Interviewee A's open source projects has built up a virtual community of users, some of whom are able to contribute by asking questions and reporting errors that enabled Company A to enhance the quality of their product. In some cases, they are even able to offer "patches" — snippets of software code that add functionality or repair an error. Interviewee B also is aware of this effect and mentioned that Company B is an organization that tries to contribute its expertise and knowledge to the improvement and enhancement of OSS that they deploy. Interviewee B considers the process a way of "bartering IP." In this sense, releasing software developed by an e-entrepreneur as OSS is not giving away something at no charge, it is an offer to exchange and share expertise, knowledge, and time with the possibility of mutual benefit to the developer(s) of the software and the wider community. Successfully trading IP with the global community is potentially a very powerful way of harnessing the synergy arising from a team of diverse backgrounds and abilities.

Based on the understanding developed from the experiences of the interviewees, the authors have identified the following three key requirements for being a successful e-entrepreneur in the field of OSS.

1. **Being Technically Competent**
   Both interviewees recognised the critical importance of technical ability. Interviewee B mentioned the depth of knowledge required and the "wizards" on Company B's staff. Interviewee A also is emphatic on the need to be "technically sound." By definition, e-entrepreneurial activities are strongly dependent on the underlying technology. The e-entrepreneur must not only be thoroughly familiar with the state of the art of the relevant technology but also possess a deep understanding of the underpinning principles in order to be able

to analyse trends and foresee opportunities. Interviewee B does caution that in the context of Company B, high quality technical ability is often found in people who are unable to liase well with customers, and it can be a challenge to find staff that strike the right balance between being "tech" and "suit." Hence, the latter can act as a marketing challenge when "tech experts" need to explain their product in layman's language to their customers.

2. **Taking the Customer Service Perspective**

   Interviewee A, while emphasising technical ability of the product, insisted that the focus should be on what the technology can do for the consumer. Ideally, the technology should be transparent and the customer should see the benefits of the technology without needing to understand the details. In many cases, they should not even have to care whether the solution is open source or not. What should be clear to the consumer is what the technology can enable them to do and what the e-entrepreneur can make possible for them via the services related to the technological product. These views hold for Interviewee A's e-entrepreneurship plans in embedded media – small, portable devices which must, by their very nature, be consumer specific. They are also relevant to Company A's online content management system products. The base product itself is available to everyone, but the true source of revenue comes from consumers wanting services based on the existing product. These services include maintaining the customers' online presence and customisation of the base product to deal with customer-specific requirements.

   Interestingly, while Company B operates in a slightly different arena, the customer service and technology transparency issues are the ones that they strongly identify with. For instance, they have a product called the small business server. This is meant to be a turnkey solution that can be set up quickly and easily. It provides small businesses the most commonly needed functionality such as Internet connection sharing and acceleration, e-mail, anti-virus, fire walling, and file and printer sharing. It so happens that the software installed on the server is all OSS. However, in Interviewee B's experience, the customer does not necessarily care — or need to know — that this is the case, as long as they are instructed on how to use and administer it. Further, Interviewee B asserts that the open source nature of the software in this product ensures that they have full access and complete control over all aspects of the software functionality, thus placing them in a position to provide maintenance and service as long as the customer is willing to pay for such services.

3. **Being Clear on the Reasons for Going OSS**

   Neither interviewee recommends OSS as a panacea. It is clear from both their interviews that a number of factors influence their choice of whether a solution is made OSS or not. In fact, while doing some consulting work, Interviewee B recalls being specifically asked to provide proprietary software-based solutions, which Company B was comfortable providing. Hence e-entrepreneurs should not perceive OSS as the next "bandwagon" or something to be done purely out of ideological reasons. The interviewers, as well as some existing literature (see Cusumano, 2004), caution against this. Still, there can be solid business reasons for focussing on OSS as evidenced by the activities of big business such as IBM, Sun, Red Hat, and Novell (Mahoney & Naughton, 2004). In fact, both Companies A and B produce or have produced offerings and services based on proprietary and

open source software. One or more of the following reasons have been compelling enough for both Companies A and B to go open source:

- *To harness the distribution and marketing power of the Internet* Interviewee A decided to release Company A's content management system under an open source license over the Internet. The idea was to make it easy and obligation free for prospective customers to download and use the product. If they were technically inclined, they also could inspect the source and assure themselves that the product was technically sound. While there are a large number of people who have chosen to use this for free, they have at least become aware of the existence of the product and Company A. Further, some of the users have requested services and support for which they have paid Company A. Interviewee B made the observation that in order to get commercial entities to try out one's software, the fact that it is open source gives them further incentive. This is because a potential customer is ensured that they can make some use of the software even if the original vendor is not readily available because they have the code and can modify it to suit their purposes, if the need arises.
- *Avoid re-inventing the wheel.* When the functionality of the product and the services are paramount (such as Company B's turnkey product), the software itself is the means to an end. Therefore, it makes sense for an e-entrepreneur to make use of the readily available OSS rather than having to devote valuable resources to rebuild what has already been done (and often done well). For e-entrepreneurs seeking to move quickly and offer novel services, this can be a major motivation. Interviewee B finds that by avoiding a lot of duplicated R&D, they are able to provide cost-effective solutions.
- *Interaction with the community.* Interviewee A acknowledges that Company A has indeed benefited from the questions, suggestions, and contributions of the online community that uses their open source content management product. Interviewee B views his Company B's building solutions based on existing OSS as bartering IP. Company B benefits from the IP of the developers of existing OSS and in turn feeds back expertise to these open source projects.
- *Get an edge over proprietary software vendors.* An e-entrepreneur may have an idea for a new product or service that can be enabled by a software package (or, indeed, the product may be a software package itself). It is often the case that the e-entrepreneur would struggle against the big businesses that offer similar products/services based on proprietary software. By releasing their product as OSS, the e-entrepreneur can get the attention of some potential customers who are deterred by the higher prices or the closed nature of the proprietary vendors. These potential customers could become a source of revenue based on custom modifications and other support-related activities. Of course, if there are already a number of OSS solutions available, then the e-entrepreneur should try to come up with a different idea.

Apart from the discussed business reasons, a strong ethical undercurrent did seem to underlie some of the issues outlined by the interviewees. For instance, Interviewee A saw releasing a proprietary product as OSS after it had reached its end of life as one way of letting customers know that they were not being left in the lurch. By granting access to software that Company A had previously developed under a proprietary license, users of that software would be able to continue to use and maintain the product well after Company A declared it as discontinued, if they so wish. For Interviewee B trying to "make the world a better place" is more important than "making a buck."

Another common factor is the passion for technology and the excitement that comes from developing new technology or watching the technology evolve by following and perhaps collaborating with the open source software community.

As a result of these findings, we agree with Mahoney and Naughton (2004) when they say that for some companies, OSS can be a strategically valuable weapon. However, the idealistic tendencies of both interviewees would cause us to stop short of agreeing completely with them when they say that it is difficult to find the "ideals of freedom, volunteerism, and a shared community of values in today's world of Monetized Open Source."

CONCLUSION

In this chapter, we have examined the increasing usage and growing acceptance of open source software within the technological world. E-entrepreneurship is a growing field and the experiences of two e-entrepreneurs trying to survive in this competitive field were presented. The underlying attributes and skills necessary for an e-entrepreneur are very similar to that of becoming an entrepreneur. These include: being a visionary; the ability to develop short- and long-term strategic plans; providing leadership; developing flexible structures; and remaining responsive to changing environmental and market demands.

Presented next are the recommendations we have elicited from the interviewees that can enable e-entrepreneurs to be successful in their ventures.

Flexibility in Strategic Planning and the Work Environment

There is a need for maintaining flexibility when doing business irrespective of the organisational size. Especially in this technologically-dominated business world, the organisation needs to have a flexible structure so as to be able to respond to the ever-dynamic and ever-changing environment. At the same time, long-term strategic decisions should be made which reinforce the vision of the company.

Provision of High Levels of Service

A high emphasis needs to be placed on providing regular and outstanding service to clients. A company's reputation (communicated though "word-of-mouth") plays a major part in obtaining repeat business from existing clients and attracting new clients.

Developing Basic Management Skills

A successful e-entrepreneur must acquire basic management skills and attributes such as leadership, negotiation, and business planning. Furthermore, a balance needs to be maintained between the technical demands and the business demands of the company, especially those relating to people management — customers, suppliers, and employees. Motivating employees will remain a key task for managers regardless of the type of organization.

Taking the Long-Term Perspective

Establishing a new business requires significant commitment in terms of effort and financial resources over a significant period of time. Hence, returns in the short-term should not be the motivating factor. Building a robust and stable business requires patience. One way of maintaining motivation over a long period is to ensure that all individuals involved keep an open mind and enjoy the journey that can provide numerous challenges and highly satisfying outcomes.

Listening to Technologists

In order to maintain a competitive advantage, it is imperative that managers regularly communicate with their technical personnel since they are ones who will have firsthand knowledge of what is happening in the technological world.

This chapter has contributed to our understanding of OSS and e-entrepreneurship. The literature highlights the need for further research in this area, particularly to do with small businesses with Internet usage (Steinberg, 2003). Gaps in the existing literature in the area of OSS and e-entrepreneurship needs to be filled with more studies. One way this could be initiated is by more qualitative studies incorporating both in-depth case studies and focus-group discussions exploring experiences of e-entrepreneurs in the current technological environment. The experiences of entrepreneurs who have now become e-entrepreneurs also need to be further explored.

REFERENCES

Amazon.com (2004). *Amazon.com*. Retrieved October 28, 2004, from www.amazon.com/exec/obidos/subst/home

Andal, A.A., & Yip, S. (2002). Advantage amnesia. *Business Strategy Review, 13*(1), 1-11.

Anonymous (2002). eEntrepreneurship key to growth. Dubai, UAE, *Khaleej Times Online*. Retrieved August 16, 2004, from www.khaleejtims.co.ae/ktarchive/291003/finance.htm

Anonymous (2003a). Class of 2000: Dot-com IPOs learn tough lesson in survival. *USA Today.*

Anonymous (2003b). *The ten rules for e-entrepreneurship in the countries of South-Eastern Europe*. Athens, Greece, Work group D5, e-Business Forum.

Anonymous (2004). *Proprietary software*. Retrieved October 6, 2004, from http://en.wikipedia.org/wiki/proprietary-software

Becker, M.C., & Knudsen, T. (2004, June 14-16). *The role of entrepreneurship in economic and technological development: The contribution of Schumpeter to understanding entrepreneurship.* Paper presented at the DRUIID Summer Conference 2004: Industrial Dynamics, Innovation and Development, Elsinore, Denmark.

Bolan, S. (2002). Artists need business know-how: HP, OnTarget partner for new media entrepreneurship. *Computing Canada, 28,* 19.

Borders (2004). *Borders Australia: Books Music Movies Cafe*. Australia. Retrieved October 28, 2004, from www.bordersstores.com

Burns, R.B. (1998). *Introduction to research methods.* Australia: Addison Wesley Longman, Australia Pty Ltd.

Busenitz, L.W., & Lau C. (1996). A cross-cultural cognitive model of new venture creation. *Entrepreneurship: Theory and Practice, 20*(4), 25-40.

Cherwitz, R.A., & Sullivan, C.A. (2002). Intellectual entrepreneurship: A vision for graduate education. *Change, 34*(6), 22-27.

Conry-Murray, A. (2001). Keeping IT alive through the dot-com bust: Ten types of hacks. *Network Magazine, 16,* 38.

Corbett, L.M., & Cutler, D.J. (2000). Environmental management systems in the New Zealand plastics industry. *International Journal of Operations & Production Management, 20*(2), 204-224.

Cusumano, M.A. (2004). Reflections on free and open source software. *Communications of the Association for Computing Machinery, 47*(10), 25-27.

Derekgnu (2004). *The GNU General Public License*. Retrieved November 8, 2004, from www.gnu.org/copyleft/gpl.htm

Duck, P.M. (2004). The dot-com boom: Was it really all wrong?" *Behavioral Health Management, 24*(2), 14-15.

Dvorak, J.C. (2001). Dot-com season of the witch. *PC Magazine, 20*, 67.

Foster, J., & Lin, A. (2003). Individual differences in learning entrepreneurship and their implications for Web-based instruction in e-business and e-commerce. *British Journal of Educational Technology, 34*(4), 455-465.

FSF (2004). *Categories of free and non-free software*. Retrieved October 6, 2004, from www.gnu.org/philosophy/categories.htm

Gannon, J.C. (2000). Strategic use of open source information: A corporate strategy that leverage the best practices. *Vital Speeches of the Day, 67*, 153-157.

Halloran, J.W. (1991). *Why entrepreneurs fail: Avoid the 20 fatal pitfalls of planning your business.* USA: Liberty Hall Press (an imprint of McGraw-Hill).

Infante, V.D. (2001). How dot-coms learned to value the tried and true. *Workforce, 80*, 15.

Jablecka, J. (2001). Entrepreneurship, innovation and quality: The successful strategy of a newly established institution: The example of Wyzsza Szkola Biznesu-National Louis University in Nowy Sacz. *Higher Education in Europe, 26*(3), 367-379.

Kitazawa, S., & Sarkis, J. (2000). The relationship between ISO 14001 and continuous source reduction program. *International Journal of Operations & Production Management, 20*(2), 225-248.

Kuratko, D.F., & Hodgetts, R.M. (2001). *Entrepreneurship a contemporary approach* (5th ed.). South-Western (a division of Thomas Learning).

Legge, J., & Hindle, K. (1997). *Entrepreneurship: How innovators create the future*. Australia: Macmillan Education Australia Pty Ltd, Supported by the Department of Industry, Science & Tourism.

Mahoney, I.G., & Naughton, E.J. (2004). Open source software monetized: Out of the bazaar and into the big business. *The Computer and Internet Lawyer, 21*(10), 1-17.

May, T. (1993). *Social research: Issues, methods and process.* USA: Open University Press.

McCutcheon, D.M., & Meredith, J.R. (1993). Conducting case studies research in operations management. *Journal of Operations Management, 11*, 239-256.

McGuire, L. (1995). *Case studies for research: Story telling or scientific method.* Working Paper June 1995. Victoria, Australia, Department of Management, Faculty of Business & Economics, Monash University.

McNiff, J. (1988). *Action research: Principles and practice*. Hong Kong: MacMillan Education.

McQueen, R.J. (2004). *E-entrepreneurship: Proposed use of the Internet in a business plan competition in New Zealand*. Hamilton, New Zealand: Department of Management Systems, University of Waikato.

Mills, D.Q. (2001). Who's to blame for the bubble. *Harvard Business Review.*

Morrison, A. (2000). Entrepreneurship: What triggers it? *International Journal of Entrepreneurial Behaviour & Research, 6*(2), 59-71.

Mottl, J.N. (2000). Brick 'n mortar vs. dot-com. *informationweek.com*, 61, 64, 66, 68, 72.

Mulcahy, K.V. (2003). Entrepreneurship or cultural Darwinism? Privatization and American cultural patronage. *The Journal of Arts Management, Law, and Society, 33*(3), 165-184.

Murray, G. (1996). A synthesis of six exploratory, European case studies of successful existed, venture capital-financed, new technology-based firms. *Entrepreneurship: Theory and practice, 20*(4), 41-61.

MySQL (2004). *MySQL case studies.* Retrieved October 6, 2004, from http://www.mysql.com/it-resources/case-studies/

Neofotistos, G., & Yagoulis, N. (2003, June). *e-entrepreneurship and Southeastern Europe: Executive summary.* Athens, Greece, e-business Forum: 4th Working Round, Working Group WG D5.

OSI (2004a). *Open Source Initiative: Welcome*. Retrieved October 6, 2004, from www.opensource.org

OSI (2004b). *Open Source Initiative: The open source definition.* Retrieved October 6, 2004, from www.opensource.org/docs/def-print.php

Palmer, J., & France, C. (1999). Informing smaller organizations about environmental management: An assessment of government schemes. *Journal of Environmental Planning and Management, 41*(3), 355-374.

Peil, M.P., Mitchell, K., et al. (1982). *Social science research methods: An African handbook*. UK: Holdder & Stoughton.

Reddy, C.R. (1987). *Research methodology in social sciences.* Delhi, India: Daya Publishing House.

Spencer, D.L. (1982). *Researcher's guide: How and why.* College-Hill Press.

Spiegel, R. (2002). Where have all the dot-coms gone? *Electronic News, 48*, 30.

Steinberg, A. (2003). *Success and decision representations: Sense-making in e-entrepreneurship in the wake of the doctom crash.* Doctoral thesis, London School of Economics and Political Science, Houghton St. Retrieved August 16, 2004, from http://personal.lse.ac.uk/steinbea/overview.htm

Steinberg, A. (2004). *Entrepreneurship and success in e-business: On changing meanings of expertise and community in e-entrepreneurship.* London: Department of Social Psychology, The London School of Economics and Political Science, Houghton Street.

Tams, E. (2002). Creating divisions: Creativity, entrepreneurship and gendered inequality - A Sheffield case study. *Carfax Publishing, Taylor & Francis Group, 6*(3), 393-402.

Telstra (2004). *Welcome to Telstra*. Australia. Retrieved October 28, 2004, from www.telstra.com/index.php

Thompson, A.A., & Stickland, A.J. (2003). *Strategic Management: Concepts and Cases.* MCGraw-Hill Irwin.

Thompson, P., & Randall, B. (2001). Can e-learning spur creativity, innovation and entrepreneurship. *Education Media International, 38*(4), 289-292.

Thornton, P.H. (1999). The sociology of entrepreneurship. *Annual Review of Sociology, 25*, 19-46.

Wallace, R.M. (1984). The use and value of qualitative research methods. *Industrial Marketing, 13*, 181-185.

Web site 1, Company A (Date accessed: September 15 2004)

Web site 2, Company B (Date accessed: September 12 2004)

Yin, R.K. (1994). *Case study research: Design and methods.* CA: Sage Publications.

ENDNOTE

[1] Netcraft (*www.netcraft.com*), in fact, reports a 67.92% market share for the open source Apache Web server in October 2004, which is a bare 0.07% change since October 2003.

ACRONYMS USED

ERP: enterprise resource planning
IP: intellectual property
MYOB: Mind Your Own Business (accounting software package)
OSS: open source software
SAP: "Systeme, Anwendungen, Produkte in der Datenverarbeitung," meaning "Systems - Applications - Products in data processing" [url:wiki_sap]
SQL: structured query language

This work was previously published in Entrepreneurship and Innovations in E-Business: An Integrative Perspective, edited by F. Zhao, pp. 62-88, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 6.15
Channel Optimization for On Field Sales Force by Integration of Business Software on Mobile Platforms

Rishi Kalra
Symbiosis International University, India

Amit Nanchahal
Symbiosis International University, India

ABSTRACT

Marketing and sales channels are a significant lifeline for the sales force of a business. Sales professionals work on the concept of creating and widening channels that are then fed by the supply chain and distribution network of the businesses. Sales teams are constantly pushed to meet customer expectations while generating revenue for the company. As companies grow, these pressures increase. Sales teams are now looking at Mobile Sales Force Automation technologies to handle the ever increasing customer demands. Companies want to keep costs low, increase productivity and efficiency through mobile devices for the much needed edge on the field. This chapter is based on literature review of channel optimization as well as mobile software platforms and challenges faced by the sales force. This chapter discusses the need for integrating business software on mobile platforms that will optimize and enhance the performance of sales processes.

INTRODUCTION

The concept of the mobile enterprise is growing[a] in today's corporate world taking interest in tools supporting enterprise mobility for their sales force. These mobile enterprise solutions promise efficiency and productivity gains resulting from the sales function optimization. The change in mindset towards mobile applications in business has created a strong opportunity for companies to

extend their core data and applications through smart phones, cell phones, and personal digital assistants (PDAs). This extension of the business process applications results in creation of location independent links between the office (or a centralized location) with an increasingly dispersed workforce – notably the sales force which is continuously on the move.

The reason why companies are looking at incorporating such small handheld devices in their business processes is because by increased usage of mobile phones, PDAs and laptops, the company can keep in constant touch with its staff and managers who are spending more and more time away from the corporate network. These devices also provide vital access to these staff for email and the Internet. Istart Technology research found that:

- 27% of e-mail requires immediate action[b]
- 40% of the workforce is mobile
- 60% of senior management time is spent away from the desk[c]

Sales force management is the art of managing sales team on the field that enable an organization to generate revenue by selling their products to customers and increase customer satisfaction.

Companies are spending huge resources on their sales force and incorporating the new systems and applications into their operations management that help automate business processes. These applications are combined with the existing CRM functions in the organization which help enhancing the selling tactics of the on field sales force across industries by providing up to date information helping them win in the global business battlefield.

On field sales force for any company is concerned with all the stages of the sales process, starting from contact management, sales forecasting, recording sales, product solution details, integration of the various departments companywide.

A sales force personnel on the field requires constant communication with the home office for sales leads, invoicing, inventory[d] tracking, order fulfillment, and other supporting information. Recent advances in wireless technology field staff had to make do with laptops that required a physical connection and voice-based mobile phones neither providing the added value of mobility.

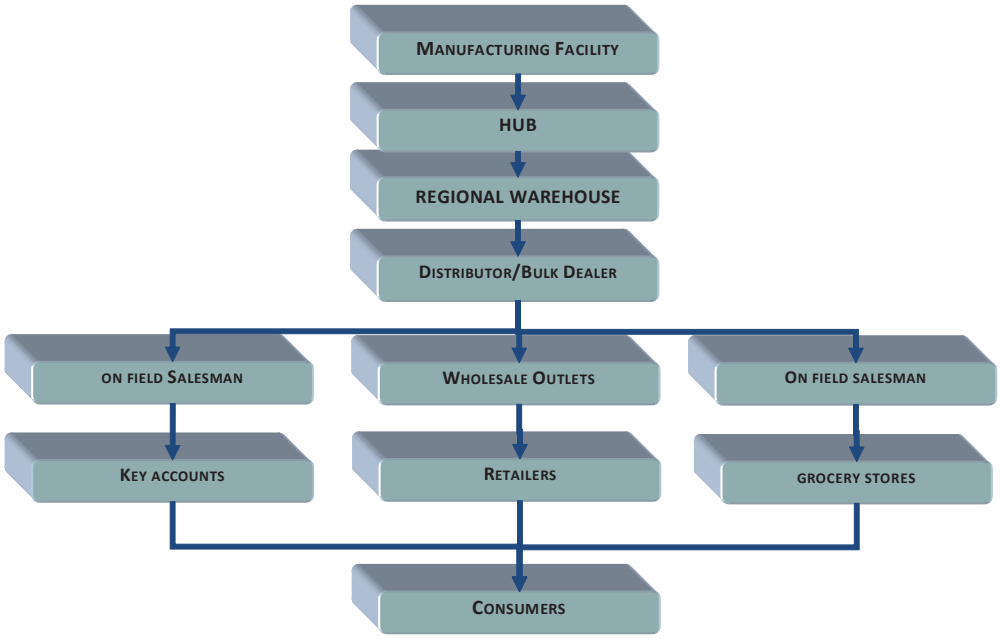
A new breed of user friendly Internet capable applications on smart phones and mobile devices are freeing the traditional sales force from their desks and allowing them to be incredibly effective. Sales personnel now have access to the same information and tools as their peers back at the office. This chapter discusses the various aspects in the channel for sales in an organization and the intervention of mobile devices to increase productivity.[e]

CHANNEL OPTIMIZATION CHALLENGE

Distribution channel is the structure that the company uses to reach its products/services to customers through intermediaries at the right place, at the right time with special consideration for profit and effectiveness. As shown in *Figure 1* products of a company may be passed from one intermediary channel member to another after keeping a certain margin for their services. The optimization of the distribution channel poses a challenge to organizations with different entities trying to transfer the product title and risk to the next successive channel member. The coordination and smooth working for optimum output for the company is the job of the sales team.

Distribution channels basically provide three functions: *information flow* (outward information about the supplier's offering and inward flow about customers' needs), *logistics* to get the supplier's product to the end customer, and *value-added services* that augment the supplier's product (local selling, financing, customization, after-sales parts

Figure 1. Distribution channel for a packaged goods industry



Reference: Report by Rishi Kalra on "Entry Stratefy for Haats/Shandies as a distribution channel in Orissa" at ITC Ltd Vizag Branch in June 2007.

Figure 2. Physical flow of goods in a distribution channel

and service, etc.). When improved alternatives for providing these functions evolve or when customer expectations rise, the failure of existing channels to respond prevents the supplier from adequately satisfying the customer. *Figure 2* describes the flow from the manufacturer to the retail point in Fast moving consumer goods category.

The shortcomings in a distribution channel are shown below:

- Excessive time taken for order processing due to lack of proper and effective communication between order procuring and order processing.[f]
- Inefficient forecasting techniques as calculated using past data rather than current on field inputs from sales force.
- Insufficient information supplied by sales personnel to retailers while pushing for sales; due to incomplete product knowledge and changing product lines.
- Increased threat of cannibalization and market fragmentation by inefficient order tracking and stock management techniques.
- Demographically heterogeneous consumers with changing habits, behavior and wants which are ineffectively monitored and stored by the sales force.
- Declining margins due to price erosion and increased costs due to excessive paperwork for completing the sales process leading to lost sales time.

Sales team for a company requires a lot of data support from the office or central location to close a deal or sell the company's products to demanding clients. The systems available now as shown in *Figure 3* incorporated on mobile devices enable the sales person to re-quote proposals, negotiate prices, manage orders and constantly update product knowledge on the move. Today the requirements from the sales force are more demanding and focused on monitoring all new business opportunities in the area, client reviews, marketing activities, competitor analysis and strategies for business development.

The various roles of a sales person are:[g]

- **Generate Sales:** Pre sales call planning, prospecting, presentations, entertain, closing sales, arrange delivery, collect payment.
- **Provide Service:** Technical consulting / solution formulation, arrange & oversee installation, train users, testing & monitoring.
- **Territory management:** Gather & analyze industry data – on customers & competitors, map distribution coverage, forecast.
- **Professional development:** Take part in sales meetings, trade shows, update product knowledge, upgrade skills, changing sales plan.
- **Other services:** Train new sales persons, counseling juniors, prepare manuals, and

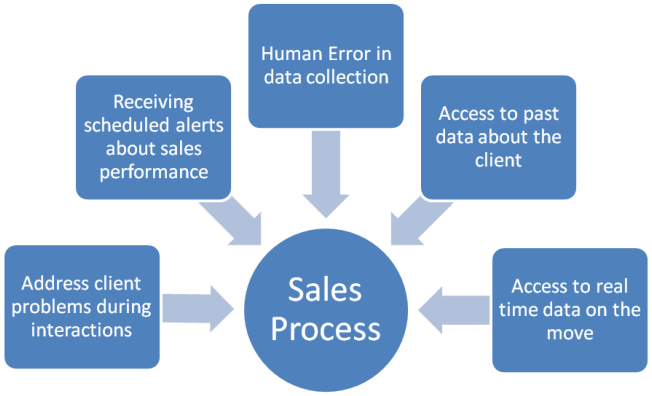
Figure 3. Process flow followed by sales force

represent company in sponsored civic/social events.

The shortcomings of the **on field sales force** are mentioned below:[h]

- *Figure 4* shows the various challenges faced by the on field sales force.
- Faster decision making required at the time of sales, access to information about price, product and stock to help close the sale on the spot.
- Need to remain as one point of contact for the company and the customer.
- Inefficiently addressing customer requirements and problems during interactions due to lack of proper negotiation and problem solving support on field.
- Challenge in identifying new sales opportunities at existing client accounts and at noncustomer, or whitespace, for companies due to lack of access to market data and current trends.
- Improper communication for receiving scheduled alerts about sales performance leading to inefficient planning and managing sales activity.
- Lack of access to customer/channel partner requirements and payment track record to manage the clients better on field.
- Inefficient use of field time[i]: A manual system requires a lot of time for data capture leading to lower efficiency. Moreover, loss of time is the most important edge against competition; increased efficiency allows sales force more time to generate sales.
- Possibility of human error: There are multiple jobs being performed by the sales force personnel while on the field which require manual data entry or estimation. Human error is critical when the decisions are made by the sales force for stock, sales forecasting, order due date, recording sales, and competition benchmarking.
- Lack of effective coordination during preparation of sales proposal due to improper flow of information between the various departments and the central database to on field agents making sales.

Figure 4. Challenges with the existing sales process system



- Inefficient monitoring and coordination of channel partners and sales force personnel on the field due to dispersed data.

The goal of Small Handheld Devices (SHD) is to streamline the entire sales process by integrating business software on the mobile devices to make businesses function efficiently, improve customer interaction, increase customer satisfaction, and save time.

Key research findings of mobile device management leader Mformation Technologies, Inc: [j]

- US enterprises reported more than half of managers using company-supplied mobile devices and nearly one-third of staff, with 56 percent reported increased usage among managers and 60 percent reporting increases in staff usage. [k]
- Mobile email, Internet and calendar applications are already pervasive, with more than 90 percent of companies using them, and businesses are set to significantly increase the use of new mobile applications such as sales force applications and company file share systems.
- 81 percent of respondents reported significant productivity increases from current mobile investments, with more than one-third of these reported increase higher than 20 percent.
- Four-fifths of CIOs interviewed look for improved management of mobile devices, applications, and data to accelerate the productivity trend, and more than 8 out of 10 US CIOs believe the mobile operator should take the lead in providing these device management services.

The major benefits of **Small Handheld Devices** for the on field sales force are:

- Data capture process optimization by pre drafted e-forms instead of manual filling of sales orders, reports, activity reports, and/or call sheets by on field sales people[l].
- Seamless information transmission from field to the central location rather than printing out reports and taking them to the sales manager. [m]
- Efficient response to client queries by use of hand held devices rather than waiting for paper based product inventory data and sales prospect lists resulting in long-lasting, profitable customer relationships.
- Ease of data entry mechanisms improve sales staff morale by reducing the amount of record keeping and/or increase the rate of closing.
- Sales staff can be easily trained with product information and sales technique training using SHD tools faster and more efficiently.
- Better communication and co-operation between sales personnel facilitates successful team selling.
- More and better qualified sales leads could be automatically generated by the software.
- This technology increases the sales person's ratio of selling time to non-selling time. Non-selling time includes activities like report writing, travel time, internal meetings, training, and seminars.
- Mobile technology providing instant access to information, applications, and services anytime and anywhere. This saves a lot of time most important business commodity.
- Improved supply chain efficiency by integrating various levels of corporate activity[n]
- Digital catalogue of products (including pictures) with sales representatives at all times.
- Orders placed by sales representatives can be automatically assigned to the sales representative that placed the order, thus streamlining the commission calculations and incentives.

Table 1.

Sales Process	Challenges	SHD Optimizations
Contact Management	1. Inefficient classification of contacts on the field 2. Cumbersome to develop new contacts from prospects with improper data capturing. 3. Difficult to remember old transactions with the contacts.	1. Efficient classification using the central database and parameters 2. Ease of recording data from prospects 3. Complete access to all transactions with the contact and customization of solution.
Order Management	1. Increased lead time in order processing due to time lag in information flow 2. Inefficient cumbersome order booking 3. Human error in recording data leading to incorrect demand estimation 4. Lack of real time access to client orders	1. Faster delivery due to reduced lead time 2. Effortless order capture 3. Increased accuracy in order processing 4. Access to past data of client orders and delivery dates.
Sales Forecasting	1. Micro level knowledge and view of the sales in the region. 2. Outdated information about targets of the served region	1.Integrated timely sales forecast generation from on field sales force (hourly, daily etc) 2. Coordinated updating of data from all mobile devices providing a holistic view of the database to sales force.
Recording Sales	1. Sales recorded manually on sale books and transferred to the database at the end of the day. 2. Human errors while recording data due to time constraints	1. Directly transferred to the database on a periodic basis synchronizing with the central database. 2. Less chances of human error due to efficient data entry mechanisms.
Charting Proposals	1. No data available from past interactions 2. Lack of access to latest sales data (product details, pricing, schemes, etc) available to field agents for preparing proposal.	1. Quicker access to past client interactions. 2. Access to latest sales data from the central database to give the client the most competitive quote.
Negotiation and Re-quote	1. Inefficient support on the field from the office during negotiations. 2. Decision making postponed till approved from higher authorities thus delaying the sales cycle,	1. Instant access to relevant sales data on the move. 2. Decision making and approval process streamlined.

- Improves cash flow by accelerating payments from customers – increasing efficiency of gathering data needed to create and deliver invoices customers.
- Since the data is collected in an electronic format the chances of errors are considerably reduced. These devices reduce human error by instructing the user with logic structure-based informational questions.

- Due to the compact form factor it can easily carried by the field work force. Handheld devices also score over notebooks as they are lightweight and fit easily into pockets. An SHD can provide quick, convenient, discreet Internet access.

MOBILE PLATFORMS REVIEW

With any new technology/solution, it is important to match requirements` with the benefits and limitations of the solution. The operating system determines a phone's features, performance, and security, providing APIs for add-on applications and technical hooks to manage it all. Decision makers face a tough choice when weighing which mobile platform or operating system to deploy to mobilize the workforce. There's BlackBerry, Windows Mobile, Palm OS, Symbian, Linux and J2ME.

Blackberry

BlackBerry offers the best combination of mobile phone, server software, push e-mail, and security from a single vendor. It integrates well with other platforms, it works with several carriers, and it can be deployed globally for the sales force which is on move. It is easy to manage, has a longer than usual battery life, and has a small form-factor with an easy-to-use keyboard. BlackBerry is good for access to some of the simpler applications, such as contact list, time management, and field force applications. It's a good device for doing e-mail, but it would be a bad choice if what you're looking for is a way to deploy business-critical applications to mobile workers.

Windows Mobile

Windows Mobile comes in two flavors. A smart phone edition is good for wireless e-mail, calendaring, and voice notes. A Pocket PC edition adds mobile versions of Word, Excel, PowerPoint, and Outlook. Palm's Treo 700w, with the full functionality of the Pocket PC edition, is a better choice for sales force professionals.

The main draw of the Windows Mobile operating system is its maker Microsoft. For a Windows or Microsoft shop, it's a wise decision, since it's fairly easy to add into the mix if a company has an affinity to other Windows applications, like Word and Excel. Windows Mobile also actively syncs to the Exchange and SQL servers. This augurs very well for use by the sales force.

Mobile sales force solutions for Windows Mobile are available from companies like SAP, Siebel, PeopleSoft, and Salesforce.com as well as other leading solution providers.

Palm OS

Palm OS is the most sensibly laid out and easy-to-use operating system on a smartphone today. This might suit the business requirement of some sales force implementations. Palm brings an open approach to business with two enterprise-ready, non-proprietary operating systems: classic Palm OS platform and the familiar Windows Mobile platform. Both platforms deliver secure mobile email, Palm's ease of use, access to thousands of industry-specific enterprise applications, and the ability to build custom applications using your existing development resources. The Palm OS does not allow for multitasking, which many enterprises could find a great hindrance. For example, if a user is working in an application and the phone rings, the application has to be closed down in order to take the call.

Symbian

Symbian[o] is one of the most widely used platforms. Its plug-in architecture makes it easier for manufacturers to add technology of their own which hastens delivery of new and in-demand features. Although Symbian is feature rich, it doesn't inte-

grate well with corporate back-end systems such as VPN connections and other enterprise-oriented tools. These devices work well for many traveling professionals, managers, knowledge workers and sales teams, but enterprise IT departments may find Symbian OS devices to be too expensive and too feature rich for verticalized field-force applications. Symbian has a strong feature set and is relatively easy to use. Its main stumbling blocks are the limited number of Symbian devices and its lack of support for CDMA. However, Symbian will start making its way to mobile workforces, even if it can't conquer the enterprise market.

Linux

Mobile Linux is not really a mobile operating system, but a kernel that can be a central part of any number of different operating systems. Mobile Linux cannot really be weighed against other mobile operating systems like Palm OS and Windows Mobile, simply because there are many flavors of Linux. Linux has an advantage over other mobile operating systems: a far-reaching community of developers ready to write smartphone applications. One can say things like 'Palm OS has an intuitive user interface' or 'Windows Mobile requires a more powerful processor,' but these generalizations can't be made about mobile Linux because it can have many different user interfaces depending on the distribution.

J2ME

J2ME is also not an operating system, it is a platform. The main selling point for J2ME is that it's lightweight and has a simple methodology for designing applications. Because it is lightweight, it requires very little storage. There are a few key applications that perform well in a J2ME environment, namely those from SAP AG, Oracle Corp. and IBM. What holds Java back is similar to what keeps Linux from infiltrating the enterprise as a viable mobile platform. The issue is there's no single party in charge of Java. In a company with a deployment of several different devices, that could create issues, since applications written to Java would have to be tested and tweaked for each, creating more work and potentially introducing performance issues if it is not tweaked just right.

Google's Android Mobile Platform

Google's Android Mobile platform is the latest mobile platform on the block. This open-source development platform is built on the Linux kernel, and it includes an operating system (OS), middleware stack and a number of mobile applications. Enterprises will benefit from Android because the availability of open-source code for the entire software stack will allow the existing army of Linux developers to create special-purpose applications that will run on a variety of mobile devices. If Android makes it into phones designed specifically for the enterprise, those products will have to include technology from the likes of Sybase, Intellisync or another such company to enable security features like remote data wipe functionality and forced password changes.

BUSINESS SOFTWARE INTEGRATION

The heart of the SHD device is its software and companies want employees on the field to get a feel of familiar applications they are trained to use on their mobile devices. Applications being developed in the SHD space depend upon the companies' requirements. There are industry-specific applications being developed; for instance, applications for an FMCG sales force, for insurance agents, the police, the military, and for e-governance. However, the applications have to be simple and effective. A salesman capturing information during a sales call has to complete all information, then and there, in a short span of

time. Hence it is very important for the applications to be simple and effective.

SHDs have proved to be a boon for the sales force of many organizations. Prior to having a handheld device the sales force had to depend on manual, paper-based systems. It used to take a month to compile data collected from different sources such as dealers, distributors etc. Software loaded in the SHD enable each member of the SF to maintain, store and feed information, and then transfer the same to central location. There are several protocols that sales people have to meet in different set-ups when they contact a distributor or retailer during their visits. They have to get information about products based on re-launch, regular purchase and different schemes. A salesperson does not spend much time with each channel partner and the challenge of recording all the parameters is automated and this has reduced the time required.

We'll understand the business software integration through an example from Insurance industry[p]. Gathering assets through partnerships and affiliations is critical for insurers to increase revenue. Third-party distribution continues to dominate the distribution of insurance products. Sales personnel spend most of their workday in the field in prospecting the customers. These sales personnel come back to home office periodically to update the office records which in turn would trigger the processing cycle. This often induces delay in the policy processing and increases the chances of data related errors. These work customs compel insurance carriers to seek cost-effective ways to deliver "anytime, anywhere" interactions between sales personnel, their home office, and their clients.

Mobile solutions[q] are being adopted in areas of new business acquisition and claims processing. It can improve day-to-day service levels for sales field force as it thwarts the need for duplicate data recording. Mobility lessens the need for multiple local offices in the same city which are frequently established for convenience of sales personnel[r]. Instead of sales personnel shuttling between local offices and client they can assess the insurability on site and immediately communicate with the carrier.

Exhibit illustrates a typical new business acquisition process flow using mobile devices to move data and information from the field to the home office of the agent. By instantly notifying underwriting[s] office of a new customer application and assigning them the case dynamically, Insurance[t] companies could expedite new policy processing. Through secure mobile environment, carriers can transmit applicant's demographics information, health information and other key details. The sales personnel can accept the business in real time, thereby improving the continuity between the sales activities and underwriting office.

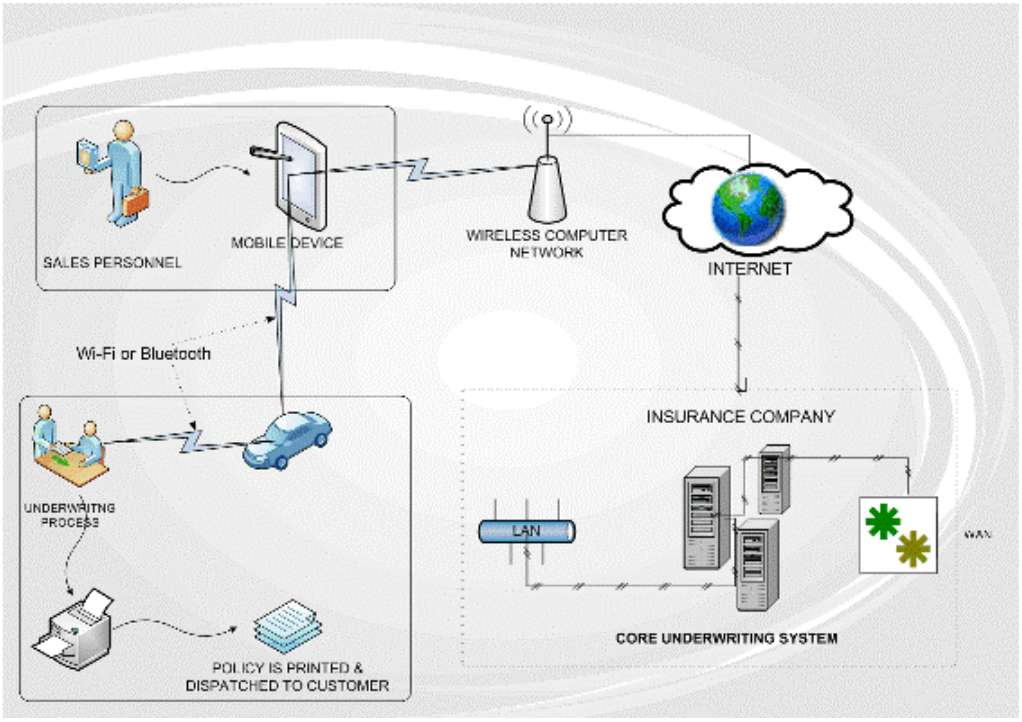
As the sales agent can collect detailed information on the spot, it often improves the accuracy of data and transactions. Mobile technology streamlines new business acquisition; increasing productivity in the process and reducing the potential for delayed policy processing and causing discontent among newly acquired customers. Both life and P&C carriers need to examine how a business process will work with handheld devices and then optimize these processes. Companies will have to do this optimization in perspective of the mobile wireless existing infrastructure available with the company and in the country, and how much is the company willing to spend on new mobile devices and wireless networks.

VALIDATION CASE

To illustrate the effectiveness of Mobile handheld devices in Sales-force management the following case has been chosen.

In a typical Consumer packaged goods company the sales force of the distributor typically perform the functions of Order Capture, Order Entry, Business Development and transfer of promotions designed by the company to the dealer. A

Figure 5. Architecture for a possible mobile sales force application



typical day of the sales force is spent visiting outlets making calls, booking orders manually, collection of pending claims, checking display efficiency. After covering his beat the sales person returns to the office to make the order entry for the day and plan the dispatch for the next morning. In addition to this he prepares invoices to be handed out to the retailers once Pros are dispatched. Since there is a time lag of 7-10 days between the dispatch and payment of the invoice the sales person also needs to keep a track of pending claims which is currently done manually and then fed into the system. If we analyze the current system there are gaps in terms of: duplication of effort when the order entry is done both at the dealer's premises and the office, manual entry of figures which can be mishandled. Moreover additional effort needs to be undertaken in terms of preparing statement claims, interpreting these and analysis of the offtake levels is adhoc.

Let us consider a scenario when handheld devices are used by the organization. The sales-force of the organization is equipped with portable devices. The first capability of the device is towards enhancing the order booking process. The device can either be 'always on' – permanently connected to office servers wirelessly – allowing users to dispatch their orders real-time. Or it can connect and synchronize at various times during the day – sending orders and receiving relevant sales information and emails etc. The order process will in many ways mimic what was

once done with paper orders, however, instead of faxing them in or delivering them to the office by hand, the orders are sent wirelessly. The suite of tools & information available in a mobile sales application include sales orders, customer sales history, corporate email, (typically synchronized with Microsoft Outlook or Lotus Notes), sales opportunities and promotions, often with the ability to record proposals and quotes, and marketing information like special trade promotions relevant to that particular customer. These modules contain potentially useful information about past promotions, their effectiveness with the customer and various options which may be customized to the buying pattern of the concerned customer.

This means the order dispatch process begins almost as soon as the order is taken, and it also means office bound order entry staff don't have to re-key orders (and potentially make mistakes while doing so) which ultimately means less credits and more satisfied customers. This is a significant improvement over the manual process where in the sales person had to replicate the effort in order booking and order entry which doubled the time required and the risk of inaccurate entries. Moreover when linked with the database maintained at the distributor/stockist premises reports related to the claims pending, off-take levels, success of promotional schemes can be tracked and used seamlessly by the company to make decisions. Another application is the automatic replenishment of stocks at the distributor level once the stocks reach a certain level. This helps in seamlessly integrating information across the supply chain and improving efficiency of channel partners and the manufacturing organization

This apart one major benefit is better utilization of a sales rep's time. The new operation enables the sales representative to undertake more calls, enhance the value of each order and gather competitive intelligence with the saved time.

We attempt to evaluate the benefit of investing in mobile technology. Let us assume the hypothetical example of a team of 20 sales reps, making an average of eight sales calls a day. Assuming that a sales person spends 15 minute per outlet visited to transfer the order information into an order sheet and this plus the travel time spent in driving to the office premises at the end of the day would amount to two hours wasted in a day. Even if the additional time generated by using as mobile application allowed the sales representative to make an additional sales call each day, or an additional $40 profit per sales (including the additional profits made due to better service) call made, this would amount to a staggering additional profit of $41,600 made in a year by the team. Add to this the additional benefits of better reporting, information synergies which help in designing strategies by the company and the reduced chances of stock unavailability at the distributor premises and the application brings excellent value. Assuming an investment of $800 per mobile device and the additional cost of altering the IT infrastructure the fixed one time cost would amount to $20000; thus illustrating the profitability of the technology.

CONCLUSION

An integral part of any mobile device system trying to automate the company's processes is companywide integration among different departments. If the systems aren't adopted and properly integrated to all departments, there might be a lack of communication which could result in different departments contacting the same customer for the same purpose. In order to mitigate this risk, sales force mobile devices must be fully integrated and kept up to date from all departments that deal with customer service management. The applications that are transferred or made mobile ready should be chosen such that they provide the most relevant, effective and up to date information to the on field sales force rather

than over burdening the sales team member with other administrative tasks.

Some of the applications available for the mobile devices are:

- There is a growing breadth of specialist applications on or coming to the market which enable fast efficient interface with your customers.
- These include specialist Sales Force Automation, Field Service Automation, Job dispatch and Tracking, Mobile Sales Force Management, Easy Order Software and Transport and Logistics Management. These applications all ensure delivery of a higher customer service level or increase the efficiency of the remote team your company has on the ground.
- To attain a sustainable advantage over competition, as the geographical boundaries become redundant, it is medium like telecommunication which will define the rules of the game.

REFERENCES

Author, Cynthia Saccocia & Author, Bob Egan(2006). Handheld device trends in the US Insurance Industry: TowerGroup Inc.Microsoft (2006). *Mobile Field Sales.* Redmond, WA: Microsoft Corporation

Frost and Sullivan report on mobile business on www.allbusiness.com, 7 Feb. 2007 Retrieved on 4th January 2008. http://www.expresscomputeronline.com/20021223/indtrend1.shtml, pricing and applications will drive PDA growth retrieved 7th January 2008.

IDC (2006). Enterprise mobility - Not a packaged solution. Network Magazine India. Retrieved January 16, 2008, from http://www.networkmagazineindia.com/200601/index.shtml

Informal telephonic survey carried out in companies in India like Cognizant and ITC Ltd.

Jennifer O'Brien (2007). Evolving to a mobile enterprise platform. ARN. Retrieved January 16, 2008, from http://www.arnnet.com.au/index.php/id;346556794;pp;1

LOMA (2006), Supporting the Insurer's Distribution Systems. Chapter 7, LOMA-290: Life Office Management Association, USA

LOMA (2006), The process of underwriting, Chapter 8, LOMA-290: Life Office Management Association, USA

Mobile Business Application Usage to Surge by 2009, Driving Need for Device Management, Business wire, Monday, July 23 2007 retrieved on 6th January 2008.

Mobile Business Application Usage to Surge by 2009, Driving Need for Device Management, publication: Business Wire, Date: Monday, July 23 2007 retrieved on 13th January 2008.

Mobile Business solution overview retrieved from www.istart.com on 19th Jan 2008.

Mobile business solutions from http://www.istart.co.nz/index/HM20/PC0/PV22447 Retrieved on 7th January 2008

Nokia, *The Symbian platform (2000). Finland*: Nokia Group

Retrieved from http://www.istart.co.nz/mobile-business.htm on 19th Jan 2008

Rohit Prakash (2006). *PDA Applicability for the Sales Field Force. Marico, India.*

Sales Force Automation Software, retrieved from http://www.stylusinc.com/Common/Scenarios/pda.php on 18th January 2008.

Sales on the move, David McNickel investigates the inherent power of today's mobile sales ap-

plications, September 2006 retrieved on 7th January 2008.

Slide presentation on the sales force by Professor of marketing Mr Abhijit Ranade, SIBM, pune.

Smartphones add to mobile productivity, by Abhinav Singh, 2001 Business Publications Division (BPD) of the Indian Express Newspapers

TCS (2007). *Handheld Solution on Symbian Platform*: Tata Consultancy Services

KEY TERMS

Claim Processing: The process of obtaining all the information necessary to determine the appropriate amount to pay on a given claim. Process of determining an insurance company's liability for each claim.

New Business Acquisition: The risk evaluation an MCO performs when it first issues coverage to a group.

Underwriting: Assessing and classifying the degree of risk represented by a proposed insured

ENDNOTES

1 Retrieved from http://www.istart.co.nz/mobile-business.htm on 19th Jan 2008

2 Frost and Sullivan report on mobile business on www.allbusiness.com, 7 Feb. 2007 Retrieved on 4th January 2008.

3 Mobile Business solution overview retrieved from www.istart.com on 19th Jan 2008.

4 Sales on the move, David McNickel investigates the inherent power of today's mobile sales applications, September 2006 retrieved on 7th January 2008.

5 Slide presentation on the sales force by Professor of marketing Mr Abhijit Ranade, SIBM, pune.

6 Mobile Business Application Usage to Surge by 2009, Driving Need for Device Management, Business wire, Monday, July 23 2007 retrieved on 6th January 2008.

7 Rohit Prakash (2006). PDA Applicability for the Sales Field Force. Marico, India.

8 Mobile Business Application Usage to Surge by 2009, Driving Need for Device Management, publication: Business Wire, Date: Monday, July 23 2007 retrieved on 13th January 2008.

9 Smartphones add to mobile productivity, by Abhinav Singh, 2001 Business Publications Division (BPD) of the Indian Express Newspapers

10 TCS (2007). Handheld Solution on Symbian Platform: Tata Consultancy Services

11 http://www.expresscomputeronline.com/20021223/indtrend1.shtml, pricing and applications will drive PDA growth retrieved 7th January 2008.

12 Sales Force Automation Software, retrieved from http://www.stylusinc.com/Common/Scenarios/pda.php on 18th January 2008.

13 Nokia, The Symbian platform (2000). Finland: Nokia Group

14 Author, Cynthia Saccocia & Author, Bob Egan(2006). Handheld device trends in the US Insurance Industry: TowerGroup Inc.Microsoft (2006). Mobile Field Sales. Redmond, WA: Microsoft Corporation

15 Jennifer O'Brien (2007). Evolving to a mobile enterprise platform. ARN. Retrieved January 16, 2008, from http://www.arnnet.com.au/index.php/id;346556794;pp;1

16 IDC (2006). Enterprise mobility - Not a packaged solution. Network Magazine India. Retrieved January 16, 2008, from http://www.networkmagazineindia.com/200601/index.shtml

[17] LOMA (2006), The process of underwriting, Chapter 8, LOMA-290: Life Office Management Association, USA

[18] LOMA (2006), Supporting the Insurer's Distribution Systems. Chapter 7, LOMA-290: Life Office Management Association, USA

This work was previously published in Handbook of Research in Mobile Business: Technical, Methodological and Social Perspectives, Second Edition, edited by B. Unhelkar, pp. 182-193, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 6.16
Revenue Models in the Open Source Software Business

Risto Rajala
Helsinki School of Economics, Finland

Jussi Nissilä
University of Turku, Finland

Mika Westerlund
Helsinki School of Economics, Finland

ABSTRACT

Profit-oriented business behavior has increased within the open source software movement. However, it has proved to be a challenging and complex issue due to the fact that open source software (OSS) business models are based on software that typically is freely distributed or accessed by any interested party, usually free of charge. It should be noted, however, that like all traditional software businesses, the business models based on OSS ultimately aim at generating profits. The aim of this chapter is to explore the key considerations in designing profitable revenue models for businesses based on OSS. We approach the issue through two business cases: Red Hat and MySQL, both of which illustrate the complexity and heterogeneity of solutions and options in the field of OSS. We focus on the managerial implications derived from the cases, discussing how different business model elements should be managed when doing business with OSS.

INTRODUCTION

Whereas the business models of the traditional providers of proprietary software are grounded in one way or another on the distribution of access to the use of software-related intellectual property (IP) protected by copyrights, business models within the open source movement have to rely on other types of revenue models. This is due to the fact that open source software (OSS) business models are based on software that typically is freely distributed or accessed by any interested

party, usually free of charge. OSS is often mistaken for shareware or freeware, but there are significant differences between the licensing models and the processes between and within these types of software. It should be noted, however, that like all traditional software businesses, the business models based on OSS ultimately aim at generating profits. However, profitability and business models of OSS are still poorly understood phenomena, and there is no single framework that would explain the potential determinants of firm-level revenue model choices.

In this chapter, we make an attempt to identify key considerations in designing successful revenue models in the OSS business. We explore the revenue models of two selected OSS business cases. Through these cases, we aim at identifying the firm-specific business model elements that guide, enable and constrain the choice of revenue model options in OSS business. As a limitation to the analysis presented in this chapter, we leave the exogenous factors (such as competition and other environmental factors) beyond the scope of our consideration.

BACKGROUND

In this chapter, we discuss the background of the OSS business, typical licence OSS choices, and the potential for conducting for-profit business with OSS.

Development of OSS Business

The history of the open source movement goes back to the early ages of computing. In the 1960s and 1970s, it was common for programmers in certain academic institutions (e.g., Berkeley, MIT) and corporate research centers (e.g., Bell Labs, Xerox's Palo Alto Research Center) to share computer program source codes with other programmers. It was not until the early 1980s that proprietary software became very popular, thus causing problems with cooperative software development (Lerner & Tirole, 2002). The predecessor of the open source movement, the Free Software Foundation (FSF), was founded in 1983 by MIT employee Richard Stallman in his attempt to formalize cooperative software development and create a complete free[1] operating system with necessary software development tools. This project was called the GNU Project. Stallman's general concept of free software possesses four essential freedoms (Stallman, 1999):

- Freedom to run the program
- Freedom to modify the program
- Freedom to redistribute the program
- Freedom to distribute modified versions of the program

Stallman didn't want to release software with restrictive copyright terms because it would prevent certain forms of valuable cooperation. On the other hand, releasing software to the public domain would leave it vulnerable to be copyrighted and included in proprietary packages. Thus, Stallman came up with the idea of copyleft, or protecting the freedom of software with the means of copyright laws. In addition, copyleft ensures that the modified works are also released under copyleft terms and, therefore, to the use of the community. Stallman, (2002) argues, "Proprietary software developers use copyright to take away the users' freedom; we use copyright to guarantee their freedom. That's why we reverse the name, changing 'copyright' into 'copyleft.'" To implement this idea, the FSF developed the GNU General Public License (GNU GPL), the first of the now extensive selection of copyleft licenses that are used to protect free/OSS. Meanwhile, the open anticommercialism of FSF led to a group of free software movement leaders deciding to find new ways to strengthen their cause, but with less radical means. They came up with the term "open source," which they thought would better describe the software ideals, and founded the Open Source

Initiative (OSI). The idea of the organization was to promote the Open Source Definition (OSD), a set of terms for licences, which is more adaptable to commercial use than the approach FSF took. OSI has since registered a certification mark, and there is a variety of OSI-certified licenses (including GNU GPL and other copyleft licenses).

What motivated the birth of OSI was the way free software was being developed in such projects as the Linux operating system since the beginning of the 1990s. The new development model introduced in the Linux project was first described in "The Cathedral and the Bazaar," an essay written by Eric Steven Raymond, one of the founders of the OSI (Raymond, 2001). The Linux development model was seen as a better way of software development that could lead to higher quality and rapid advancement. Cooperational software development was not only for the ideologists and community-spirited anymore, but rather something also to be used in more commercial projects. The new emphasis born with the OSI made it possible for the business world to intensively embrace OSS. Before 1998, relatively few people in the IT industry knew about free software; however, a couple years later, open source was on many people's lips. With the participation of big IT companies such as IBM, Hewlett Packard, and Nokia, open source has become a credible player in the IT field.

OSS Licensing

OSS, exactly defined, is software fulfilling the terms of distribution given in the OSD and adopting a license approved by the OSI (Open Source Initiative, 2004). Summarizing the ideas behind the terms in OSD, the software license must generate the following effects:

- Source code must be readable and available, either included with the binary code or publicly downloadable
- Free distribution of the software by any party, on any medium, to any party, gratis or for a fee
- Derivative works must be allowed, either under similar license or not, depending on the specific OSS license type
- No discrimination against persons, groups, or fields of endeavor

The nature of OSS is in the licensing terms and not just the accessible source code, which is just one part of the features the licensing terms generate. In addition, the licensing terms allow the free use, redistribution, and modification of the software. The copyright owner preserves the moral rights and some economic rights, such as the right to dual-license the software, but transfers many important rights to the users and developers of the software in order to enable the development of the software and to increase its adoption. It is important to understand that the OSD licensing terms allow the creation of many types of OSS licenses, each with different qualities. Välimäki (2005) categorizes OSS licenses into three functionality classes, ranging from the most liberal to the most restrictive. The categories are permissive licenses, licenses with standard reciprocity obligation, and licenses with strong reciprocity obligation. Standard reciprocity means that the distribution terms of the source code must be maintained in further developed versions, which is also called the "copyleft" effect. Strong reciprocity obligation means that in addition to standard reciprocity effects, derivative works and adaptations must keep the licensing terms intact, also called the "viral" effect.

Välimäki (2005) has studied the prevalence of different OSS license types. Table 1 presents the most popular licenses as surveyed in his study at SourceForge.net in 2004 (Välimäki 2005), together with their functionality and relative popularity in project licensing.

In Table 1, the popularity percentage refers to the occurrences of these license types among

Table 1. Most popular OSS licenses and their functionality (Source: Adapted from Välimäki, 2005)

License	Functionality	Popularity
GNU GPL	strong reciprocity	66.50%
GNU LGPL	standard reciprocity	10.60%
BSD	permissive	6.90%
Public domain	permissive	2.70%
Artistic	permissive	2.00%
MIT	permissive	1.70%
Mozilla	standard reciprocity	1.50%
Common Public License	strong reciprocity	0.60%
Zlib	permissive	0.50%
QPL	strong reciprocity	0.40%
Open Software License	strong reciprocity	0.40%
Python License	permissive	0.40%
Academic Free License	permissive	0.30%

all OSS licenses (surveyed at the SourceForge.net in late 2004).

Special Characteristics of OSS Business

One of the most critical issues for OSS business is that the licensing terms allow free redistribution of the licensed software (i.e., the licenser doesn't necessarily gain any revenue from these copies of the software). In fact, charging a fee for OSS is usually not feasible, because (1) any buyer may start to resell the software or give it away and (2) fees could severely diminish the rate at which both developers and users adopt the software product (De Laat, 2005), which often is the motivation behind licensing a product as OSS. Therefore, it is usually not feasible to base the revenue logic on licensing fees. It is also possible to use OSS as part of a firm's other products; namely, software packages, hardware, and/or services. This approach is not free of challenges either, since the unique licensing of OSS may create risks as well as opportunities.

Many firms conducting business with OSS are in some way dependent on the OSS community for developing software in their product offerings, for support, or for customers. However, the OSS community is outside the hierarchical control of the firms since there are normally no contractual agreements between them. In addition, the idea of exploiting the financial value of a jointly developed community might go against the values of the community (Dahlander & Magnusson, 2005) in which the code is actively protected from being appropriated by commercial firms through the use of legal and normative mechanisms (O'Mahony, 2003). However, the attitudes and policies toward the commercial exploitation within the OSS community range from the critical attitudes of FSF

Table 2. Firm-community relationship (Source: Modified from Dahlander & Magnusson, 2005)

Approaches	Description	Nature of Relationship
Parasitic approach	Focuses on firm's own benefits without considering possible damages to the community	Search for useful input without obeying norms, values, and rules
Commensalistic approach	Firm aims to benefit from the community	Search for useful input from the community
Symbiotic approach	Firm tries to codevelop itself and the community	Give something to the community, often through a firm-established community

and copyleft licensing to the more liberal attitude of OSI and permissive licenses.

Dahlander and Magnusson (2005) propose three approaches a firm can use to relate to the OSS community. In question is the parasitic approach in which the firm focuses on its own benefits without considering possible damages to the community. Since the firm doesn't share the norms, values, or rules of the community, the possibility to influence community development does not exist. The commensalistic approach is about benefiting from the community while leaving it otherwise indifferent. Since the firm isn't considered hostile, influencing the community is possible but difficult. Also in question is the symbiotic approach in which the firm tries to co-develop itself and the community. This demands heavy involvement in community development and sharing of norms and values but also allows influencing community development in a desired direction. These approaches are illustrated in Table 2.

MAIN FOCUS OF THE CHAPTER

According to recent studies, the business-model concept includes some elements of business strategy and aims to describe the business as a manifestation derived from strategy (Rajala, Rossi, & Tuunainen, 2003; Osterwalder, 2004; Morris, Schindehutte, & Allen, 2005). It has also been defined as an abstraction of business (Seddon & Lewis, 2003), which characterizes revenue sources and specifies where the company is positioned in its value-creating network in a specific business. The essential elements of various business models are defined in different words by several researchers (Rajala et al., 2003; Hedman & Kalling, 2003; Osterwalder, 2004; Morris et al., 2005, Rajala & Westerlund, in press). Many of the studies identify a number of elements that are characteristic of various business models. These elements, expressed in different words by different authors, include the following: (1) offerings; (2) resources needed to develop and implement a business model; and (3) relationships with other actors (Timmers, 2003; Osterwalder, 2004; Morris et al., 2005). Finally, these elements are interconnected with (4) the revenue model, including sources of revenue, price-quotation principles, and cost structures, which is characteristic of a particular business. Grounded on the previous review and summation of the prior research literature, we identify three business model elements in order to describe the revenue models in the OSS business. These

business model elements are key considerations on which firms should focus after the decision to participate in an OSS business. In the following we discuss these elements in detail.

Offering

In the literature of business and management, the concepts of product strategies and product offerings are discussed widely (Cravens, 1987; Kotler, Armstrong, Saunders, & Wong, 1996). We see that offerings embody several aspects within the concept of a business model and, thus, affect the revenue model. Generally, type of offering, target market, product vs. service orientation, licensing model, and so forth, can be considered as aspects related to the product strategy. Likewise, the product offering includes aspects such as complexity, the essential benefit that the customer is really buying, and product features, styling, quality, brand name, and packaging of the product offered for sale (Kotler et al., 1996).

From the business model perspective, a defining characteristic of OSS as a product is that it is not a physical but rather an information product. Information, or digital, products have unique characteristics that differ largely from physical product characteristics. However, certain open source business models, such as widget-frosting and accessorizing (see the following), consist also of physical products. In addition, OSS revenue models such as support selling, service enabling, and software franchising, are comprised mostly of service components, which also have a very different nature.

In addition to the type of offering, license types are considered part of the offering element in our conceptual model as a determinant of revenue model choices. Indeed, the licensing issues and commitment to the principles of OSS licenses (GPL, etc.) are key issues related to information products such as OSS solutions (Lee, 1999).

Resources

The development of resources in the industrial-network perspective is linked to its strategy (Håkansson & Snehota, 1995; Gadde & Håkansson, 2001; Sallinen, 2002). According to this view, resources vary according to the business and product strategy. The resources and capabilities of a firm are among the central issues in understanding and analyzing its business. This accentuates the essence of resources in core competencies (Selznick, 1957; Prahalad & Hamel, 1990), as they are generally seen as firm-specific property that is subordinate to the core competencies. The resource-based view of the firm originated from the work of Penrose (1959) and was further developed by Wernerfelt (1984). According to Penrose (1959), bundles of resources that are activated in different ways lead to incoherent performance and heterogeneous outputs in various organizational settings.

In our analysis of the resources in the OSS business, we share the view of Metcalfe and James (2000), who define tangible and intangible assets as physical and nonphysical resources, and capabilities as intangible knowledge resources. Furthermore, we see that the increasing complexity of OSS markets makes it difficult for firms to have all the necessary resources in their possession to compete effectively. This view is consistent with the research of Ariño and de la Torre (1998). These resource-related approaches provide us with a basis on which to identify key resources in various types of OSS business models. They deepen our understanding, especially of how resources are applied and combined by a firm, and take inimitable resources as a basis for the creation of sustainable capabilities as described in other technology-intensive industries such as those by Hart (1995) and Gabrielsson (2004).

Relationships

We see that the elements in our conceptual model are interrelated with each other and, therefore, are consistent with Håkansson and Snehota (1995) and Rosenbröijer (1998) that capabilities of a company reflect its success in combining resources to perform activities through internal and external relationships.

As pointed out in the previous discussion, we need to consider the interaction of companies with other actors as an inseparable part of a business model, similar to offerings and resources. Timmers (2003) points out that in the context of business models, the focus shifts from creating value through internal activities to creating value through external relations. He identifies these relationships within the value-creating network as an important element in the development and distribution of offerings. In addition to being an important intangible company asset, a firm's network offers access to the resources of other network actors (Foss, 1999; Gulati, Nohria, & Zaheer, 2000; Chetty & Wilson, 2003, Möller & Svahn, 2003).

Revenue Model

Discussion of the revenue models in the context of OSS has traditionally been problematic since the OSS movement emphasizes free distribution of intellectual property. However, since the emergence of the OSS movement, there has also coexisted a favorable attitude toward earning money and, more generally, toward profit-oriented behavior based on the OSS (Raymond, 2001).

Concerning open source as an economic phenomenon, De Laat (2005) argues that whether an enterprise involved in the open source business chooses to license its own software product as open source or tries to benefit from existing OSS products, the ways of making money with open source are basically the same. These ways include selling services to facilitate OSS use, selling connected hardware, and selling commercial closed applications to use with OSS. However, Hecker (1999) has identified eight possible revenue models to be applied in conjunction with OSS. These models are described in Table 3.

Although Hecker's list of OSS revenue models (summarized in Table 3) was published as early as 1999, it still remains one of the most comprehensive classifications of OSS revenue models. It clearly points out that a company has a multitude of options to capture revenue with OSS.

CASE EXAMPLES

In our literature review, we identified three endogenous business model elements (i.e., offering, resources, and relationships) that affect the revenue models in the OSS business. In this chapter, we illustrate these determinants and their interconnectedness with the revenue model in two empirical examples: MySQL and RedHat. We see that these case examples improve understanding of the interrelatedness of these business model elements, and especially their role as determinants in setting up the revenue model. Furthermore, the cases illustrate the complexity and heterogeneity of solutions and options related to revenue models in the field of OSS business.

MySQL

The MySQL trademark and copyright are owned by the Swedish company MySQL AB. Two Swedes, David Axmark and Allan Larsson, founded MySQL AB, together with Michael "Monty" Widenius, a Finn who is broadly appreciated as the chief designer and developer of the system. The company develops and maintains its key product offering, the MySQL open source database system, in close collaboration with the

Table 3. Summary of OSS revenue models (Source: Modified from Hecker, 1999; Välimäki, 2005)

Revenue Model	Description	License Types	Revenue Sources
Support selling	A for-profit company provides support for a software that is distributed free of charge.	Any	Revenue comes from media distribution, branding, training, consulting, custom development, and post-sales support for physical goods and services.
Loss-leader	A no-charge open source product is used as a loss leader for traditional commercial software (i.e., the software is made free by hoping that it will stimulate demand for a related offering of the company).	Varies	Complementary offerings (e.g., other software products)
Widget-frosting	Companies that are in business primarily to sell hardware can use this model to enable software such as driver and interface code. By making the needed drivers open, the vendor can ensure that they are debugged and kept up to date.	Any	The company's main business is hardware. This is quite similar to the loss-leader model.
Accessorizing	Companies that distribute books, computer hardware, and other physical items associated with and supportive of OSS.	Any	Supplementary offerings
Service enabler	OSS is created and distributed primarily to support access to generating revenue from consulting services and online services.	Any	Service fees
Brand licensing	A company charges other companies for the right to use its brand names and trademarks in creating derivative products.	Strong reciprocity	Copyright compensations
Sell it, Free it	A company's software products start out their product life cycle as traditional commercial products and then are converted to open source products when appropriate.	Alteration of license type	Initial revenue from software product offerings converted into other models (e.g., the loss-leader model)
Software franchising	A combination of several of the preceding models (in particular, brand licensing and support sellers) in which a company authorizes others to use its brand names and trademarks in creating associated organizations doing custom software development; in particular, geographic areas or vertical markets.	Strong reciprocity	The franchiser supplies franchisees with training and related services in exchange for franchising fees of some sort

OSS community over the Internet. Unlike projects such as Apache, MySQL is owned and sponsored by a single for-profit firm, MySQL AB. In addition to providing the database product under the GPL license, the company sells support through service contracts as well as commercially-licensed copies

of the MySQL database software, and employs people all over the world to communicate about the use and development of the product.

Offering

The offering of MySQL AB is a multithreaded, multiuser SQL (structured query language) relational database server (RDBS) software. The software is available either under the GNU GPL or under other licenses when the GPL is inapplicable to the intended use. MySQL provides database products for integrating software vendors and original component manufacturing (OCM) partners, enterprise organizations, and private users in the OSS community. To distribute its offering to a large number of users worldwide, MySQL AB has applied a dual licensing principle by making the MySQL database software available for free on the Internet under the GPL and selling it under proprietary licenses when the GPL is not an ideal option and in situations such as inclusion of MySQL technology in closed source products. In summary, the core offering of MySQL AB embodies an in-house developed software product and related services.

Resources

As a symbol of the key resources of MySQL AB, chief technology officer Widenius began programming databases in 1981. He worked previously in Tapio Laakso Oy developing systems that needed data storage. Similarly, Axmark and Larsson, his two colleagues and later cofounders of MySQL, collaborated in programming projects from 1983 to 1995 and accumulated knowledge about database systems. By licensing the MySQL product under an OSS license, the company transferred some of its internal intellectual property resources to the open source community, thus gaining possible future clients as well as developers and enthusiasts to support its offering. The internal programming resources can still be considered the key element in the MySQL business model. Currently, 80% of the source code in the MySQL core database product (version 4.0) is programmed by in-house programming resources; the company has systematically invested in professional management resources to successfully manage its growing for-profit business.

Relationships

As already described, the collaboration based on personal relationships between key individuals can be seen as the key determinant of success in the early phases of the MySQL product development. This open atmosphere and knowledge-sharing culture between the cofounders of MySQL AB provided a sound base for enlarging the network to OSS-oriented Internet communities. At present, partners in the business network of MySQL include companies such as suppliers, distributors, outsourcing service providers, other key companies in the OSS field, commercial research institutions, and other strategic partners. Relationships with these actors are based on commercial multi- or bilateral activity. Furthermore, relationships in the business network include collaboration with public (government) organizations, research institutes, and so forth.

Relationships within the OSS community are a multifaceted phenomenon. According to the company CEO, the community of 5 million MySQL users includes several groups that produce MySQL books and articles as well as conduct courses and presentations. Furthermore, these ecosystems develop applications in different OSS projects. Currently, MySQL AB is balancing between the OSS community and commercial business networks that have somewhat disparate needs and values. We see that MySQL AB depends on the OSS community for its ecosystems and even more for the customer base, but they mostly conduct the product development in-house. However, the company also has made a significant contribution to the OSS community by licensing the database as

an OSS. Therefore, we define MySQL's approach toward the OSS community as a symbiotic one.

The Revenue Model

MySQL AB is often cited as the champion of the second generation of open source projects. These projects are open source but are directed by for-profit companies. The revenues of these corporations derive from selling consulting services for their products. MySQL AB makes MySQL available under the GPL for free and sells it under proprietary licenses for clients when the GPL is not an ideal option (e.g., inclusion of MySQL technology in a closed source product). Currently, MySQL AB receives more income from proprietary license sales than from its other income sources, branding, and services. Its main income seems to come from embedded commercial users (Välimäki, 2003). In terms of Hecker (1999), the revenue models of MySQL AB include features from support selling and dual licensing, both of which can be considered incarnations of the loss-leader model.

Red Hat

The U.S.-based Red Hat is one of the world's leading Linux software provider and one of the highest profile companies employing OSS in its business model. Red Hat's offerings resemble those of a classical software vendor: software distributed on CDs or over the Internet, deployment support, add-on products, and so forth. The unique aspect of the business model is that, for the most part, Red Hat has neither developed the software offering itself nor paid the development for suppliers. The role of Red Hat in its value network is related to its main activities in packaging, branding, and distributing the open source Linux operating system, thus making it usable for those who are not familiar with the ins and outs of the constantly evolving project.

Offering

Red Hat offers Linux and open source solutions into the mainstream by making high-quality, low-cost technology accessible (Rappa, 2005). In particular, Red Hat provides operating system software along with middleware, applications, and management solutions. In recent years, the target market has shifted mainly to corporate customers, thus influencing the heavy emphasis on enterprise Linux and network tools. Major parts of the software offering are provided under the GPL, which governs the redistribution of source code as well as monetary licensing rights for the binaries (Microsoft, 2005). In addition, Red Hat offers support, training, and consulting services to its customers worldwide and through top-tier partnerships. These services range from complete Linux migration to client-directed engineering to custom software development, especially in industry-specific solutions.

Resources

From the perspective of Red Hat's business model, it is obvious that key resources are related to brands and their development and management, as well as to marketing and business management. The funding provided by investors has enabled Red Hat to systematically develop these resources. In addition to marketing and management capabilities, relationships with OSS communities as the supplier network form a key resource in Red Hat's business model. Indeed, the company makes an extensive use of external resources for developing the software in its offering. The internal production resources include personnel and technology aimed at producing services.

Relationships

Red Hat has succeeded in establishing strong ties with large enterprise and academic custom-

ers such as Amazon.com, AOL, Merrill Lynch, Credit Suisse First Boston, DreamWorks, VeriSign, Reuters, and Morgan Stanley. In addition, its customer portfolio includes local, state, and federal governments in various countries. The company also maintains key industry relationships with hardware and middleware suppliers. In June 2002, Red Hat, Oracle, and Dell formally launched a combined Linux effort that includes joint development, support, and hardware and software certification. It was considered as an emphatic declaration in the strategy of Red Hat to focus on enterprise customers. Due to the inherent sharing nature of OSS, Red Hat considers balance as a key aspect in building a successful business without sacrificing customer trust, and in creating shareholder value without severing ties to the open source community.

Red Hat is gaining significantly from the software produced in the OSS community. It participates in OSS and Linux development by collaborating in standards creation as well as sponsoring the Fedora Project. According to the classification of Dahlander and Magnusson (2005) presented in the theoretical part of the study, the company's approach toward the OSS community could be defined as a symbiotic relationship, although the emphasis on enterprise customers embodies commensialistic elements.

The Revenue Model

Despite the release of software under the GPL-license mode, the services employed by Red Hat for commercial viability places a layer of restriction upon the binary and source code usage based on support contracts. This hybrid approach enables the company to provide OSS solutions in a commercial way (Microsoft, 2005). Thus, the primary revenue model is currently what Red Hat calls "subscriptions," which allows the company to effectively develop and deliver its technology based on customer feedback, as well as to provide support to customers over the life of an agreement. In terms of Hecker (1999), we identify this revenue model as support selling.

It has been claimed that this is a high-margin activity demanding only a little investment (Mantarov, 1999). On the other hand, little investment means lower entry barriers, and support offers a very weak basis for differentiation to gain sustainable competitive advantage. Microsoft clearly has nearly a monopoly on desktop operating systems, but its market share in services related to desktop operating systems is much smaller. Thus, there is potential for revenue models based on service provisioning, as in some OSS-based businesses.

CONCLUSION

This chapter aims at identifying the key determinants of OSS revenue model choices. On the basis of our literature review and through our case studies, we see that there are several motives for firms to participate and contribute to the OSS movement.

In this chapter, we identify three business model elements that affect firms' revenue model choices. These identified elements are offering, resources, and relationships. The type of offering in terms of the user environment and, thus, the target market of the software (private vs. enterprise applications and desktop vs. server applications) constrain the possibilities to form a revenue model. Furthermore, the licensing model affects the revenue model choice through defining the free and commercial components, as well as the use and further development terms and conditions of the software.

In addition to the type of offering, we argue that a firm's resources are an important factor affecting the revenue model. We see that the internal resources and capabilities of firms are essential determinants of the actor-driven devel-

Table 4. Summary of the cases

Business Model Elements	MySQL	Red Hat
Offering	Core offering embodies an in-house developed database software product and related services.	Operating system software maintenance and services along with operating system software.
Resources	The internal programming resources and professional management resources.	Resources related to the development and management of brands, as well as to marketing and business management.
Relationships	Balancing between the OSS community and commercial business networks that have somewhat disparate needs and values. Dependence on the OSS community mainly as a user community.	OSS community as significant product developer.
Revenue model	A majority of revenue originates from proprietary license sales, and a smaller proportion stems from other sources such as services. The main income seems to come from business users. The revenue model includes features from support selling and dual licensing, both of which can be considered incarnations of the loss-leader model.	The primary revenue model is currently what Red Hat calls "subscriptions," which allows the company to effectively develop and deliver its technology based on customer feedback, as well as to provide support to customers over the life of an agreement. The revenue model is identified as support selling.

opment activity in the collection and integration of divergent OSS components into commercial offerings. Our cases illustrate that they strongly enable and constrain the possibilities to collect revenue based on OSS. Furthermore, relationships between business actors and the OSS community form the essential external resource and capability base of the firm. The importance of relationship management is emphasized in balancing between the noncommercial culture of OSS communities and the for-profit business networks. The objectives and characteristics of these two networks differ in terms of the development of loyalty, trust, and motivation of actors into activities in which some actors may benefit economically.

The managerial implications of this chapter suggest that profit-seeking firms in the field of OSS must maintain a balance between their profit-oriented business objectives and the noncommercial principles of the OSS community. This is consistent with Dahlander and Magnusson (2005), who argue that an intention to control the community development may allow a firm to manipulate the development toward its strategic goals, but might also diminish the creativity and general interest of the community toward the project.

Our empirical observations from the two case examples indicate that the selection of the revenue model is dependent on other business model elements. The case of MySQL illustrates

that the need to maintain relationships with both the OSS community and the business network has led to a revenue model based on dual licensing. In this model, the community has access to the software for free, but business users may buy a software license for their commercial purposes. Furthermore, the dual-licensing model used by MySQL illustrates that a change in any of the elements of the identified key determinants may affect the revenue model choice. In this model, the company owns all copyrights to the software and, therefore, can license the software with two licenses, one allowing gathering of revenue from sold copies of the software and the other based on the principles of the loss-leader model.

The lesson learned from the Red Hat case is that internal resources (e.g., well-known brands) and superior commercialization capabilities allow a company to benefit from the development efforts of the OSS community. The business model of Red Hat is based on the ecosystem developing the core product collaboratively. The role of Red Hat in this collaboration is to deliver the results of the development work commercially added with service elements essential for the users of software.

REFERENCES

Ariño, A., & de la Torre, J. (1998). Learning from failure: Towards an evolutionary model of collaborative ventures. *Organization Science, 9,* 306-325.

Chetty, S. K., & Wilson, H. I. M. (2003). Collaborating with competitors to acquire resources. *International Business Review, 12,* 61-81.

Cravens, D. W. (1987). *Strategic marketing.* Homewood, IL: Richard D. Irwin, Inc.

Dahlander, L., & Magnusson, M.G. (2005). Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy, 34*(4), 481-493.

de Laat, P. B. (2005). Copyright or copyleft? An analysis of property regimes for software development. *Research Policy, 34*(10), 1511-1532.

Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development.* Pearson Education Limited.

Foss, N. J. (1999). Networks, capabilities, and competitive advantage. *Scandinavian Journal of Management, 15,* 1-16.

Gabrielsson, P. (2004). *Globalizing internationals: Product strategies of ICT companies.* Series A: 229, Helsinki: Helsinki School of Economics.

Gadde, L.-E., & Håkansson, H. (2001). *Supply network strategies.* Chichester, UK: Wiley.

Gulati, R., Nohria, N., & Zaheer, A. (2000). Strategic networks. *Strategic Management Journal, 21,* 203-215.

Håkansson, H., & Snehota, I. (1995). Analyzing business relationships. In D. Ford (Ed.), *Understanding business marketing and purchasing* (3rd ed., pp. 162-182). London: Thomson Learning.

Hart, S.L. (1995). A natural-resource-based view of the firm. *Academy of Management Review, 20,* 986-1014.

Hecker, F. (1999). Setting up shop: The business of open source software. *IEEE Software, 16*(1), 45-51. Retrieved August 24, 2000, from http://www.hecker.org/writings/setting-up-shop.html

Hedman, J., & Kalling, T. (2003). The business model concept: Theoretical underpinnings and empirical illustrations. *European Journal of Information Systems, 12,* 49-59.

Kotler, P., Armstrong, G., Saunders, J., & Wong, V. (1996). *Principles of marketing.* Hertfordshire, UK: Prentice Hall.

Lee, S. H. (1999). *Open source software licensing.* Retrieved February 14, 2006, from http://eon.law.harvard.edu/openlaw/gpl.pdf

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics, 50*(2), 197-234.

Mantarov, B. (1999). *Open source software as a new business model: The entry of Red Hats Software, Inc. on the operating system market with Linux*. Retrieved August 24, 2000, from http://www.lochnet.net/bozweb/academic/dissert.htm

Metcalfe, J. S., & James, A. (2000). Knowledge and capabilities: A new view of the firm. In N. J. Foss & P. L. Robertson (Eds.), *Resources, technology and strategy: Explorations in the resource-based perspective*. New York: Routledge.

Microsoft. (2005). Software licensing models. Retrieved March 23, 2006, from http://www.microsoft.com/resources/sharedsource/licensingbasics/licensingmodels.mspx

Möller, K., & Svahn, S. (2003). Managing strategic nets: A capability perspective. *Marketing Theory, 3*, 201-226.

Morris, M., Schindehutte, M, & Allen, J. (2005). The entrepreneur's business model: Toward a unified perspective. *Journal of Business Research, 58*, 726-735.

O'Mahony, S. (2003). Guarding the commons: How community managed software projects protect their work. *Research Policy, 32*(7), 1179-1198.

Open Source Initiative. (2004). *The open source definition* (Version 1.9). Retrieved March 5, 2004, from http://www.opensource.org/docs/definition.php

Osterwalder, A. (2004). *The business-model ontology: A proposition in design science approach*. Academic dissertation, Universite de Lausanne, Ecole des Hautes Etudes Commerciales, Lausanne, France.

Pateli, A. G., & Giaglis, G. M. (2004). A research framework for analysing eBusiness models. *European Journal of Information Systems, 13*(4).

Penrose, E. (1959). *The theory of the growth of the firm*. New York: Oxford University Press.

Perens, B. (1999). The open source definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution*. Sebastobol, CA: O'Reilly & Associates, Inc.

Prahalad, C. K., & Hamel, G. (1990). The core competence of the corporation. *Harvard Business Review, 32*, 79-91.

Rajala, R., Rossi, M.., & Tuunainen, V.K. (2003). Software vendor's business model dynamics case: TradeSys. *Annals of Cases on Information Technology, 5*, 538-549.

Rajala, R., & Westerlund, M. (In press). A business model perspective on knowledge-intensive services in the software industry. *International Journal of Technoentrepreneurship.*

Rappa, M. (2005). *Case study: Red Hat. Managing the digital enterprise*. Retrieved November 20, 2002, from http://digitalenterprise.org/cases/redhat.html

Raymond, E. S. (2001). *The cathedral and the bazaar: Musings on Linux and Open Source by an accidental revolutionary*. Sebastopol, CA: O'Reilly & Associates, Inc.

Rosenbröijer, C.-J. (1998). *Capability development in business networks*. Doctoral dissertation, Swedish School of Economics and Business Administration, Helsinki.

Rossi, M. A. (2004). *Decoding the "free/open source (F/OSS) software puzzle" a survey of theoretical and empirical contributions*. Quaderni, Siena: Dipartimento di Economica Politica, Università di Siena.

Sallinen, S. (2002). *Development of industrial software supplier firms in the ICT cluster.* Doctoral dissertation, University of Oulu, Oulu.

Seddon, P. B., & Lewis, G. P. (2003). Strategy and business models: What's the difference. In *Proceedings from the 7th Pacific Asia Conference on Information Systems*, Adelaide, South Australia (pp. 1-30).

Selznik, P. (1957). *Leadership in administration: A sociological interpretation*. New York: Harper Row.

Stallman, R. M. (1999). The GNU operating system and the free software movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution*. Sebastobol, CA: O'Reilly & Associates, Inc.

Stallman, R. M. (2002). *What is copyleft?* Retrieved November 20, 2002, from http://www.gnu.org/licenses/licenses.html

Timmers, P. (2003). Lessons from e-business models. *ZfB—Die Zukunft des Electronic Business, 1*, 121-140.

Välimäki, M. (2003). Dual licensing in open source software industry. *Systemes d´Information et Management, 8*(1), 63-75.

Välimäki, M. (2005). *The rise of open source licensing. A challenge to the use of intellectual property in the software industry.* Helsinki: Helsinki University of Technology.

Wernerfelt, B. (1984). A resource-based view of the firm. *Strategic Management Journal, 5*, 171-180.

KEY TERMS

Business Model: An abstraction of business, or the manifestation of strategy, that characterizes the business and specifies in which the company is positioned in its value-creating network.

Offering: An inseparable part of a business model that includes aspects such as complexity; the essential benefit that the customer is really buying; and product features, styling, quality, brand name, and packaging of the product offered for sale.

Relationships: The ties and interaction of companies with other actors.

Resources: Specific properties that are subordinate to the core competencies of companies.

Revenue Model: The method of value capturing that includes the description of the sources of revenue, price-quotation principle, and cost structure.

Software Licensing: The definition and agreement of rights to use, redistribute, or modify software.

Source Code: The programming that allows software to perform a particular function or operation.

ENDNOTE

[1] The adjective "free" refers to freedom, not price.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant & B. Still, pp. 541-554, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 6.17
Knowledge Management and Organizational Performance in the Egyptian Software Firms

Ahmed Seleim
Alexandria University, Egypt

Omar Khalil
Kuwait University, Kuwait

ABSTRACT

Management research has often overlooked the role of knowledge and knowledge management (KM) in the analysis of organizations and their performance. Also, the literature on KM is short of empirical evidence on the likely interrelatedness of the KM processes. This investigation adopted a research model and used data from 38 Egyptian software firms to examine: (1) the relationship of the KM processes of knowledge acquisition, documentation, transfer, creation, and application to organizational performance, and (2) a number of relationships within the KM processes themselves. The results suggest that knowledge application influences organizational performance, knowledge acquisition and knowledge creation influence knowledge application, and knowledge acquisition and knowledge transfer influence knowledge creation. Although they provide a limited support to the research mode, the results signify the value of continued examination and enhancement of such a model.

INTRODUCTION

Knowledge is an asset that needs to be effectively managed (Davenport & Prusak, 1998; Drucker, 1993). Interest in knowledge management (KM) has grown dramatically in the recent years, as more researchers and practitioners have become aware of the knowledge potential to drive innovation and improve performance (e.g., Cavaleri, 2004; Machlup, 1962, 1983; Penrose, 1959). For an organization to remain competitive, it must effectively practice the activities of creating, acquiring, documenting, transferring, and applying

knowledge in solving problems and exploiting opportunities (e.g., Sambamurthy & Subramani, 2005; Zack, 1999).

Argote and Ingram (2000) argue what the organization comes to know explains its performance. The ultimate test of any business concept, such as KM, is whether it improves business performance. If organizations cannot use knowledge to improve performance, knowledge does not have measurable value (Gorelick & Tantawy-Monsou, 2005). However, management research has often overlooked the role of knowledge and KM in the analysis of organizations and their performance (Nonaka, 1994; Spender, 1996). Most of KM research consists of either theoretical analyses of KM issues or case-based reviews of organizations' KM practices. Consequently, KM research is short of offering an unambiguous understanding of the role of KM in improving organizational performance (Kalling, 2003).

On the other hand, effective KM entails an understanding of the interrelationships that may exist among KM processes such as knowledge acquisition, knowledge creation, knowledge documentation, knowledge transfer, and knowledge application (e.g., Lee, Lee & Kang, 2005; Seleim, Ashour & Khalil, 2005a). These processes are not necessarily sequential but rather iterative and overlapping (Holsapple & Joshi, 1999, p. 7-1; Lee & Choi, 2003). Furthermore, an analysis for the purpose of understanding the relationship of the KM processes to organizational performance is incomplete if it does not also include the analysis of the interrelationships among the KM dimensions themselves. In other words, effective KM requires an understanding of the direct and indirect influence of KM processes on organizational performance.

Thus far, the literature on KM is short of a cohesive theoretical framework that provides a structure that can be used to understand how to enhance KM within an organization as well as to assess the potential impact of KM efforts on organizational effectiveness (Hoffman, Hoelscher & Sherif, 2005). This research proposes and tests a model to explore the interrelationships among the KM processes as well as the relationship of the KM processes to organizational performance in the Egyptian software industry. The exploration of the possible interrelationships among the KM processes is believed to be an extension to the absorptive capacity theory (Cohen & Levinthal, 1990), which attends to the organizational processes and activities by which organizations acquire, absorb, transfer, and exploit organizational knowledge, and the SECI— socialization, externalization, combination, and internalization—model of knowledge creation (Nonaka & Takeuchi, 1995).

The article is organized accordingly. A background on KM is presented first, followed by the research method, research results, implications, and conclusions.

BACKGROUND

KM Processes

KM is a natural function in human organizations. Gorelick and Tantawy-Monsou (2005) view KM as a system or framework that integrates people, processes and technology to achieve sustainable results by increasing performance through learning. Therefore, effective KM requires viewing knowledge as a process rather than a resource (e.g., Alavi & Leidner, 2001; Davenport & Prusak, 1998; Lee & Choi, 2003; Spender, 1996).

Researchers have adopted different views of what the KM process entails. Johnston and Blumentritt (1998), for example, define the KM process to comprise knowledge identification, acquisition, generation, validation, capture, diffusion, embodiment, realization, and use. Zack (1999) asserts that KM includes the acquisition, refinement, storage, retrieval, distribution, and presentation of knowledge. Bennett and Gabriel (1999) view KM as a process that involves the cap-

ture, storage (i.e., documentation), dissemination, and use of knowledge. Also, Gold, Malhotra, and Segars (2001) view KM as a knowledge process capability that consists of knowledge acquisition, conversion, application, and protection. More recently, Salojarvi, Furu, and Sveiby (2005) view KM as a perspective on the management of the firm as a whole that encompasses activities in all relevant managerial areas.

Based on the aforementioned varying views of KM, these researchers classify the five fundamental dimensions of the KM process as knowledge acquisition (KA), knowledge creation (KC), knowledge documentation (KD), knowledge transfer (KT), and knowledge application (KAP). These five KM processes are likely interdependent (e.g., Darroch, 2003; Lee et al., 2005). Successful KM systems, which can be IT or non-IT based and aim to manage organizational knowledge, should support the performance of these KM processes (Alavi & Leidner, 2001; Edward, Feng & Liou, 2005; Jennex & Olfman, 2005). Moreover, organizational outcomes are a function of how well organizations can use, apply, and leverage what they know.

KM and Organizational Performance

The conceptualization of the relationship between KM and organizational performance can be traced back to the resource-based theory of the firm, the knowledge based theory of the firm, and the organizational learning theory. These schools of thought view organizations as systems for creating, using, and distributing knowledge (e.g., Grant, 1996; Spender, 1996); the success/performance of a firm is attributable to superior capabilities in knowledge processing (e.g., Hedlund, 1994; Reinmoreller & Chong, 2002).

According to the resource-based theory of the firm, the possession of unique resources such as knowledge within organizations is an important source of competitive advantage (Barney, 1991; Penrose, 1959; Prahalad & Hamel, 1990). Also, the knowledge-based theory of the firm focuses on applied knowledge as the most strategically significant resource of the firm. It postulates that the primary rationale for the firm is to produce, create, and transfer knowledge (e.g., Bierly & Chakrabarti, 1996; Choi & Lee, 2003; Demsetz, 1991; Grant, 1996; Nonaka, 1994; Nonaka & Takeuchi, 1995; Spender, 1994). Consequently, differences in the firms' performances are attributed to differences in the rules that the firms apply to knowledge creation, development, distribution, and usage.

Similarly, the organizational learning theory emphasizes the need for an organization to build its learning capabilities. Organizational learning is the process by which firms develop new knowledge and insights (e.g., Davenport & Prusak 1998; Garvin, 1994; Huber, 1991; Tippins & Sohi, 2003). From this perspective, KM is viewed as a process of capturing an organization's knowledge and using it to foster innovation through organizational learning (Nonaka, 1994; Nonaka & Takeuchi, 1995).

Scholars argue that the acquisition, creation, documentation/codification, transfer, and application of knowledge are the keys to achieving competitive advantage (e.g., Argote & Ingram, 2000; Connor & Prahalad, 1996; Darroch & McNaughton, 2002; Davenport & Prusak, 1998; Grant, 1996; Powell & Dent-Micallef, 1997; Rastogi, 2002; Spender, 1994), to improving organizational performance (e.g., Bassie, 1997; Teece, 1998), and to fostering innovation (e.g., Antonelli, 1999; Nonaka & Takeuchi, 1995). Holsapple and Jones (2004, 2005) propose a knowledge chain model that links KM to organizational performance. The model identifies five primary activities (acquisition, selection, generation, assimilation, and emission) and four secondary activities (leadership, coordination, control, and measurement) as focal points in improving organizational competitiveness. However, the model has yet to be empirically validated.

A number of researchers investigated the relationship between KM and different performance outcome measures. Gold et al. (2001) reported that KM capability, including knowledge acquisition, conversion, application, and protection are related to organizational effectiveness. Wiig (2002) empirically investigated how KM enhances decision making in public services, aids the public in participating effectively in decision making, builds competitive societal intellectual capital capabilities, and develops a knowledge management work force. Wiig's findings suggest the importance of having a comprehensive KM function in support of a public administration that allows the society to prosper and increase its viability by making its people and institutions work smarter.

Jennex and Olfman (2002) found information systems that incorporated organizational memory and knowledge improve productivity. Kremp and Mairesse (2004) found KM to influence a firm's innovative performance and productivity. Also, Liu, Chen, and Tasi (2004) found a positive and significant correlation between KM capability factors—for example, obtaining knowledge, refining knowledge, storing knowledge, and sharing knowledge—and competitiveness. Cavaleri (2004) underscores that reports of major cost reductions and performance increases as a result of KM initiatives have helped KM gain the attention of managers worldwide.

More recently, Edward et al. (2005) found that the adopters of KM systems gain a competitive advantage over the nonadopters. Salojarvi et al. (2005) examined the relationship between sustainable sales growth and KM activities in Finnish small and medium-sized enterprises, and found higher levels of KM maturity to be positively correlated with long-term sustainable growth. Lee et al. (2005) found a significant correlation between the KM activities of knowledge creation, accumulation, sharing, utilization, and internalization and three financial performance measures. In addition, Zhang, Tian, and Qi (2006) suggest a conceptual model describing the impact of organizational memory on organizational performance. The model was tested using data from Chinese manufacturing companies, and the results confirmed that organizational memory had a strong positive impact on organizational performance.

The Interrelationships among KM Processes

An understanding of the interrelationships among the KM processes is essential to the discovery of knowledge gaps, unused knowledge, and the development of knowledge maps in organizations (Liebowitz, 2005). This understanding helps organizations avoid randomized and unsystematic knowledge processes so that organizational knowledge is consistently and systematically developed (Ichijo, Krogh & Nonaka, 1998, p. 197). In addition, the coordination of KM dimensions is critical, since the shortage of any dimension may result in less than optimum outcomes of KM efforts (Darroch, 2003). Such coordination, however, requires a thorough understanding of the strength and direction of the relationships among the KM processes from a holistic perspective (Lee & Choi, 2003).

A number of scholars have conceptually deliberated on the interrelatedness of the KM processes. KM is viewed as a cross-functional, multifaceted phenomenon (Lee & Choi, 2003), and therefore, KM processes appear to be interconnected. Each KM process seems to depend on one or more of the other processes (Darroch, 2003). Beckman (1999, pp. 1-7) asserts that the KM processes often work simultaneously rather than sequentially.

Gold et al. (2001) present an integrative view of KM as a one knowledge process capability, which implies that the KM processes are complementary. For example, once knowledge is acquired, it needs to be documented in knowledge repositories in order to be transferred/shared, combined with

other knowledge to create new knowledge, and applied. Also, documented knowledge constitutes an important input for creating new knowledge and facilitating knowledge transfer, which, in turn, fosters knowledge creation and application. Furthermore, knowledge transfer and knowledge creation are critical elements and inputs for successful knowledge application.

Besides, the SECI model of knowledge creation (Nonaka & Takeuchi, 1995), which focuses on socialization, externalization, combination, and internalization, can provide another theme to establish the interrelationships among the KM processes. The common factor of the four processes of the model is knowledge sharing. Socialization is the process of converting new tacit knowledge through shared experience that allows non communicated knowledge to be communicated (Nonaka & Takeuchi, 1995); therefore, it involves knowledge transfer. Externalization is the process of articulating tacit knowledge into explicit knowledge that can be shared with others.

Combination and internalization are the processes of converging explicit knowledge into more complex and systematic sets of explicit knowledge. In these processes, knowledge is exchanged and reconfigured through documents, meetings, or communication networks, and the focus is on knowledge utilization (Nonaka & Reinmoeller, 2000, p. 90). Lee and Choi (2003) add that the SECI model includes not only knowledge creation but also knowledge transfer. Knowledge transfer, which is enabled by knowledge acquisition and documentation, is seen as a key element towards a better understanding of the knowledge creation process (Marr, Gupta, Pike & Roos, 2003). Also, transferring and sharing of knowledge from various sources is essential for knowledge application.

On the other hand, Alavi and Leidner (2001) argue that knowledge creation and codification do not necessarily lead to improved performance, nor do they create value. Value is created only when knowledge is located and transferred from its previous site and applied where it is needed. The creation, codification, and storage of new knowledge without its exploitation (e.g., application) lead to its underutilization as a driver of performance (Alavi & Tiwana, 2000).

Empirically, however, only few investigations aimed at exploring the interrelatedness of KM processes. Liu et al. (2004) found a significant correlation among all knowledge management capability factors of obtaining, refining, storing, and sharing knowledge. Faraj and Sproull (2000), for example, investigated the importance of coordination of expertise knowledge in 69 software development teams, and the results revealed that expertise coordination correlated strongly with team performance. Syed-Ikhsan and Rowland (2004) found knowledge that was created and stored on paper and electronic documentation to be positively related to knowledge transfer. They also found a negative but nonsignificant relationship between the status of knowledge documentation and the accessibility of knowledge assets.

To summarize, the review of the existing KM literature reveals that the direct relationship between KM processes and organizational performance has yet to be empirically explored and validated (Lee & Choi, 2003). In addition, the literature on KM is short of empirical evidence on the probable interrelatedness of the KM processes. A significant number of the cited relationships within KM processes and between KM processes and organizational performance are hypothetical. Therefore, the validation of such relationships is deemed important. Empirical evidence on the strength and direction of the interrelationships among KM processes and between KM processes and organizational performance should enhance our understanding of the KM dynamics, enable knowledge managers to justify resources allocation to KM, and achieve the required effective integration of KM processes.

RESEARCH METHOD

The Research Model

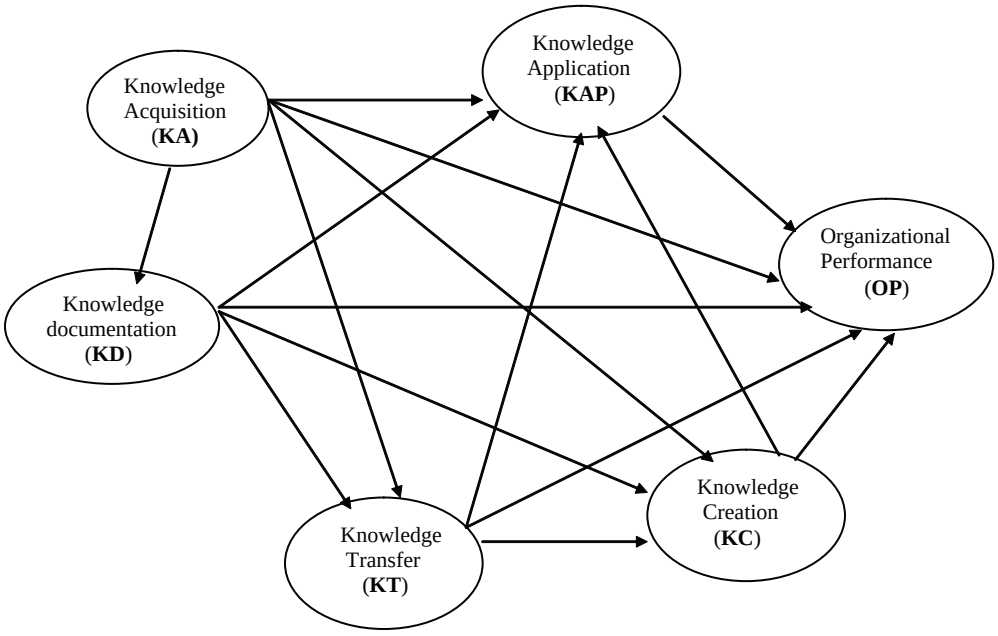
Figure 1 depicts the research model used in this study, including the research variables and the hypothesized relationships. The model is drawn based on the relevant KM literature reviewed in the previous section. The research model depicts two groups of relationships: (1) relationships between the five KM processes (KA, KD, KT, KC, and KAP) and organizational performance (OP), and (2) relationships among the five KM processes themselves. The model also implies indirect relationships between the KM processes and organizational performance.

Research Hypotheses

The KM Processes and OP Hypotheses (H1-H5)

1. KA and OP. KA was found to be related to innovation (Darroch & McNaughton, 2002). In addition, knowledge strategies (e.g., explorers, exploiters, loners, and innovators) were reported to influence organizational performance (Bierly & Chakrabarti, 1996; Pablos, 2002), where innovators and explorers tended to be more profitable than loners. In the international business field, Lyes and Salk (1996) found a significant relationship between KA and performance indicators.

Figure 1. The research model



Makino and Delios (1996) found that partnering with local firms could be a primary strategy for accessing local knowledge and improving joint venture performance.

KA activities—for example, active participation in professional networks or associations, regular attendance of courses, seminars or other training programs, collection of information about the needs of customers, hiring of consultants and new staff members, and so forth–are expected to enhance organizational performance. This expectation is formulated in the following hypothesis:

H1. KA positively influences OP.

2. KD and OP. KM research emphasizes KD as a critical KM process. Staff turnover is frequently cited as the main cause of the institutional memory loss (e.g., Mason & Pauleen, 2003), and the loss of organizational tacit knowledge (Stovel & Bontis, 2002). Therefore, KD is used as an important strategy to manage through the turnover dilemma, which could influence organizational performance (Hansen, Nohria & Tierney, 1999).

   Skandia Insurance Company, for example, was able to set up its office in Mexico in six months using formerly acquired and documented knowledge, whereas the task was expected to take seven years (Loermans, 2002). Documented knowledge is one form of the knowledge storage bins in the organizational memory that enables firms to avoid mistakes and capitalize on past experience (Walsh & Ungson, 1991). Therefore, KD is expected to be related to OP. This expectation is formulated in the following hypothesis:

H2. KD positively influences OP.

3. KT and OP. Wagner (2003) argues for the importance of knowledge transfer as a KM process. The foundation of knowledge transfer can be traced back to the past research on technology transfer, research and development (R&D) collaboration (Amabile, Nasco, Mueller, Wojcik, Odomirok, Marsh, et al. 2001), and strategic alliances (Borys & Jemison, 1989; Das, Sen & Sengupta, 2003). Such research provides evidence of the benefits of knowledge transfer. Bloodgood and Salisbury (2001) posit that knowledge transfer is beneficial to the organizations. Teece (2000) asserts that knowledge which is inside the minds of employees, the files, databases, and the like is of little value if it is not transferred to the right people at the right time. Also, Sveiby (2001) argues that knowledge transfer between individuals does not only benefit the organization but also improves the competences of the individuals that are involved in the process of knowledge transfer.

   Darr, Argote, and Epple (1995) found that knowledge transfer between the organizational units within the same corporation was essential to increased productivity. Tsai (2001) investigated the influence of knowledge sharing within organizations on outcome variables, and found that the absorptive capacity improved innovation and performance. Andersen Consulting saves millions of dollars a year in FedEx bills alone by using an intranet and other knowledge sharing tools (Stewart, 1997). Nelson and Cooprider (1996) found that increasing the level of shared knowledge between the information systems (IS) group and line groups improves IS performance. The ability to transfer and diffuse knowledge is expected to improve OP. This expectation is formulated in the following hypothesis:

H3. KT positively influences OP.

4. KC and OP. KC has become a critical element for organizational competitive advantage (Alavi & Leidner, 2001; Nonaka & Takeuchi, 1995; Syed-Ikhsan & Rowland, 2004). Linderman, Zaheer, Leidtke, and Choo (2004) speculate that quality management practices create knowledge that, in turn, contributes to organizational performance. Hussi (2004) adds that successful companies create sustainable value through the creation of knowledge. Kim and Kogut (1996) indicate that the capability of a company to improve its products relies on the accumulated knowledge from past experience, research, and experimentation.

   In addition, KC is perceived as an important element to new products development (Camelo-Ordaz, Fernandez-Alles, Martín-Alcázar, Romero-Fernández & Cabrera, 2004; Kim & Kogut, 1996), and an essential factor for the continuity and survival of the organization (Nonaka, Byosiere & Konno, 1994; Sharkie, 2003). Empirically, Lee and Choi (2003) found that KC was positively related to organizational creativity, which, in turn, was positively related to organizational performance. Therefore, organizations are expected to improve their performance outcomes through KC activities. This expectation is formulated in the following hypothesis:

H4. KC positively influences OP.

5. KAP and OP. The knowledge-based theory of the firm focuses on applied knowledge as the most strategically significant resource of the firm. In KM, emphasis should be on the application of knowledge, not merely on the acquisition and protection of such knowledge (Grant, 1996; Spender, 1994; Teece, 1998; Wiig, 1997). Demarest (1997) states that the application of knowledge should be the ultimate goal of any effective KM system. A firm that treats knowledge as a by-product of its operations rather than a purpose of those operations, especially in the knowledge intensive firms, will fail, not because it could not create knowledge, but because it could not apply it to create market-valued delivery in the marketplace.

   Hasan and Al-hawari (2003) speculate on the strong relationship between KM and organizational performance through making knowledge applicable to all organizational activities. Lopez, Peon, and Ordas (2004) add that the firm's ability to create and put knowledge into practice is a principal capacity in order to attain a sustainable competitive advantage. In addition, effective reuse of knowledge is frequently cited as an organizational concern that is related to organizational effectiveness (Markus, 2001). Therefore, organizations that emphasize the strategic value of managing the application of their knowledge (Bartlett & Ghoshal, 2000), and systematically apply their knowledge, are expected to capitalize on their performance. This expectation is formulated in the following hypothesis:

H5. KAP positively influences OP.

The KM Processes-Interrelationships Hypotheses (H6-H9)

The investigation of the interrelationships among KM processes is an extension to the absorptive capacity theory (Cohen & Levinthal, 1990) that stresses the organizational processes and activities, by which organizations acquire, absorb, transfer, and exploit organizational knowledge. The KM processes often work simultaneously rather than sequentially (Beckman, 1999, p. 1-7; Holsapple & Joshi, 1999, p. 7-1; Lee & Choi, 2003), are likely interrelated, and the ineffectiveness of

one or more of the processes may turn KM ineffective (Darroch, 2003).

The existing KM literature appears to be short of well-defined and broadly acceptable theories and frameworks that guide the formulation of hypotheses describing possible interrelationships among the KM processes. As a result, the researchers propose the following four hypotheses (*H6*-*H9*) in order to explore four sets of interrelationships among the KM processes.

6. The relationship of KA to KD. Once knowledge is acquired, it needs to be documented in knowledge repositories so that it can be transferred/shared and combined with other knowledge to create new knowledge (e.g., Nonaka & Takeuchi, 1995). When knowledge is acquired and becomes available (KA), it is expected to be codified and documented (KD) in order to be transferred and reused. This expectation is formulated in the following hypothesis:

H6. KA positively influences KD.

7. The relationship of KA and KD to KT. For knowledge to be transferred and shared (KT) by people and organizational units, the knowledge must exist (KA) in a form (KD) that facilitates its transfer. Syed-Ikhsan and Rowland (2004) found the knowledge that was acquired and documented positively related to KT. Therefore, KA and KD are believed to be important processes for KT. This expectation is formulated in the following hypothesis:

H7. KA and KD positively influence KT.

8. The relationship of KA, KD, and KT to KC. Knowledge cannot be created in a vacuum. To create new knowledge (KC) through experience, research, and experimentation, prior knowledge (KA) must be available in a form (KD) that can be transferred and shared (KT) by those who are responsible for performing tasks and/or conducting research. Nonaka and Takeuchi's (1995) SECI model of knowledge creation establishes the interrelationships among the KM processes. The common factor of the four processes of the model is KT (Lee & Choi, 2003), which is seen as a key element towards a better understanding of the knowledge creation process (Marr et al., 2003). The expected relationships of KA, KD, and KT to KC are formulated in the following hypothesis:

H8. KA, KD, and KT positively influence KC.

9. The relationship of KA, KD, KT, and KC to KAP. Knowledge application is the ultimate goal of KM. For knowledge to be used and applied (KAP), it must be acquired (KA) and/or created (KC), documented (KD), and transferred (KT). Value is created only when knowledge is located, transferred, and applied where it is needed (Alavi & Tiwana, 2000). Gold et al. (2001) add that knowledge transfer and knowledge creation are critical elements and inputs for successful knowledge application. Documented knowledge constitutes an important input for creating new knowledge and facilitating knowledge transfer, which, in turn, fosters knowledge creation and application. The expected relationships of KA, KD, KT, and KC to KAP are formulated in the following hypothesis:

H9. KA, KD, KT, and KC positively influence KAP.

Research Variables

a. Organizational performance (OP). OP is the dependent variable in the research model, and refers to an organization-level performance relative to the competition.

b. KM variables. The five KM variables are defined as follow:
   1. KA includes activities that select and acquire external knowledge (e.g., via journals, magazines, books, Web sites, professional meetings, etc.).
   2. KD involves activates that institutionalize knowledge in the form of an organizational memory so that it can be transferred and reused in the future.
   3. KT includes activities that enable the exchange of knowledge between individuals, groups, organizational units, and at the different organizational levels.
   4. KC comprises activities that develop and create insights, skills, and relationships in the organization as well as the generation of internal knowledge (e.g., through experience, experimentation, special groups such as R&D, etc.).
   5. KAP refers to the organization's use of the available knowledge in order to improve its processes, products and services, and organizational performance.

Measurement

1. OP. Measurement of organizational performance is one of the most difficult issues that researchers have to deal with. Performance measures can be obtained from either primary or secondary sources. Also, these measures can be objective or subjective. Subjective or indirect measures for OP may represent an adequate alternative for objective or direct measures, especially when the financial data are unavailable (Dess, 1987; Dess & Robinson, 1984; Powell, 1992; Powell & Dent-Micallef, 1997; Spanson & Lioukas, 2001; Tippins & Sohi, 2003).
   In addition, Venkatraman and Ramanujam (1987) demonstrate that managers tend to be less biased in their assessments of their organizational performance than what researchers had believed. They state that perceptual data from senior managers can be used as acceptable criteria for organizational performance. Other researchers found the measurements of perceived organizational performance to have positive and significant correlations with objective measures of financial performance (Delaney & Huselid, 1996; Dollinger & Golden, 1992; Hansen & Wernerfelt, 1989; Powell, 1992).
   Because objective measures of OP were unavailable in the investigated firms, the researchers had to use a self-reported item that asked each respondent to indicate the performance of the respondent's firm relative to the competitors in the software market. The question used a five-point Likert scale, whereas 1 = very low and 5 = very high.
2. The KM variables. A generally accepted measuring instrument for KM constructs is currently lacking. The scale used in this investigation is a modified version of the original survey of Filius, de Jong, and Roelofs (2000). The items that measure the five KM constructs use a 5-point Likert-type scale (1 = very low practice and 5 = very high practice). A number of items have been added in order to strengthen the original survey and fit it to the Egyptian context.
   In the course of modifying the instrument, one item (item 3, knowledge gap diagnostic) has been adopted from Hall and Andriani (2002) and added to the KA items. Another item (item 15, use of knowledge maps and networks) has been added to the KD items. In addition, four items (items 17, 18, 20, and 25) have been added to the KT part of the instrument. These four items include statements regarding extraction of experience and knowledge of experts, keeping knowledge and know-how away from others in the firm,

distributing knowledge in formal ways, and using methods and mechanisms for knowledge exchange among employees.

Furthermore, four items (items 30, 33, 35, and 36) have been added to modify the KC part. The added items cover the existing idiosyncratic knowledge resources in the firm, development of important knowledge at the industry level, conducting benchmarking analysis, and data mining. In addition, four items (items 43, 44, 45, and 46) have been added to strengthen the KAP section of the instrument. The four items address knowledge integration, leveraging and improving knowledge usage, discovering problems that cause gaps between targets and achievements, and the ability of the firm to use its knowledge in different projects.

The instrument was tested and revised a number of times in order to fit the context of the study. The final instrument (in Arabic) includes two sets of questions. The first set consists of 46 items to measure KM variables. The second set comprises questions that are designed to gather data on each firm's performance and the firm's background such as age, number of software developers, and the number of other employees in the firm. Appendix A includes an English copy of the adopted KM survey.

Sampling

A cross-sectional field survey strategy was chosen for this investigation. This investigation focused on the software companies in the Egyptian private sector as the primary population. Software firms were selected because of the relevance and usefulness of KM practice in software industry, which is a knowledge intensive industry (Mathiassen & Pourkomeylian, 2003). In addition, the software development in Egypt is a promising industry. With more than 200 software firms and over 2,000 programmers, Cairo, Egypt has established itself as the software development hub of the Arab world (Albrecht, 1996). In the year 2000, the software exports were estimated at $50 million, and expected to eventually reach as high as $600 million within a year (Howeidy, 1999). Moreover, Egypt has thousands of qualified software professionals and has become a major supplier of software products, services, and software developers to its neighboring Arab countries.

The research population consisted of 107 software companies which were members in the Egyptian Chamber of Software Industry. Upon an initial contact with these firms, which are located in the Cairo and Alexandria areas, 38 firms agreed to participate in this investigation. The firms' CEOs or their representatives were asked to answer the survey questions in a structured interview setting.

The research sample (n = 38) represents 35.5% of the total population. Most of the firms in the entire sample are relatively small, with averages of 62 total employees and approximately 26 software developers. The largest firm has 300 employees, 80 of whom are software developers, and the smallest firm has only 5 employees, 2 of whom are software developers. In addition, the firms in the sample are relatively young (the average age in the sample is less than eight years), with the oldest firm only seventeen years old, and the youngest two years old. However, the relatively high standard deviations suggest wide dispersions in the firms' number of employees and ages.

Tables 1 and 2 summarize the respondents' characteristics, including level and type of education and tenure in the firm and in the software field.

Reliability and Validity

The reliability of the KM scales was assessed by estimating the latent variables' composite scale reliability, which is a measure of internal consistency reliability analogous to Cronbach's alpha. Table 3 shows that all the composite scale

Table 1. The respondents' educational characteristics (N = 38)

Respondents' Characteristics	Frequency	Percentage
Level of Education:		
Graduate	7	18.4
Undergraduate	31	81.6
Total	38	100
Educational background:		
Engineering	18	47.4
Business	13	34.2
Computer and Information	4	10.5
Science	2	5.3
Other	1	2.6
Total	38	100

Table 2. The respondents' experiences

Years of experiences	Minimum	Maximum	Average	Standard Deviation
Tenure in the firm	1	22	5.28	4.88
Tenure in the industry/ field	1	32	10.86	7.36

reliability coefficients for the five KM variables were higher than .70, the criterion recommended by Fornell and Larcker (1981). Fornell and Larcker (1981) also recommend the use of average variance extracted (AVE) by the latent variables from the observed or manifest variables as another index of internal consistency reliability. In Table 3, the AVEs for the five KM variables (KA, KD, KT, KC, and KAP) are .468, 486, .447, .527, and .518, respectively. The AVEs are either greater than the recommended 0.5 (House, Spangler & Woycke, 1991) or close to it. In addition, the factor loading of the indicators associated with the five KM variables are all higher than the recommended level (0.55) for adequate practical significance (Hair, Anderson, Tatham & Black, 1998, p. 111).

This procedure has lead to the exclusion of 17 items in order to improve the reliability and validity of the KM instrument (Table 3). The final scale consists of 29 items that measure the five KM constructs.

To assess its content and face validity, the instrument was pretested and refined several times as a result of feedback received from a number of experts in software development in Egypt as well as from a number of academicians at Alexandria University, who were asked to comment on the clarity and the relevance of the measures. After

the preliminary test, a pilot study was conducted with the participation of six software firms in order to evaluate the instrument and its administrative procedures.

The convergent validity was assessed by examining the factor loading of each manifest variable on its latent variable. If the manifest variables load high on the latent variable, there is evidence for

Table 3. Internal consistency reliability and factor loadings for the research variables

Constructs and Items	**Factor Loading**	**Composite Scale Reliability**	**Average Variance Extracted**
Knowledge acquisition (KA)			
2	0.7082		
3	0.7788	.814	.468
5	0.6368		
6	0.6646		
8	0.6201		
Knowledge documentation (KD)			
9	0.6486		
12	0.6247	0.789	0.486
14	0.6907		
15	0.8090		
Knowledge transfer (KT)			
16	0.5928		
20	0.6248		
21	0.7126	.827	.447
22	0.8324		
23	0.5933		
24	0.6230		
Knowledge creation (KC)			
30	0.6186		
31	0.6191	0.868	
33	0.6574		0.527
34	0.7706		
35	0.8481		
36	0.8069		
Knowledge application (KAP)			
37	0.5103		
38	0.6597		
39	0.7440		
40	0.7749	0.894	0.518
41	0.6976		
42	0.7675		
43	0.7823		
44	0.7799		
Organizational performance (OP)	0.9950	1	1

the convergent validity of the measure (e.g., House et al., 1991). As shown in Table 3, all observed variables had factor loadings on their respective latent variables greater that .5, which has been traditionally accepted in principle component factor analysis and exploratory factor analysis.

The discriminant validity of the measures was examined following the recommendation of House et al. (1991); discriminant validity exists if the correlation between two composite constructs is not higher than their respective reliability estimate. Using this criterion, the comparison between the reliability estimates (Cronbach's alpha) for the research variables in Table 3 (.814, .789, .827, .868, .894, and 1, respectively), and the correlations among the variables (Table 4) shows that the correlation coefficient are not higher than the reliability estimates.

Discriminant validity can also be demonstrated if the square root of the average variance extracted in the measurement model is higher than the correlation between the construct and the other constructs in the correlation matrix (Fomell & Larcker, 1981). The diagonal elements are the square roots of the average variance extracted. Off-diagonal elements are the correlations between constructs. In Table 5, the square root of the AVEs for KA, KD, KT, KC, KAP, and OP are .684, .697, .668, .725, .719, and 1, respectively.

While KD and KT satisfy the discriminant validity criterion, the AVEs for KA, KC, and KAP are slightly lower than their correlations with

Table 4. The correlation matrix of the research variables

The Variables	KA	KD	KT	KC	KAP	OP
1. KA	1.000					
2. KD	.280	1.000				
3. KT	.393*	.212	1.000			
4. KC	.684**	.343*	.531**	1.000		
5. KAP	.706**	.282	.575**	.750**	1.000	
6. OP	.065	.331*	.168	.073	.255	1.000

**p <.05; **p < .01*

Table 5. Discriminant validity of the latent variables in substantive model

Con-structs	KA	KD	KT	KC	KAP	OP
1. KA	.684					
2. KD	.280	.697				
3. KT	.393	.212	.668			
4. KC	.684	.343	.531	.725		
5. KAP	.706	.282	.575	.750	.719	
6. OP	.065	.331	.168	.073	.255	1.000

some of the other KM variables in their respective rows and columns. Based on the results of the two discriminant validity tests, we may conclude that the KM measures used in this investigation have adequate discriminant validity.

RESULTS

Descriptive Statistics

Table 6 display a summary description of the mean, standard deviation, maximum value, and minimum value for each of the five KM variables and the organizational performance (OP) variable. The results indicate that the respondents generally agree that their firms moderately practice the five KM processes, and their practice of KD (mean = 3.51) and KAP (mean = 3.51) is slightly higher than their practice of KA (mean = 3.19). In addition, KC practice (standard deviation = .66) seems to be the most dispersed and KT practice (standard deviation = .48) to be the least dispersed practice in the investigated firms.

On the other hand, the respondents feel that, on average, their firms perform well (mean = 4.18), compared to the competitors, and the performance scores seem to slightly fluctuate from one firm to another (standard deviation = .69).

Testing the Research Hypotheses

The partial least squares (PLS) method was used to test the hypothesized relationships in the research model (Figure 1). PLS is a technique for estimating path models involving latent variables indirectly observed by multiple indicators (Fornell & Cha, 1994). It maximizes the explanatory power of a conceptual model by examining the R^2 values for the dependent (endogenous) variables (Hulland, 1999). PLS was chosen in the present study because it does not require multivariate normal data, and it is appropriate for the analysis of a small sample size (e.g., House et al., 1991).

Testing the Hypotheses on the Relationships of KM Processes to OP

Figure 2 displays the final research model in a structural equation map. The PLS results show that the KM processes explain 19.7% of the variance in OP, the endogenous variable. The explanatory power of the model is relatively reasonable for such new measures and phenomenon. This result also supports the significance of the KM processes to organizational performance.

The examination of the effect of each KM process on OP (Table 7) reveals that only the path from KAP to OP (*H5*) has a positive and significant

Table 6. Descriptive statistics for the research variables

Research Variables	N	Minimum	Maximum	Mean	Standard Deviation
KA	38	1.86	4.29	3.1955	.5592
KD	38	1.57	4.43	3.5075	.5404
KT	38	2.27	4.27	3.3852	.4761
KC	38	1.73	5.00	3.3876	.6616
KAP	38	2.00	4.80	3.5053	.5927
OP	38	3.00	5.00	4.1842	.6919

Figure 2. The research model results

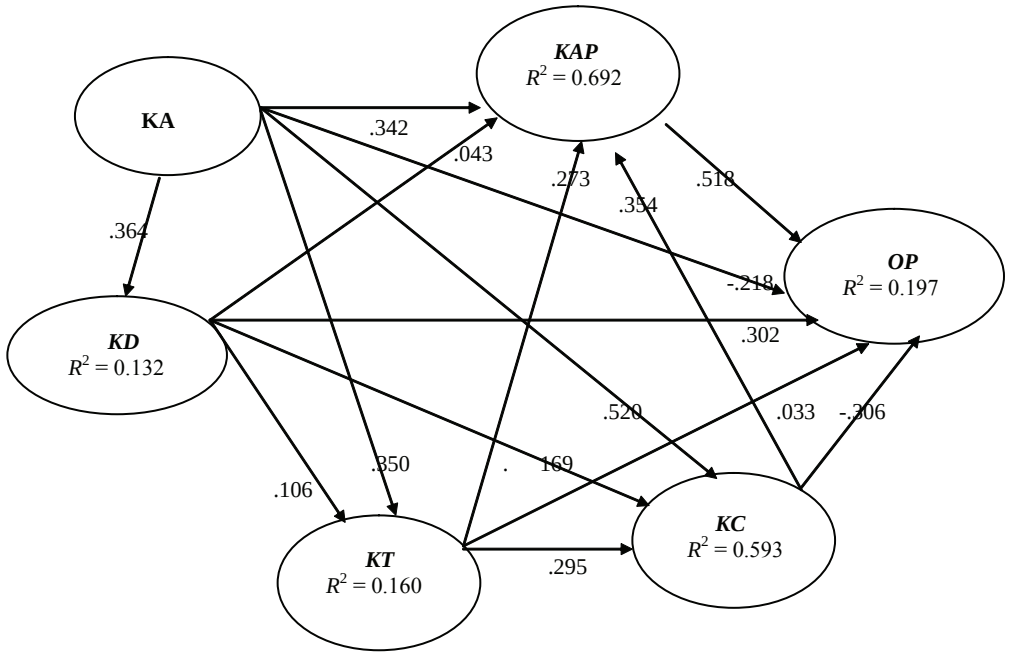


Table 7. The relationships of KM processes to OP

Dependent variable	Independent Variables	R^2	Hypotheses
OP	All KM variables	.197	*H1-H5*

Dependent Variables	Independent Variables	Hypothesis Number	Predicted Path Coefficient	Standardized Path Coefficient
OP	KA	*H1*	−	.218 T =. 8106
	KD	*H2*	+	.302 T = 1.1106
	KT	*H3*	+	.033 T = .1764
	KC	*H4*	−	-.306 T = .8784
	KAP	*H5*	+	.518 T = 2.1185

beta coefficient of .618 (T = 2.1185). These results suggest that, since KAP influences OP, there must be processes and systems that facilitate knowledge application in order to enhance organizational performance. Accordingly, the results support the acceptance of *H5*.

The other paths from KA, KD, KT, and KC to OP are not significant. There is a negative but insignificant beta coefficient of .218 (T = .8106) between KA and OP (*H1*), a positive but insignificant beta coefficient of .302 (T = 1.11106) between KD and OP (*H2*), a positive but insignificant beta coefficient of .033 (T = .1764) between KT and OP (*H3*), and a negative but insignificant beta coefficient of -.306 (T = .8748) between KC and OP (*H4*).

These nonsignificant paths suggest that KA, KT, and KC do not directly influence organizational performance, and therefore, the results do not support the acceptance of *H1, H2, H3,* and *H4*.

Testing the KM Interrelationships Hypotheses

- **The relationship of KA to KD (*H6*):** In Table 8, the PLS results show a positive but insignificant beta coefficient of .364 (T = 1.3063) between KA and KD. Although it has no significant influence on KD, KA explains 13.2% of the variance in KD, as the endogenous variable. Therefore, the results fail to support the acceptance of *H6*.
- **The relationship of KA and KD to KT (*H7*):** The PLS results in Table 9 confirm that KA and KD explain 16.1% of the variance in KT, as the endogenous variable. In addition, there is a positive but insignificant beta coefficient of .350 (T = 1.3929) between KA and KT, and a positive but insignificant beta coefficient of .106 (T = .3969) between KD and KT. KA and KD appear to have no influence on KT, and therefore, these results fail to support the acceptance of *H7*.
- **The relationship of KA, KD, and KT to KC (*H8*):** The PLS results of Table 10 indicate that KA, KD, and KT explain 59.3% of the variance in KC, as the endogenous variable. This model has a relatively high explanatory power. As to the effect of the individual processes on KC, there is a positive and significant beta coefficient of .520 (T = 3.6433) between KA and KC, a positive and significant beta coefficient of .295 (T =

Table 8. The relationship of KA to KD

The Dependent Variable	Independent Variables	R^2	Hypothesis Number
KD	KA	.132	*H6*

Dependent Variable	Independent Variables	Hypothesis Number	Predicted Path Coefficient	Standardized Path Coefficient
KD	KA	H6	+	.364 T = 1.3063

Table 9. The relationship of KA and KD to KT

Dependent Variable	Independent Variables	R^2	Hypothesis Number
KT	KA and KD	.161	*H7*

The Dependent Variable	Independent Variables	Hypothesis Number	Predicted Path Coefficient	Standardized Path Coefficient
KT	KA	*H7*	+	.350 T = 1.3929
	KD	*H7*	+	.106 T = .3969

Table 10. The relationship of KA, KD, and KT to KC

Dependent Variable	Independent Variables	R^2	Hypothesis Number
KC	KA, KD, and KT	.593	*H8*

Dependent Variable	Independent Variables	Hypothesis Number	Predicted Path Coefficient	Standardized Path Coefficient
KC	KA	*H8*	+	.520 T = 3.6433
	KD	*H8*	+	.169 T = 1.1403
	KT	*H8*	+	.295 T = 2.0413

Table 11. The Relationship of KA, KD, KT, and KC to KAP

Dependent Variable	Independent Variables	R^2	Hypothesis Number
KAP	KA, KD, KT, and KC	.692	*H9*

Dependent Variable	Independent Variables	Hypothesis Number	Predicted Path Coefficient	Standardized Path Coefficient
KAP	KA	*H9*	+	.342 T = 2.1549
	KD	*H9*	+	.043 T = .2713
	KT	*H9*	+	.273 T = 1.8014
	KC	*H9*	+	.354 T = 2.4152

2.0413) between KT and KC, and a positive but insignificant beta coefficient of .169(T = 1.1403) between KD and KC. Therefore, the results demonstrate that only KA and KT have a significant positive influence on KC, and consequently, the results provide support for a partial acceptance of *H8*.

- **The relationship of KA, KD, KT, and KC to KAP (*H9*):** From Table 11, the PLS results show that KA, KD, KT, and KC explain 69.2% of the variance in KAP, as the endogenous variable in this model. Therefore, this model appears to have a relatively high explanatory power.

The results of the individual effect of each of the four KM processes on KAP show a positive and significant beta coefficient of .342 (T = 2.1549) between KA and KAP, and a positive and significant beta coefficient of .354 (T = 2.4152) between KC and KAP. However, there is a positive but insignificant beta coefficient of .043 (T = .2713) between KD and KAP, and a positive but insignificant beta coefficient of .273 (T = 1.8014) between KT and KAP. Therefore, the results reveal that only KA and KC have a significant positive influence on KC, and consequently, provide support for a partial acceptance of *H9*.

To summarize, the PLS results lend support to the acceptance of *H5*, a partial acceptance of *H8*, and a partial acceptance of *H9*. The results, however, fail to support *H1*, *H2*, *H3*, *H4*, *H6*, and *H7*. These results are further discussed in the next section.

DISCUSSIONS

This investigation examined the relationship of the KM processes to organizational performance as well as the interrelationships within the KM processes in the form of nine hypotheses. The results of the final models, depicted in Figure 2, show that: (1) the five KM processes of knowledge acquisition (KA), knowledge documentation (KD), knowledge transfer (KT), knowledge creation (KC), and knowledge application (KAP) explain approximately 20% of the variance in the organizational performance (OP) of the investigated software firms; (2) KA, KD, KT, and KC explain 69% of the variance in KAP; and (3) KA, KD, and KT explain 59% of the variance in knowledge creation.

Although relatively low (approximately 20%), the explained OP variance is significant because it is the first empirically based evidence on the contribution of KM processes to OP in the Egyptian software industry. This finding can be interpreted in light of the assumption that knowledge is not always used, and does not always result in improved performance when used. As Kalling (2003) explains, if knowledge is not utilized, it will not contribute much to performance (e.g., profit); and even if knowledge is utilized, it might not contribute to improved performance. In order for knowledge to improve performance, knowledge application decisions should be made in light of the software firms' strategic processes and products-enhancement plans.

The relatively low explained variance in OP may be also attributed to the research design employed in this investigation. First, OP has been operationalized and measured using a subjective single-item measure. Such a measure may not effectively reflect all possible contributions that KM processes may have on OP. Second, the proposed causal relationship between KAP and OP is not necessarily straightforward. The effect of KAP may occur over several layers of causal relationships that should be identified and explicitly incorporated when modeling the relationships between KM processes and OP. Third, there may be other factors—such as environment, leadership, and organizational culture—that may as well influence OP (Yeo, 2003), and are not accounted for in the research model or the data analysis.

Among the KM processes, only KAP was found to directly influence OP. This finding

confirms the theoretical argument that knowledge application is a strategic resource of the firm, and that the primary rationale for the firm is to produce and apply knowledge (Choi & Lee, 2003; Grant, 1996; Nonaka, 1994; Nonaka & Takeuchi, 1995; Spender, 1994). It also supports Alavi and Leidner's (2001) argument that knowledge creation and codification do not necessarily contribute to improved performance or created value. Performance is improved and value is created only when knowledge is applied. In addition, this finding confirms Markus' (2001) assertion that knowledge usage associates with organizational effectiveness, and Alavi and Tiwana's (2000) observation that knowledge becomes underutilized as a driver of performance when it is not put into application.

Zack (1999) points out that the ability of an organization to create knowledge and to continue to learn from it may become a competitive advantage, since the innovative knowledge developed today becomes the core knowledge of tomorrow. However, the finding of no direct influence of knowledge creation on organizational performance may be attributed to the possible "gap" between knowledge creation and knowledge application. Components of the newly created knowledge in the software firms are perhaps still in an informational form and need to be further modified in order to become applied knowledge. Sveiby (1996) adds that newly created knowledge should be explored and used in order to find more appropriate applications for it.

Although they did not directly influence OP, KA, KT, and KC were found to indirectly influence organizational performance. KA indirectly influenced OP through its direct influence on both KC and KAP, which directly influenced OP. In addition, KT indirectly influenced OP through its direct influence on KC, which, in turn, influenced KAP. Also, KC indirectly influenced OP through its direct influence on KAP.

These findings are consistent with those of Gold et al. (2001), who found that knowledge acquisition, conversion, application, and protection are positively related to performance variables. They also confirm the research results of Kremp and Mairesse (2004), who found KM to influence innovative performance and productivity. These findings are also consistent with the other empirical studies that confirmed the importance of KM processes and activities in improving organizational performance or effectiveness (e.g., Edward et al., 2005; Jennex & Olfman, 2002; Kremp & Mairesse, 2004; Lee et al., 2005; Liu et al., 2004; Salojarvi et al., 2005).

As to the interrelationships among the KM processes, KA was found to positively influence KC and KAP. KT was found to positively influence KC. And KC was found to positively influence KAP. This means knowledge that is acquired from outside sources and transferred/shared within the firm facilitates the creation of new knowledge as well as the application of such knowledge to improve the performance of the software firms.

These findings support Nonaka and Takeuchi's (1995) SECI model of knowledge creation that establishes the interrelationships among the KM processes. Knowledge transfer is perceived to be the common factor that facilitates the four processes of knowledge creation in the model (Lee & Choi, 2003; Marr et al., 2003). These findings also agree with the argument that firms normally seek additional knowledge in the same area in which they already have a knowledge base and they may seek knowledge in a complementary area (Shenkar & Li, 1999), which is a prerequisite for knowledge search and use. In addition, these findings support the argument that existing knowledge bases facilitate new knowledge creation (Shenkar & Li, 1999; Cohen & Levinthal, 1990).

KA and KT appear to be essential processes to knowledge creation in the software firms. In order to create and build new knowledge, software firms need to acquire and transfer knowledge. Sources for knowledge acquisition and creation in the investigated software firms were found to include visitation with customers, secondary

information on demand and trends of software, market research, the Internet, individual initiatives of software developers, and R&D activities. Also, the firms were found to use e-mail, voice mail, forums, traditional meetings, groupware, on and off-the-job training, and education as methods for internal knowledge transfer (Seleim, Ashour & Khalil, 2005b).

On the other hand, KD is the only KM process that was found to have no direct or indirect influence on organizational performance and the other four KM processes of KA, KT, KC and KAP. This finding is inconsistent with the knowledge reuse theory (Markus, 2001), where organizational memory plays an important role in organizational success (Stein & Zwass, 1995).

One possible reason for the insignificant impact that KD had on OP and the other KM processes in the investigated firms is the lack of quality systems for knowledge documentation within the investigated software firms. Although the CEOs of a number of these firms recognize the important role of IT-based systems in building relationships, facilitating the exchange of information, and sharing experience and knowledge, the firms are currently not fully utilizing the capabilities of such systems (Seleim et al., 2005a).

IMPLICATIONS

The findings of this investigation contribute to the growing empirical KM research. Measuring scales for KM processes were adopted, revised, validated, and used to empirically examine the interrelationships between KM processes (in the form of antecedents and consequences) and their influence on the Egyptian software firms' performance. This is a step forward towards a future adoption, revision, and validation of a comprehensive KM measuring instrument and producing a generally accepted instrument that facilitates the accumulation of an empirically-based and consistent body of knowledge on KM.

Wiig (1999, p. 6) argues the need for a new theory of the firm in order to effectively manage knowledge and link it to the firm's strategy, tactics, and daily operations. The findings of this investigation extend and substantiate the absorptive capacity theory (Cohen & Levinthal, 1990) that emphasizes the organizational processes and activities, by which organizations acquire, absorb, transfer, and exploit knowledge. In particular, this research explored a number of relationships within the KM processes. Although they provide limited support for the proposed relationships, the findings indicate the value of continued investigation and refinement of the proposed research model in order to enhance our understanding of a phenomenon that is as complex as KM.

The findings of this research further support the significance of the knowledge-based theory of the firm as a paradigm to investigate and understand organizations in the knowledge economy context. It confirms the core of the knowledge-based theory of the firm because software firms not only create knowledge but also emphasize knowledge application (Grant, 1996; Spender, 1994).

For practitioners in the Egyptian software companies, the findings of this investigation demonstrate the importance of the KM processes, especially knowledge application to organizational performance. Software firms should adopt knowledge strategies in order to create value by generating and applying knowledge. These knowledge strategies may be guided by Hamel's (1991) recommended ways for creating value through knowledge utilization and application:

1. apply new knowledge to old or existing software products and services in order to add new functionalities and characteristics that better fulfill users' needs,
2. develop new software products and services through knowledge,
3. create value by globalizing or exporting deeply embedded local knowledge such as leading business practices, software prod-

ucts, software development techniques and methods, and the like to the other Middle Eastern countries, and

4. convert knowledge into strategic knowledge to create shareholders' wealth through effective leverage of organizational knowledge.

The Egyptian software firms need to adopt KM strategies—codification, personalization, or both—in order to guide knowledge acquisition, creation, and transfer practices in the software firms. Codification strategies should emphasize the systematization and storage of knowledge and make it available for software developers and other employees in the firms (Hansen et al., 1999). The development of comprehensive and formal KM systems that capture, edit, store, update, and disseminate knowledge is essential to improve the knowledge documentation process (Seleim et al., 2005b).

Software firms should also capitalize on the Egyptian national culture and adopt personalization KM strategies in order to develop a collaborative knowledge culture that allows knowledge creation and transfer within the firms. The Egyptian culture, which is rich in its social capital and interpersonal relationships (House et al., 2002; Javidan & House, 2001; Nahapiet & Ghoshal, 1998), encourages software developers and managers to assist each other and exchange experience and knowledge. It also encourages openness, trust, and mutual respect, which are vital in enabling knowledge transfer and learning (Kautz, Thaysen & Vendelo, 2002).

Finally, since the findings of this investigation provide some evidence on the interrelatedness of the KM processes, managers in the software firms need to manage knowledge from a holistic perspective that takes into consideration that KM processes are interrelated and transcend the departmental boundaries within the firm. Formal and informal integrating mechanisms must be adopted in order to facilitate a coordinated and balanced KM processes, since the deficiency of one or more of the KM processes may negatively affect the result in less than optimum outcome of the KM efforts in the firm.

RESEARCH LIMITATIONS AND FUTURE RESEARCH

Due to the lack of well-defined and broadly accepted theories and frameworks in the KM literature to guide the formulation of hypotheses describing possible interrelationships among the KM processes, only a selected number of interrelationships were included and tested in our research model. In addition, our results failed to detect a number of the hypothesized relationships. These undetected relationships may be due to possible measurement errors, the relatively small sample size, or contextual factors that were not accounted for in the research model. Future research should use research designs that employ larger samples to further validate the KM measuring instrument used in this investigation, hypothesize, and test new sets of relationships within KM processes, and investigate the role of the intermediary effect of the organizational context (e.g., organizational culture and climate, organizational size, organizational age, and business environment)on the relationships between knowledge application and organizational performance.

The use of an exclusive indicator to measure organizational performance represents a limitation to this research (de Pablos, 2002). Using multiple variables to measure organizational performance may help to get a more accurate representation of the relationship between KM and organizational performance. Future research may measure organizational performance as a function of multiple financial variables, such as return on investment (ROI), return on equity (ROE), sales growth, and profit growth. Or, since the financial measures may be confounded by other business, economic, and environmental factors, future research may adopt less confounded nonfinancial measures,

such as the fourteen-item scale developed by Gold et al. (2001), to measure KM contributions to organizational performance.

Furthermore, a cross-sectional data was used in this research with assumed causal flows within KM processes and from KM processes to organizational performance. There is, however, a possibility that the relationships may occur in reverse order. Therefore, future longitudinal investigations are recommended in order to conclusively replicate the findings presented in this research. In addition, future research may investigate alternative paths that connect KM processes and organizational performance. For instance, the application of knowledge may result in the creation of new knowledge (Gold et al., 2001), and the transfer of more knowledge among the individuals in an organization. Organizational learning may also mediate the relationship between knowledge application and organizational performance.

CONCLUSION

This investigation examined the relationship of the KM processes of knowledge acquisition, documentation, transfer, creation, and application to organizational performance and the interrelationships among the KM processes themselves in the form of nine hypotheses. Although the findings provide limited support for the proposed research model, they indicate the value of continued investigation and refinement of the model. Only knowledge application, among the KM processes, has a significant, direct and positive influence on organizational performance. This implies that KM is most effective in the Egyptian software firms when their employees not only receive and share knowledge but also apply it, and, thus, affects organizational performance.

Given the exploratory nature of this investigation, the proposed research model is considered adequate in explaining the variance in the organizational performance of the Egyptian software firms, as a function of the KM processes. Also, two submodels of the research model exhibit high power in explaining the variance in knowledge application and knowledge creation as functions of the other KM processes.

With the exception of knowledge documentation, the KM processes have either direct or indirect influence on organizational performance. Also, knowledge acquisition and knowledge creation directly influence knowledge application, while knowledge application and knowledge transfer directly influence knowledge creation. Therefore, KM processes within the Egyptian software firms should be viewed and managed as integrated, rather than separate, processes. Follow up research is warrant in order to validate and extend our research model.

REFERENCES

Albrecht, K. (1996, April 25). *Bits and bytes from the Nile*. Middle East.

Alavi, M., & Leidner, D.E. (2001). Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly, 25*, 107-136.

Alavi, M., & Tiwana, A. (2000). Knowledge integration in virtual teams: The potential role of KMS. *Journal of the American Society for Information Science and Technology, 53*, 1029-1037.

Amabile, T.M., Nasco, C.P., Mueller, J., Wojcik, T., Odomirok, P.W., Marsh, M., et al. (2001). Academic-practitioner collaboration in management research: A case of cross- profession collaboration. *Academy of Management Journal, 44*(2), 418-431.

Antonelli, C. (1999). The Evolution of the industrial organization of the production of knowledge. *Cambridge Journal of Economics, 23*, 243-260

Argote, L., & Ingram, P. (2000). Knowledge transfer: A basis for competitive advantage in firms. *Organizational Behavior and Human Decision Processes, 82,* 15-169.

Barney, J. (1991). Firm resources and sustained competitive advantage. *Journal of Management, 17*(1), 99-120.

Bartlett, C., & Ghoshal, S. (2000). Going global lesson from late movers. *Harvard Business Review, 78,* 132-142.

Bassie, L.J. (1997). Harnessing the power of intellectual capital. *Training & Development, 51,* 25-30.

Beckman, T. (1999). The current state of knowledge management. In J. Liebowitz (Ed.), *Knowledge management handbook* (pp. 1-22). Boca Raton, FL: CRC Press.

Bennett, R., & Gabriel, H. (1999). Organizational factors and knowledge management within large marketing departments: An empirical study. *Journal of Knowledge Management, 3(*3), 212-25.

Bierly, P., & Chakrabarti, A. (1996). Generic knowledge strategies in the U.S. pharmaceutical industry. *Strategic Management Journal, 17* (Winter Special Issue), 123-135.

Bloodgood, J.M., & Salisbury, W.D. (2001). *Understanding the influence of organizational change strategies on information technology and knowledge management strategies. Decision Support Systems, 31,* 55-69.

Borys, B., & Jemison, D.B. (1989). Hybrid arrangements as strategic alliances: Theoretical issues in organizational combinations. *Academy of Management Review, 14*(2), 234-250.

Camelo-Ordaz, M.C., Fernandez-Alles, M., Martín-Alcázar, F., Romero-Fernández, P.M., & Cabrera, R. V. (2004). Internal diversification strategies and the processes of knowledge creation. *Journal of Knowledge Management, 8*(1), 77-93.

Cavaleri, S.A. (2004). Leveraging organizational learning for knowledge and performance. *The Learning Organization, 11*(2), 159-176.

Choi, B., & Lee, H. (2003). An empirical investigation of KM styles and their effect on corporate performance. *Information & Management, 40*(5), 403-417.

Cohen, W.M., & Levinthal, D.A. (1990). Absorptive capacity: A new perspective learning and innovation. *Administrative Science Quarterly, 35*(1), 128-152.

Connor, K.R., & Prahalad, C.K. (1996). A resource-based theory of the firm: Knowledge versus opportunism. *Organizational Science, 7,* 477-501.

Darr, E.D., Argote, L., & Epple, D. (1995). The acquisition, transfer, and depreciation of knowledge in service organizations: Productivity in franchises. *Management Science, 41*(11), 1750-62.

Darroch, J. (2003). Developing a measure of knowledge management behaviors and practices. *Journal of Knowledge Management, 7*(5), 41-54.

Darroch & McNaughton (2002). Developing a measure of knowledge management. In N. Bontis (Ed.), World congress of intellectual capital readings (pp. 226-242). Boston: Butterworth-Heinemann-KMCI Press.

Das, S., Sen, P.K., & Sengupta, S. (2003). Strategic alliances: A valuable way to manage intellectual capital. *Journal of intellectual capital, 4*(1), 10-19.

Davenport, T.H., & Prusak, L. (1998). *Working knowledge: How organizations manage what they know.* Boston: Harvard Business School Press.

Delaney, J.T., & Huselid, M.A. (1996). The impact of human resource management practices on perceptions of organizational performance. *Academy of Management Journal, 39,* 949-69.

Demarest, M. (1997). Understanding knowledge management. *Long Range Planning, 30*, 374-84.

Demsetz, H. (1991). The theory of the firm revisited. In O.E. Williamson & S. Winter (Eds.), *The nature of the firm* (pp. 159-78). New York: Oxford University Press.

de Pablos, P.O. (2002). Knowledge management and organizational learning: Typologies of knowledge strategies in the Spanish manufacturing industry from 1995 to 1999. *Journal of Knowledge Management, 6*(1), 52-62.

Dess, G. (1987). Consensus on strategy formulation and organizational performance: Competitors in fragmented industry. *Strategic Management Journal, 8*(3), 259-277.

Dess, G.G., & Robinson, R.B. (1984). Measuring organizational performance in the absence of objective measures. *Strategic Management Journal, 5*, 265-73.

Dollinger, M.J., & Golden, P.A. (1992). Interorganizational and collective strategies in small firms: Environmental effects and performance. *Journal of Management, 18*, 695-713.

Drucker, P.F. (1993). *Post-capitalist society.* New York: HarperCollins.

Edward, C., Feng, K., & Liou, W. (2005). Implementation of knowledge management systems and firm performance: An empirical investigation. *Journal of Computer Information Systems, 45*(2), 92-104.

Faraj, S., & Sproull, L. (2000). *Coordinating expertise in software development teams. Management Science, 46*(12), 59-85.

Filius, R., de Jong, J., & Roelofs, E. (2000). Knowledge management in the HRD office: A comparison of three cases. *Journal of Workplace Learning, 12*(7), 286-295.

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of Marketing Research, 18*, 39-50.

Fornell, C., & Cha, J. (1994). Partial least square. In R. P. Bagozzi (Ed.), *Advanced methods of marketing research*. Oxford: Basil Blackwell Ltd.

Garvin, D.A. (1994). Building a learning organization. *Business Credit, 96*(1), 19-28.

Gold, A.H., Malhotra, A., & Segars, A.H. (2001). Knowledge management: An organizational capabilities perspective. *Journal of Management Information Systems, 18*, 185-214.

Gorelick, C., & Tantawy-Monsou, B. (2005). For performance through learning, knowledge management is *the* critical practice. *The Learning Organization: International Journal, 12*(2), 125-139.

Grant, R.M. (1996). Toward a knowledge-based theory of the firm. *Strategic Management Journal, 17*(Winter Special Issue), 109-122.

Hair, J.F., Anderson, R.E., Tatham, R.L., & Black W.C. (1998). *Multivariate data analysis.* Upper Saddle River, N.J: Prentice Hall.

Hall, R., & Andriani, P. (2002). Managing knowledge for innovation. *Long Range Planning, 35*, 29-48

Hamel, G. (1991). Competition for competence and inter-partner learning within international strategic alliances. *Strategic Management Journal, 12*(Summer Special Issue), 83-103.

Hansen, M.T., Nohria, N., & Tierney, T. (1999). What's your strategy for managing knowledge? *Harvard Business Review, 77*, 106-116.

Hansen, G., & Wernerfelt, B. (1989). Determinants of firm performance: The relative importance of economic and organizational factors. *Strategic Management Journal, 10*(5), 399-411.

Hasan, H., & Al-hawari, M. (2003). Management styles and performance: A knowledge space framework. *Journal of Knowledge Management, 7*(4), 15-28.

Hedlund, G. (1994). A model of knowledge management and the n-form corporation. *Strategic Management Journal, 15,* 73-90.

Hoffman, J.J., Hoelscher, M., & Sherif, K. (2005). Knowledge management and sustainable superior performance. *Journal of Knowledge Management, 9*(3), 93-100.

Holsapple, C.W., & Joshi, K.D. (1999). Knowledge selection: Concepts, issues, and technology. In J. Liebowitz (Ed.), (pp, 7-1, 7-17). New York: CRC Press.

Holsapple, C.W., & Jones, K. (2004). Exploring primary activities of the knowledge chain. *Knowledge and Process Management, 11*(3), 155-174.

Holsapple, C.W., & Jones, K. (2005). Exploring the secondary activities of the knowledge chain. *Knowledge and Process Management, 12*(1), 3-31.

House, R., Spangler, W., & Woycke, J. (1991, September). Personality and charisma in U.S presidency: A psychological theory of leader effectiveness. *ASQ, 36*(3), 364-396.

Howeidy, A. (1999, September 1). The software challenge. *Al-Ahram Weekly,* (447), 16-22.

Huber, G.P. (1991). Organizational learning: The contributing processes and the literatures. *Organization Science, 2*(1), 88-115.

Hulland, J. (1999). *Use of partial least squares (PLS) in strategic management research: A review of four recent studies. Strategic Management Journal, 20,* 195-204.

Hussi, T. (2004). Reconfiguring knowledge management-combining intellectual capital, intangible assets and knowledge creation. *Journal of Knowledge Management, 8,* 36-52.

Ichijo, K., Krogh, G.V., & Nonaka, I. (1998). Knowledge enablers. In G.V. Krogh, J. Roos, & D. Kleine, (Eds.), *Knowing in firms: Understanding, managing and measuring knowledge* (pp. 174-203). Thousand Oaks, CA: AGE Publication INC.

Javidan, M., & House, R. (2001). Culture acumen for the GLOBE manager: Lessons from project GLOBE. *Organizational Dynamics, 29*(4), 289-305.

Jennex, M.E., & Olfman, L.(2002). Organizational memory/knowledge effects on productivity, a longitudinal study. *In Proceedings of the 35th Hawaii International conference on System Sciences, 10 pgs.*

Jennex, M.E., & Olfman, L. (2005). Assessing knowledge management success. *International Journal of Knowledge Management, 1*(2), pp. 33-49.

Johnston, R., & Blumentritt, R. (1998). *Knowledge moves to centre stage. Science Communication, 20*(1) 99-105.

Kalling, T. (2003). Knowledge management and the occasional links with performance. *Journal of Knowledge Management, 7,* 67-81. Kautz, K., Thaysen, K., & Vendelo, M.T. (2002). Knowledge creation and IT systems in a small software firm. *OR Insight, 15,* 11-17.

Kim, D., & Kogut, B. (1996). *Technological platforms and diversification. Organization Science, 7*(3), 283-301.

Kogut, B., & Zander, U. (1996). What firms do? Coordination, identity, and learning. *Organization Science, 7*(5), 502-518.

Kremp, E., & Mairesse, J. (2004, January). *Knowledge management, innovation and productivity: A firm level exploration based on French manufacturing CIS 3 data* (Working Paper No. 10237). Cambridge, MA: National Bureau of Economic Research.

Lee, H., & Choi, B. (2003). Knowledge management enablers, processes, and organizational performance: An integrative view and empirical examination. *Journal of Management Information Systems, 20*(1), 179-229

Lee, K.C., Lee, S., & Kang, I.W. (2005). KMPI: Measuring knowledge management performance. *Information & Management, 42*, 469-482.

Liebowitz, J. (2005). Linking social network analysis with the analytic hierarchy process for knowledge mapping in organizations. *Journal of Knowledge Management, 9*(1), 76-86.

Linderman, K., Schroeder, R.G., Zaheer, S., Leidtke, C., & Choo, A.S. (2004). Integrating quality management practices with knowledge creation processes. *Journal of Operations Management. 22*(6), 589-607.

Liu, P., Chen, W., & Tasi, C. (2004). An empirical study on the correlation between knowledge management capability and competitiveness in Taiwan's industries. *Technovation, 24*(12), 971-977.

Loermans, J. (2002). Synergizing the learning organization and knowledge management. *Journal of Knowledge Management, 6*, 285-294.

Lopez, S.P., Peon, J.M.M., & Ordas, C. J.V. (2004). Managing knowledge: The link between culture and organizational learning. *Journal of Knowledge Management, 8*(6), 93-104.

Lyes, M.A., & Salk, J.E. (1996). Knowledge acquisition from foreign parents in international joint ventures: An empirical examination in the Hungarian context. *Journal of International Business, 27*(5), 877-903.

Machlup, F. (1962). *The production and distribution of knowledge in the United States.* Princeton, NJ: Princeton University Press.

Machlup, F. (1983). *Knowledge, its creation, distribution and economic significance.* Princeton, NJ: Princeton University Press.

Makino, S., & Delios, A. (1996). Local knowledge transfer and performance: Implications for alliances formation in Asia. *Journal of International Business, 27*(5), 905-927.

Markus, M.L. (2001). Toward a theory of knowledge reuse, types of knowledge reuse situation and factors in reuse success. *Journal of Management Information Systems, 18*, 57-93.

Marr, B., Gupta, O., Pike, S., & Roos, G. (2003). Intellectual capital and knowledge management effectiveness. *Management Decision, 41*, 771-781.

Mason, D., & Pauleen, D.J. (2003). Perceptions of knowledge management: A qualitative analysis. *Journal of Knowledge Management, 7*(4), 38-48.

Mathiassen, L., & Pourkomeylian, P. (2003). Managing knowledge in a software organization. *Journal of Knowledge Management, 7*(2), 63-80.

Nahapiet, J., & Ghoshal, S. (1998). Social capital, intellectual capital, and the organizational advantage. Academy of Management Review, 23(2), 242-266.

Nelson, K.M., & Cooprider, J.G. (1996). The contribution of shared knowledge to IS group performance. *MIS Quarterly, 20*(4), 409-432.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science, 5*(10), 14-37.

Nonaka, I., Byosiere, P., & Konno, N. (1994). Organizational knowledge creation theory: A first comprehensive test. *International Business Review, 3*, 337-351.

Nonaka, I., & Reinmoeller, P. (2000). Dynamic business systems for knowledge creation and utilization. In C. Despres & D. Chauvel (Eds.), *Knowledge horizons: The present and promise of*

knowledge management (pp.89-112). New York: Butter-worth-Heinemann,.

Nonaka, I., & Takeuchi, H. (1995). The knowledge creating company-how Japanese companies create the dynamics of innovation. Oxford University Press.

Pablos, P.O. (2002). Evidence of intellectual capital measurement from Asia, Europe and the Middle East. *Journal of Intellectual Capital, 3*, 287-302.

Penrose, E.T. (1959). *The theory of the growth of the firm*. Oxford: Basil Blackwell.

Powell, T.C. (1992). Organizational alignment as competitive advantage. *Strategic Management Journal, 13,* 119-34.

Powell, T.C., & Dent-Micallef, A. (1997). Information technology and competitive advantage: The role of human, business, and technology resources. *Strategic Management Journal, 18*(5), 375-405.

Prahalad, C.K., & Hamel, G. (1990). The core competence of the corporation. *Harvard Business Review, 68*(3), 79-91.

Rastogi, P.N. (2002). Knowledge management and intellectual capital. *Human Systems Management, 22*, 229-240.

Reinmoreller, P., & Chong, L.C. (2002). Managing the knowledge-creating context: A strategic time approach. *Creative and Innovation Management, 11*, 165-174.

Salojarvi, S., Furu, P., & Sveiby, K. (2005). Knowledge management and growth in Finnish SMEs. *Journal of Knowledge Management, 9*(2), 103-122.

Sambamurthy, V., & Subramani, M. (2005). Special issue on information technologies and knowledge management. *MIS Quarterly, 29*(1), 1-7.

Seleim, A., Ashour, A., & Khalil, O. (2005a). Knowledge documentation and application in the Egyptian software firms. Journal *of Information & Knowledge Management, 4*, 47-59.

Seleim, A., Ashour, A., & Khalil, O. (2005b). Knowledge acquisition and transfer in Egyptian software firms. *International Journal of Knowledge Management, 1*(4), forthcoming.

Sharkie, R. (2003). Knowledge creation and its place in the development of sustainable competitive advantage. *Journal of Knowledge Management, 7*(1), 20-31.

Shenkar, O., & Li, J. (1999). Knowledge search in international cooperative ventures. *Organization Science, 10*(2), 134-143.

Spanson, Y.E., & Lioukas, S. (2001). An examination into the causal logic of rent generation: Contrast Porter's competitive strategy framework and the resources based perspective. *Strategic Management Journal, 22*(10) 907-934.

Spender, J.C. (1996). Competitive advantage from tacit knowledge? Unpacking the concept and its strategic implications. In B. Moingeon & A. Edmondson (Eds), *Organizational learning and competitive advantage* (pp. 56-73). London: Sage.

Stein, E.W., & Zwass, V. (1995). Actualizing organizational memory with information systems. *Information Systems Research, 6*(2), 86-117.

Stewart, T.A. (1997). *Intellectual capital: The new wealth of organizations.* New York: Double Day Publishing Group, Inc.

Stovel, M., & Bontis, N. (2002). Voluntary turnover: Knowledge management-friend or foe? *Journal of Intellectual Capital, 3*, 303-322.

Sveiby, K.E. (2001). *A knowledge-based theory of the firm to guide in strategy formulation. Journal of Intellectual Capital, 2*(4), 344-58.

Syed-Ikhsan, S.O., & Rowland, F. (2004). Knowledge management in a public organization: A study on the relationship between organizational elements and the performance of knowledge transfer. *Journal of Knowledge Management, 8*, 95-111.

Teece, D.J. (1998). *Capturing value from knowledge assets: The new economy, markets for know-how and intangible assets. California Management Review, 40*, 55-79.

Teece, D.J. (2000). *Strategies for managing knowledge assets: The role of firm structure and industrial context. Long Range Planning, 33*, 35-54.

Tippins, M.J., & Sohi, R.S. (2003). IT competence and firm performance: Is organizational learning a missing link. *Strategic Management Journal, 24*(8), 745-761.

Tsai, W. (2001). Knowledge transfer in intraorganizational networks: Effects of network position and absorptive capacity on business unit innovation and performance. *Academy of Management Journal, 44*(5), 996-1004.

Venkatraman, N., & Ramanujam, V. (1987). Measurement of business economic performance: An examination of method convergence. *Journal of Management, 13*, 109-23.

Wagner, B.A. (2003). Learning and knowledge transfer in partnering: An empirical case study. *Journal of Knowledge Management, 7*(2), 97-113
Walsh, J.P., & Ungson, G.R. (1991). Organizational memory. *Academy of Management Review, 16*, 57-91.

Wiig, K.M. (1997). Integrating intellectual capital and knowledge management. *Long Range Planning, 30*(3), 399-405.

Wiig, K.M. (2002). *Knowledge management in public administration. Journal of Knowledge Management, 6*, 224-39.

Yeo, R. (2003). The tangibles and intangibles of organizational performance. *Team Performance Management, 9*(7/8), 199-204.

Zack, M. (1999). Developing a knowledge strategy. *California Management Review, 41*(3), 125-46.

Zhang, L., Tian, Y., & Qi, Z. (2006). Impact of organizational memory on organizational performance: An empirical study. *The Business Review, 5*(1), 227-232.

APPENDIX A.

An English Form of the KM Practice Measuring Instrument Adopted in this Investigation*

(1 = very low practice, and 5 = very high practice)

Knowledge acquisition (KA):
1. To what extent the members in your firm actively participate in professional networks or associations.
2. To what extent your firm regularly collects information about the needs of its customers.
3. To what extent your firm regularly conducts knowledge gab analysis.
4. To what extent your firm hires consultants when important skills/information are not available in-house.
5. To what extent your firm hires new staff members who possess missing knowledge.
6. To what extent your firm conduct research (i.e., with universities and/or research centers) to explore future possibilities or to gain technical knowledge.
7. To what extent the employees in your firm regularly attend courses, seminars, or other training programs to remain informed.
8. To what extent your firm considers competitors as a source of inspiration for developing new methods and/or products.
Knowledge documentation (KD):
9. To what extent your firm uses brainstorming sessions for problem solving.
10. To what extent your firm evaluates failures and successes and "lesson learned" are set down.
11. To what extent your firm has available up-to-date handbooks, manuals, CDs, and so forth, which are frequently used.
12. To what extent your firm informs its members systematically of changes in procedures, handbook, and so forth.
13. To what extent your firm has documented the specific knowledge and skills of its individual members.
14. To what extent your firm encourages its experts to make explicit the methods they use in developing software products.
15. To what extent your firm keeps and maintains knowledge maps, knowledge networks, and data warehouses.
Knowledge transfer (KT):
16. To what extent your firm assigns mentors to the new hires to help them find their way in the organization.
17. To what extent your firm extracts the experiences of its experts and shares them with others in the organization.
18. To what extent the employees in your firm share with colleagues and others their knowledge/know how.
19. To what extent the knowledge in your firm is distributed in informal ways.
20. To what extent the knowledge in your firm is distributed in formal ways.
21. To what extent your firm holds regular business update meetings to discuss software development issues.
22. To what extent the members in your firm regularly inform each other about positive experiences and successful work methods.

23. To what extent your firm conducts intercollegual review in which members discuss their methods of working.
24. To what extent the members in your firm change jobs regularly in order to distribute their know-how.
25. To what extent your firm uses mechanisms and means for knowledge exchange across individuals, groups, and organizational levels.
Knowledge creation (KC):
26. To what extent Individual performances are assessed regularly and discussed in individual evaluative conferences.
27. To what extent problems, failures, and doubts are discussed openly in your firm.
28. To what extent that new ideas lead to re-design of work methods and processes in your firm.
29. To what extent members are assigned to new projects depending on know-how and availability.
30. To what extent your firm endeavors to find knowledge combination that contributes to its identity.
31. To what extent the members in your firm are rewarded for developing new knowledge and testing new ideas.
32. To what extent your firm promotes and stimulates a learning climate among employees.
33. To what extent your firm contributes to the development of the important ideas and knowledge in the industry.
34. To what extent the important issues in your firm are explored using scenario- or simulation techniques.
35. To what extent your firm analyzes benchmark at the industry level.
36. To what extent your firm conducts data mining to discover new knowledge and insights.
Knowledge application (KAP):
37. To what extent selling knowledge, products, or services gets explicit attention in your firm.
38. To what extent customer feedback is used to improve products/services in your firm.
39. To what extent your firm uses existing know-how in a creative manner for new applications.
40. To what extent your firm does marketing research among potential clients before developing new products or services.
41. To what extent your firm tries to conquer dysfunctional beliefs within the organization.
42. To what extent your firm utilizes multi-disciplinary teams to perform tasks and/or make decisions.
43. To what extent your firm has capabilities to integrate its knowledge across different areas.
44. To what extent your firm maximizes knowledge use through its organizational structure, management systems, and practices.
45. To what extent your firm attempts to discover the problems that cause gabs between targets and achievements.
46. To what extent your firm attempts to use its stocks of knowledge across different software projects.

** Note: The items that have been added to the original scale are written in italics.*

This work was previously published in the International Journal of Knowledge Management, edited by M. Jennex, Volume 3, Issue 4, pp. 37-66, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Section VII
Critical Issues

This section addresses conceptual and theoretical issues related to the field of software applications, which include ethics in software engineering, software piracy, and morality in free and open source software. Within these chapters, the reader is presented with analysis of the most current and relevant conceptual inquires within this growing field of study. Overall, contributions within this section ask unique, often theoretical questions related to the study of software applications and, more often than not, conclude that solutions are both numerous and contradictory.

Chapter 7.1
A Survey of Object-Oriented Design Quality Improvement

Juan José Olmedilla
Almira Lab, Spain

ABSTRACT

The use of object-oriented (OO) architecture knowledge such as patterns, heuristics, principles, refactorings and bad smells improve the quality of designs, as Garzás and Piattini (2005) state in their study; according to it, the application of those elements impact on the quality of an OO design and can serve as basis to establish some kind of software design improvement (SDI) method. But how can we measure the level of improvement? Is there a set of accepted internal attributes to measure the quality of a design? Furthermore, if such a set exists will it be possible to use a measurement model to guide the SDI in the same way software processimprovement models (Humphrey, 1989; Paulk, Curtis, Chrissis, & Weber, 1993) are guided by process metrics (Fenton & Pfleeger, 1998)? Since (Chidamber & Kemerer, 1991) several OO metrics suites have been proposed to measure OO properties, such as encapsulation, cohesion, coupling and abstraction, both in designs and in code, in this chapter we review the literature to find out to which high level quality properties are mapped and if an OO design evaluation model has been formally proposed or even is possible.

INTRODUCTION

In the last two decades there has been a growing interest and effort put after the idea of improving the quality of the software processes (Humphrey, 1989). This increasing trend had it origin in the application of statistical process control techniques (Oakland, 1990) from the manufacturing industry to our sector, thus creating a new discipline that has been called software process improvement (SPI) (Humphrey, Snyder & Willis, 1991). This discipline aids organisations to improve their software producing processes by, firstly, identifying all the broad areas of the process, their goals

and the activities and sub-activities needed to achieve them and secondly by establishing a path through which the process can be incrementally improved, this path is a set of quality levels, each of them defined by the areas and their associated goals to be accomplished. Fundamental to the SPI are the associated metrics (Fenton & Pfleeger, 1998) that are the tool by which the organisation can tell at each moment where it is in the path, each of the aforementioned goals has an associated set of metrics that help to tell if it has been achieved and to what extent. Although there are alternative SPI models and methods, like CMMI (Paulk et al., 1993) or SPICE (ISO/IEC, 1999), an organisation can always adhere to a concrete definition of process quality and a way to measure it and improve it.

However the product arena does not seem to be so established in terms of quality improvement models. A fundamental question that managers and developers often face is when it is worth to improve a software product by reengineering it or on the contrary start it all over from scratch. Fowler (2000) states that

There are times when the existing code is such a mess that although you could refactor it, it would be easier to start from the beginning ... I admit that I don't really have good guidelines for it. (p. 66)

In this case Fowler was talking about the refactoring technique but something similar can be said about other OO design knowledge elements. There is plenty of knowledge, more or less formalised, about identifying situations in which to apply an specific design improvement (Brown, Malveau, Brown, McCormick & Mowbray, 1998; Fowler, 2000; Gamma, Helm, Johnson & Vlissides, 1995; Riel, 1996), but is there a formal method to know which design transformations should be applied first or are more important? Is it possible to establish which design transformations, pattern applications, refactorings, and so forth, are more important in a certain quality level?

An organisation or a project could be interested in attaining only a moderate quality level that is acceptable for the time being and it is foreseeable that will consume only a limited amount of resources. Such an organisation could be interested in a guide that tells what quality indicators are really crucial to that quality level, to what extent, in terms of a measurable quantity, how to measure them and which design transformation affect the properties object of the measurements.

First of all it would be necessary to define what is design quality, by identifying what general properties or high level indicators comprise it; second, to organise those indicators in sets that constitute an incremental ladder of quality, so that depending on the situations, the (non-functional requirements and the resources a designer can choose the target level for his or her design; thirdly, choose the metrics that help in the assessment of the goals accomplishment; and finally, define the OO knowledge elements that apply in each case. We are talking here about something that we could call an *OO design maturity model* that would help designers to assess and improve a design before having to implement it.

Product quality has been defined in ISO 9126 (ISO/IEC, 2001) by external and internal attributes that describe the quality of the final software product and its intermediate subproducts, such as design; according to that, design quality should be measured through internal attributes that will predict, somehow, the final outcome of the external attributes. Some authors (Bansiya & Davis, 2002; Basili, Briand & Melo, 1996) have proposed numerical relations between some internal quality attributes and general OO properties, such as coupling or cohesion, for which there are already defined metrics. Other authors (Miller, Hsia & Kung) have proposed directly to measure the levels of accomplishment of certain OO knowledge elements like design principles,

using some of the existing OO metrics and map them to internal quality attributes. If the first approach was used then it would be necessary to establish how much each OO knowledge element impacts each OO property or, at least, know what design transformations are directed to which properties and re-measure after applying them and repeat the process until the level of quality is reached (see Figure 1). This chapter will review the state of the art to see whether there is an accepted set of internal quality properties as high level indicators or goals for the assessment of OO micro architecture design quality and if there are already assessment models that map metrics, OO properties and these high level indicators. A secondary objective of the review is the identification of what role the OO knowledge elements play in the assessment. There might be evaluation models based on the detection of the elements in the design so that they will not only be part of the improvement model but also of the appraisal. The sources of review will be journals, transactions, conference proceedings, and other periodicals published in the main areas of knowledge affected by this study which are object orientation design, metrics, and software maintenance.

In the next section a background on software product quality and OO metrics and OO knowledge will be presented, the method followed to perform the review will be explained one section later and the results exposed along with critical comments on the most relevant studies, and conclusions will be drawn in the closing section.

BACKGROUND

Software Product Quality

Quality can be measured at a process or product level, although there are obvious relationships among them, as Figure 2 illustrates. There are other quality aspects besides those two but are not of interest to us. ISO/IEC has issued two standards that refer to software product and process quality respectively, ISO 14598 (ISO/IEC, 1999) and ISO 9126. As is usual with these kind of standards the authors do not give an explicit assessment methodology nor do they give guidelines to achieve quality through specific software processes or development methodologies. Rather they simply put in places what is understood by quality, in terms of attributes that must be measured, in each

Figure 1. Measurement and improvement model for object-oriented design

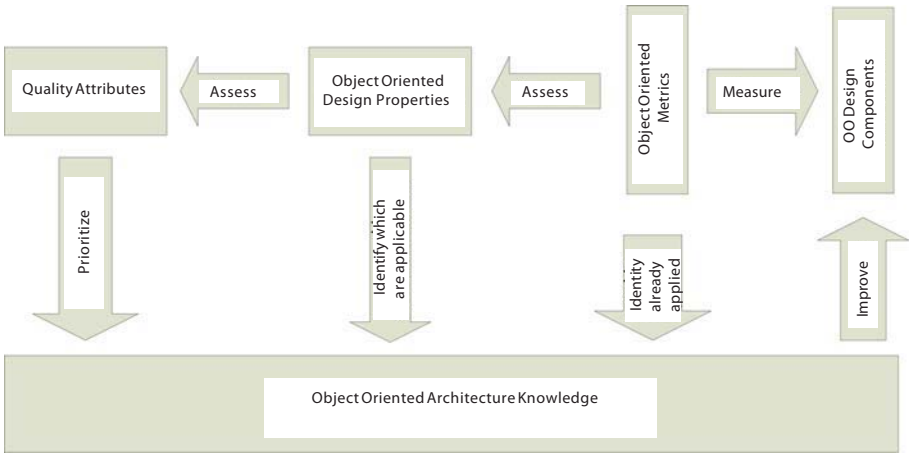
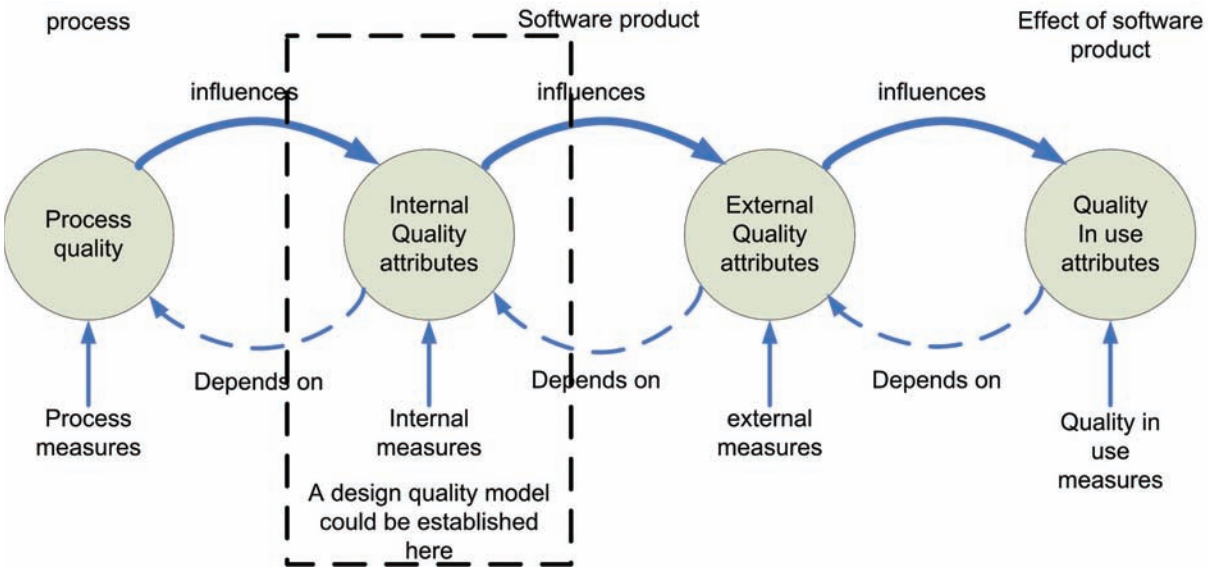


Figure 2. Quality in the software lifecycle



of the cases. The CMMI model connects with ISO 14598 in being a classification of maturity levels in the software process and giving a set of specific guidelines to assess and evaluate the quality of a software process.

ISO 9126 states that software product quality can be evaluated by measuring internal attributes, or by measuring external attributes; the former are obtained through metrics defined on intermediate products, such as design, and the latter are based on the behavior of the final executing code. It also takes in account "quality in use" which deals with the perspective of behavioral quality of the finished product in a speciûc environment under a user's perspective. In both cases, internal and external, quality is defined as a set of six characteristics:

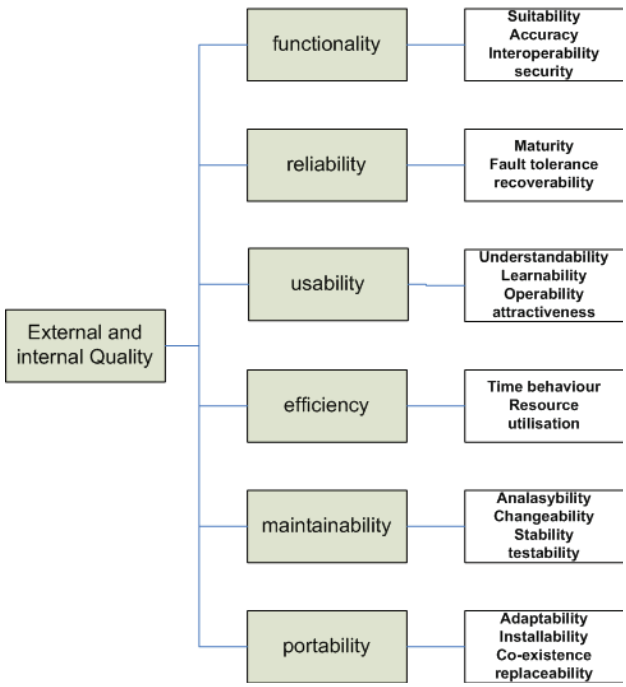
- **Functionality:** The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
- **Reliability:** The capability of the software product to maintain a specified level of performance when used under specified conditions.
- **Usability:** The capability of the software product to be understood, learned, used, and attractive to the user, when used under specified condition.
- **Efficiency:** The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- **Maintainability:** The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- **Portability:** The capability of the software product to be transferred from one environment to another.

These characteristics are general for any kind of software product and being object-oriented or

structured does not affect the choice of characteristics, although it will affect the way to measure them. These characteristics are further divided in subcharacteristics as shown in Figure 3. But we are centered in the design quality in early stages of the development cycle and certain attributes in that list, apparently, should not be addressed yet. Let us see which of them are addressable at the design stage and which are not:

- The correct coverage of all user requirements should be addressed in previous phases of the life cycle, such as analysis, and verified during testing; therefore, it is logical to suppose that the quality of a design should not be measured by the functionality attribute, as defined in ISO 9126. In fact the OO design improvement techniques always state that to apply them, first the current design must behave "mostly" correctly (Fowler, 2000) (in terms of functionality and in terms of reliability). But how can we decide if a design is better than another, for the same requirements? If one of them does not implement the full set of requirements as described in the corresponding specifications then the design is not "worse" than the other, it is simply not correct or incomplete.
- Again we tend to suppose that reliability should not be one of the internal attributes that define design quality, given that reliability has been traditionally measured during testing; however, McCabe (1976) introduced the cyclomatic complexity (CC) metric which has been used to calculate, for a given, software program, the minimum set of tests that are necessary to ensure a certain level of test coverage and therefore a prediction of the ratio of defects yet uncovered. Later works (Chidamber & Kemerer, 1991) created equivalent metrics for OO software and there are certain studies that try to predict

Figure 3. ISO 9126 Quality characteristics and sub-characteristics



reliability (Basili et al., 1996; Briand, Wust, Daly, & Victor Porter, 2000; Brito e Abreu & Melo, 1996) from design. These and other works[1] establish a relationship between the complexity, as an OO property, and the defect density or fault prone-ness, which we interpret as synonyms for reliability.

- Usability deals with the way the final user feels about the finished product. No evidence was found in the sense of establishing usability as an internal attribute for design quality. Understandability is mentioned in the literature (Bansiya & Davis, 2002; Deligiannis, Shepperd, Roumeliotis, & Stamelos, 2003; Dumke & Kuhrau, 1994) as a desired property to have in a design. However, it is more than arguable that it refers to the ISO 9126 sub-characteristic of usability, and it must be understood as "analysability." However, other voices (Fowler, 2000) claim that the "user" of a design is not the same as the user of the ûnal product. He or she could be the developer that has to implement the specified design, or that same designer (or other) in the future when a new feature has to be introduced in the system or the design must be modified for whatever reason, the latter case is already contained under "maintainability," but the former is not so clear.
- Efficiency is divided into "time behavior" and "resource utilisation" which has made this attribute a clear measuring target in the testing phases; however, there are proposals, mostly in the real-time systems area, to measure efficiency, usually addressed as "performance," early in the design stage. This could be a good candidate quality attribute for a higher level of design quality and not for a basic one. Making an OO design more understandable by, for instance, introducing patterns, introduces indirections which in turn penalises performance, so it looks like they are opposite. However if a design is more understandable thanks to being decomposed in more entities, that allows a better isolation and identification of those spots where most of the performance issues tend to be (Fowler, 2000).
- Maintainability is a clear focus of most of the OO knowledge dedicated to improve designs as we will see, this is intuitive since most design efforts in OO paradigm are centred around ideas such as data hiding, encapsulation and abstraction which enhance a better understand ability (analysability) through domain concept representation, separation of components in testable units, and so forth.
- Portability looks completely out of the scope of the design stage since the design must remain conceptually separated from the actual implementation environment. In any case, we have not found sufficient evidence to support that this attribute is important, quality wise, during early design phases.

In Bansiya & Davis (2002), a set of six design quality attributes are derived from ISO 9126 quality characteristics, although they are not taken exactly but rather adapted to the particularity of design. Two are discarded as not measurable in design, two are changed for equivalent ones, and two are added from general concepts present in software design literature.

These quality attributes are abstract concepts and, therefore, not directly observable, so we need some properties that can be observed and quantified and that are particular of object-oriented design. In many of the OO metrics suites the speciûc metrics are implicitly mapped to general design properties as cohesion, coupling, encapsulation, complexity, and inheritance, although not all of them are specific to OO design and could be applied to modular design as well. Measurements are proposed in different works for those properties and in some cases there is an explicit mapping from the former to the latter, as in Bansiya & Davis (2002) where each design

property is measured by a single metric. In Miller et al. (1999), 11 properties were chosen. In fact, object-oriented design principles, which according to Garzás and Piattini (2005) are part of the OO architecture knowledge, as we previously said, and five measurements, all of them at class level, are used to assess their degree of fulfillment. On the contrary, Bansiya and Davis (2002) choose not only classes but also class attributes, methods, and packages.

A very sound set of object-oriented design properties could be:

- Design size
- Hierarchies
- Abstraction
- Encapsulation
- Coupling
- Cohesion
- Composition
- Inheritance
- Polymorphism
- Messaging
- Complexity

But these quality attributes and OO design properties must be measured on specific components or entities of the design (see Figure1), thus it is important to define what a design is or what components are in a design susceptible to be measured. Purao and Vaishnavi (2003) survey product metrics for OO systems and propose a framework and a formalism, according to which, the product goes through different "states" during the development process and in each of them different components that he calls "entities," are produced or modified. In his work, Purao, reviewed all the different metrics suites to see what entities where measured in each state and gathered an extensive set of which we only recall here those in the design state, along with the their attributes (see Table 1).

Object-Oriented Metrics

We are going to present a summary of the most important object-oriented metrics and identify which OO properties they can measure.

Chidamber and Kemerer's Metrics Suite

Chidamber and Kemerer (C&K henceforth) first proposed in 1991a suite of six metrics. All except one were applied to the class entity and measured complexity, coupling, cohesion, inheritance, and messaging (see Table 2).

Henderson-Sellers Metrics

Another important suite was given by Henderson-Sellers, Constantine, and Graham (1996) but it was related to coupling and cohesion. Only AID (average inheritance depth of a class) was an Inheritance measure. AID was defined as zero, for a class without ancestors and the average AID of its parent classes increased by one.

Conclusion on Metrics

In Purao and Vaishnavi (2003) and Briand et al. (2000), a detailed listing of metrics is presented. An important conclusion drawn after reviewing these metrics is that, although they claim, in many cases, to be OO design metrics they are not since they need source code to be analysed or measure code size. Another important conclusion is that most metrics suites focus just in a very constrained set of properties, namely, coupling, cohesion and inheritance. There are some exceptions like Briand et al. (2000), which gives a set of metrics taken partly from previous suites that cover all properties considered in their assessment model.

Table 1. Measurable entities in design and their attributes

Entity	Attribute
Association	Size
Attribute	Position
Class	Abstractness
	Behavior
	Comments
	Effort
	Interaction
	Interface
	Performance
	Position
	Reuse
	Size
	Structure
Hierarchy	Structure
Link	Arity
Method	Abstractness
	Effort
	Interaction
	Interface
	Performance
	Position
	Reuse
	Size
	Structure
Package	Abstractness
	Interaction
	Size
	Structure
Parameter	Size
Scenario	Size
System	Behavior
	Change
	Comments
	Dynamics
	Effort
	Interface
	Performance
	Requirements
	Reuse
	Size
	Structure
Use case	Interface
	Size
	structure

Table 2. Metrics of the C&K suite

Metric	Definition	Properties
Weighted methods per class (WMC)	Consider a class C_1 with methods $M_1, M_2, \ldots, M_n$. Let $c_1, c_2, \ldots, c_n$ be the static complexity of the methods. Then: $WMC = \sum_{i=1}^{n} c_i$ The static complexity can be measured in many ways, one of them being CC(McCabe, 1976).	Complexity
Depth of inheritance tree (DIT)	The DIT metric of a class A is its depth in the inheritance tree. If A is involved in a multiple inheritance the maximum length to the root of the tree will be the DIT.	Inheritance
Number of children (NOC)	NOC of a class is the number of immediate subclasses subordinated to a class in the class hierarchy.	Inheritance
Coupling between object classes (CBO)	CBO for a class is a count of the number of other classes to which is coupled. One class is coupled to another if it uses its methods or instance variables, excluding inheritance related couples.	Coupling
Response for a class (RFC)	$RFC = \lvert RS \rvert$ where RS is the response set for the class, given by $RS = \{M\} \bigcup_{\text{all } i} \{R_i\}$ where $\{R_i\}$ = set of methods called by method i and $\{M\}$ is the set of all methods in the class. The response set of a class is the set of all methods that can potentially be executed in response to a message received by an object of that class.	Messaging
Lack of cohesion in methods (LCOM)	Consider a Class C_i with n methods $M_1, M_2, \ldots, M_n$. Let $\{I_j\}$ be the set of instance variables used by method M_i. There are n such sets $\{I_1\}, \ldots \{I_n\}$. Let $P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}$ and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$. If all n sets $\{I_1\}, \ldots \{I_n\}$ are $\emptyset$ then let $P = \emptyset$. $LCOM = \lvert P \rvert - \lvert Q \rvert$, if $\lvert P \rvert > \lvert Q \rvert$ or 0 otherwise.	Cohesion

Table 3. Li and Henry's metrics

Metric	Definition	Properties
Message passing coupling (MPC)	MPC= the number of method invocations in a clas.s	Coupling/ Messaging
Data abstraction coupling (DAC)	The number of attributes in a class that have as their type another class.	Coupling/ Abstraction
SIZE1	It is a variation of traditional LOC (Lines of Code) defined specifically for the Ada language. We obviate its definition.	Design size
SIZE2	SIZE2 = number of attributes + number of local methods.	Design size

Table 4. Bansiya and Davis metric model

Metric	Definition	Properties
Design size of classes (DSC)	Total number of classes in the design	Design size
Number of hierarchies (NOH)	Number of class	Hierarchies
Average number of ancestors (ANA)	Average number of ancestors	Abstraction
Data access metric (DAM)	Ratio of the number of private (protected) attributes to the total number of attributes declared in the class	Encapsulation
Direct class coupling (DCC)	Count of different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods	Coupling
Cohesion among methods of class (CAM)	This metric computes the relatedness methods of a class based upon the parameter list of the methods. The metric is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.	Cohesion
Measure of aggregation (MOA)	This metric measures the extent of the part-whole relationship, realised by using attributes. The metric is a count of the number of data declarations whose types are user de?ned classes.	Composition
Measure of functional abstraction (MFA)	Ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class	Inheritance
Number of polymorphic methods (NPM)	Number of methods that can exhibit polymorphic behaviour (virtual in C++ and non ?nal in Java)	Polymorphism
Class interface size (CIS)	Count of the number of public methods in a class	Messaging
Number of methods (NOM)	Count of methods de?ned in a class.	Complexity

CURRENT STATE OF THE ART IN DESIGN QUALITY ASSESSMENT MODELS

Review Questions

The objective behind this review is to find out if there is any solid research to relate software design high level indicators with OO design knowledge (patterns, heuristics, bad smells, best practices, rules, and principles). The future trend of research, on the one hand, is to relate current (or new) OO metrics with ISO 9126 internal characteristics, or establish a new set if necessary, and see if they can be measured through them. On the other hand, we are after the relation between the application of OO knowledge and the impact of those metrics, and therefore, the quality characteristics.

The review tries to determine if this gap exists or not in the OO area of knowledge. We are constrained exclusively to the design phase and more specifically to OO micro architecture.2 We are not interested in metrics or models for process quality, effort, estimation, or project tracking metrics. The area of research is always focused on metrics intended for design improvement.

The primary question of research is:

Research Question 1: Are there object-oriented design quality assessment models that use a set of metrics, based only in design entities,3 to measure levels of accomplishment in internal product quality attributes (as those in ISO 9126)? And what are those attributes or high level indicators?

Needless to say, we are interested in quantified models, so that those metrics are numerically related to the characteristics, directly or through other numerical relations with intermediate elements, such as OO properties, which, on the other hand, is what is found in all cases as we have advanced in the Background section.

The secondary question of research is:

Research Question 2: Do those models use any of the OO knowledge elements in Garzás & Piattini (2005) as part of that assessment model and how?

This secondary question leads necessarily to models where there is a detection of those elements in the design by using speciûc metrics. Since the future work will establish a way to improve the design through the application of design transformations based on those knowledge elements, it is very useful to know in advance how much that will impact the desired quality attributes and if those transformations are already present in the design.

Review Methods

A systematic review protocol following Kitchenham (2004) was used. The sources consulted were all digital, and the intention was to cover as many periodicals and conference proceedings related with metrics, software quality, OO knowledge, and software maintenance as possible. The sources chosen were IEEE Digital Library, ACM Digital Library, Journal of Systems and Software (Elsevier), Journal of Software Maintenance and Evolution (Wiley) and Software Practice and experience (Wiley).

Our search strategy was to compose queries that included in different Boolean expressions the following terms:

- Object-oriented design
- Metrics
- Quality
- High level indicators
- Assessment
- Method
- Patterns
- Heuristics
- Bad smells
- Principles
- Rules
- Refactoring
- Lessons learned
- Best practice

Different synonyms were chosen for some of the above terms like assessment for which we chose "assess," "assessing," "evaluation," "evaluate," and "evaluating"; also, "method" had different synonyms. The singular was chosen as in "pattern" instead of "patterns" in order to obtain expressions that included both variations, since the first is a substring of the second. Different queries were created with these terms and executed in the search engines. However, some of them had to be expanded, like in the case of the Journal of Systems and Software given to obtain a decent set of results and afterwards the selection had to be manually reviewed to exclude publications that were completely out of the scope. Several different queries were tried in both engines and the search was broaden more than initially thought given the initial low number of results, for instance in IEEE Digital Library the following query was executed:

"(metrics<in>metadata)<and>(object-oriented design<in>metadata)." Surprisingly enough, this query threw only 62 results (in IEEE Digital Library) which was less than other more restrictive ones. One first conclusion of the first round of searches was that, although there are many metrics suites for OO systems there are very few centered exclusively in the design phase and using only design entities as the measurable elements.

Repetitions were taken out since there were references to the same study from different sources (i.e., from IEEE and ACM Digital Libraries), the same can be said about those studies that were different publications of the same work, where we always took the most recent one, as Kitchenham (2004) advises. Once repetitions were taken out 481 studies remained. After obtaining this initial list of results we did a quick review of abstracts when available or the introduction of the paper and directly discarded those elements not having to do with object-oriented metrics. After that initial review we did a more thorough review by reading one by one the publications and using the exclu-

sion criteria explained in section "Included and excluded studies" to discard those not interesting to us. We recorded the reason for each discard whether it was in the first quick review or in the thorough one.

For the ones not discarded, that is, the primary sources object of our review, we recorded its type according to a set quality assessment levels suggested by Kitchenham (2004) according to the experimental data they included. The set of quality levels is given in Table 5.

After the review we could verify that all primary sources were below three (concurrent cohort). We decided not to establish a quality threshold and take into account all studies that passed the exclusion criteria regardless of their experimental (quality) level.

Included and Excluded Studies

The exclusion criteria used in the more thorough review were:

Exclusion criterion 1: *Study not focused on metrics for design improvement, like those too general and including effort and quality assurance.*

Exclusion criterion 2: *It does not propose an assessment model with quality attributes as target of the metrics*

Given the low number of studies (only 23) that passed the exclusion criteria we decided to record as well those studies that, not proposing a general assessment model, were focused on prediction of one or two quality attributes, recording those attributes as well.

Data Extraction

We were interested primarily in obtaining the high level indicators or internal quality attributes that could be utilised in a design improvement methodology, so we recorded all those indicators in the primary studies. We also recorded which of these studies, or "primary sources," were proposing an explicit model of assessment with a full mapping of metrics to intermediate properties and from there to high level indicators or attributes. Although the selected studies proposed a method for quality evaluation based on high level indicators and their associated metrics, only four of them proposed formally a complete mapping with explicit mappings or relations to the high level indicators, we called them "full models" (see Table 6). Later on we also recorded attributes for those discarded studies that were focused only on the prediction of one or two attributes, that we have indicated in Table 7 as "secondary sources"; in that table we can see all the indicators or attributes collected from both sources.

Table 5. Quality levels for primary sources, as in Kitchenham (2004)

Level	Name	Description
1	Randomised trial	Evidence obtained from at least one properly-designed randomised controlled trial
2	Pseudo-randomised trial	Evidence obtained from well-designed pseudo-randomised controlled trials
3	Concurrent cohort	Evidence obtained from comparative studies with concurrent controls and allocation not randomised, cohort studies, case-control studies or interrupted time series with a control group.
4	Historical control	Evidence obtained from comparative studies with historical control, two or more single arm studies, or interrupted time series without a parallel control group
5	Randomised experiment	Evidence obtained from a randomised experiment performed in an artificial setting
6	Case series	Evidence obtained from case series, either post-test or pre-test/post-test
7	Pseudo-randomised experiment	Evidence obtained from a quasi-random experiment performed in an artificial setting
8	Expert opinion	Evidence obtained from expert opinion based on theory or consensus

Table 6. Summary of data collected

Number of Studies			Accepted Studies		Primary Sources	
Total	Accepted	Discarded	Primary Sources	Secondary Sources	Full Model	Basic
481	584	232	3	354	1	9

Table 7. Collected high level indicators

Property Name	Full Model	Primary Sources	Secondary Sources
Adaptability	0	1	0
Analysability	1	1	0
Change proneness	0	0	1
Changeability	1	2	3
Completeness	0	2	0
Complexity	0	3	0
Comprehensibility	1	1	0
Consistency	0	2	0
Correctness	0	2	0
Effectiveness	1	1	0
Efficiency	0	0	1
Extensibility	1	4	3
Flexibility	1	1	2
Functionality	1	1	0
Maintainability	1	10	10
Performance	1	2	0
Realisability	0	1	0
Reliability	0	4	9
Reusability	2	5	5
Security	0	1	0
Stability	1	1	2
Testability	1	5	2
Traceability	0	1	0
Unambiguity	0	1	0
Understandability	1	4	3
Usability	0	1	0
Verifiability	0	0	1

It can be observed that there are many attributes that are really synonyms, as for example analysability, comprehensibility, and understand ability. As we expected, maintainability is the most referred in both kinds of studies by itself or adding their subcharacteristics, analysability, comprehensibility, changeability, stability, and testability. On the other hand, it must be taken into account that all these attributes refer to design and that, for instance, "understandability" means in this context that the design is easily understandable by a software developer other than its author; therefore it must not be taken as the sub-characteristic beneath "usability" in ISO 9126.

Results: Description of Primary Studies and Findings

From the data obtained in Table 7 we can conclude that, for the so called *full models*, maintainability, counting its subcharacteristics, was the highest

scoring quality attribute with a total of eight appearances. Maintainability is comprised of analysability, changeability, stability, and testability, and, in this context, an analysability, comprehensibility, and understandability are synonyms, or at least that was the semantic behind the word in the selected studies, and the same can be said about changeability, flexibility, and extensibility. Another important attribute or high level indicator is reusability.

In primary studies, those not stating explicit relation between OO properties and quality attributes, maintainability and reusability were the two most important indicators and reliability appears as the next one. In the secondary studies, reliability outperforms reusability. Apparently reliability and, in decreasing importance, performance and efficiency (and effectiveness as its synonym) take more importance as the studies try to predict a specific quality attribute instead of evaluating the overall quality.

Apparently, when trying to establish a quality evaluation method, reliability, defect proneness, and functional correctness and consistency are way less important than maintainability. On the other hand, there are many studies that try to predict and decrease defects early in the design phases. One possible interpretation is that quality evaluation methods, as we said before, are not seen as a replacement for software quality assurance and try to establish a way to measure and compare different designs that are semantically equivalent, that is, built for the same functional requirements, and mostly correct (defect-free).

Given the resulting figures, a statistical analysis was not considered relevant. Only four studies were considered relevant to our study and they will be summarised. In Table 8, a brief summary is shown about the use these studies make of the OO micro architecture knowledge; only those elements that have appeared in at least one of the studies is listed.

Bansiya and Davis's QMOOD

Bansiya and Davis (2002) propose a model for the assessment of high level design quality attributes in object-oriented designs called *quality model for object-oriented design* (QMOOD). It is decomposed in four levels, OO design components (level 4 or L_4), OO design metrics (level 3 or L_3), OO design properties (level 2 or L_2) and finally design quality attributes (level 1 or L_1), and links between adjacent levels, L_{34} (from L_3 to L_4), L_{23} (from L_3 to L_2) and L_{12} (from L_2 to L_1). This model is very similar to that depicted in Figure 1 and the quality attributes are taken partly from the ISO 9126 quality attributes: reusability, flexibility, understandability, functionality, extendability, and effectiveness.

In each of the links, Bansiya and Davis identify explicit relation between components of both levels, for L_{23} the mapping is exactly one to one, one metric for each of the properties (design size, hierarchies, abstraction, encapsulation, coupling, cohesion, composition, inheritance, polymorphism, messaging, and complexity), and in L_{12}

Table 8. Use of OO knowledge in the four full models

	Principles	Patterns	Heuristics	Bad smells
Bansiya & Davis	No	No	No	No
Miller, Hsia & Kung	Yes	No	No	No
Barber & Graser	No	Indirectly	Yes	No
Marinescu & Ratiu	No	Partially	No	Yes

the mapping is even more explicit because each quality attribute (from ISO 9126) is the result of the sum of each of the calculated metrics multiplied by its weight, for instance, Reusability equals 0.25*Coupling +0.25*Cohesion+ 0.5*Messaging +0.5*Design Size.

The weights can be positive or negative and were calculated somehow intuitively; in fact, the study states that the weights, as well as other mappings, can be changed to reflect the goals of the organisation.

On the other hand, relative importance of each quality attribute is not stated and is left to the designer's decision.

Finally, there is no use in the model of OO knowledge such as design patterns, principles, refactorings, or other such elements.

Miller, Hsia and Kung OO Architecture Measures

Miller, Hsia, and Kung (1999) define an OO architecture quality measurement method that fills, in a way, the gap that Bansiya and Davis had. They use quantitatively OO architecture knowledge, in the form of well known principles. Again there are defined design components, metrics (they call them measurements) and OO principles; they substitute the OO properties for these latter. Unfortunately, they stop there, not quantifying the impact of those principles in general quality attributes; in fact, they do not identify quality attributes or refer to ISO 9126, although they give high importance to the extendability, flexibility, and maintainability of the architecture.

The design components they use are hierarchies, relationship, classes, methods, and attributes (they call the operations) but in defining how to obtain the measures, they give a high importance to operations, and the impact on all of the measures. Their measurements are class abstractness, hierarchy chain brittleness, class abstraction cohesion, pure inheritance index, and relationship abstraction index. As for the principles used: open-closed, Liskov substitution, dependency inversion, interface segregation, reuse/release equivalency, common closure, common reuse, stable abstractions, least astonishment, deep abstract hierarchies, and Demeter (see Miller et al., 1999 for references).

In summary, this study is not really a quality evaluation model since it does not calculate (explicitly) internal or external software quality attributes. It does not use general OO properties either but it is interesting because it uses some OO knowledge in a quantitative way.

Barber and Graser's RARE

Barber and Graser's (2000) study is not, strictly speaking, a quality evaluation model or method but a tool for creating evaluation models by specifying which quality attributes are the target for the designer. This tool is called RARE (Reference Architecture Representation Environment). Barber and Graser state that quality attributes have an impact on each other and that not all of them can be maximised at the same time and, therefore, the designer must explicitly solve the conflicts that arise. The idea behind this study is that no single quality model can be established given that, for instance, flexibility or extensibility will negatively impact on performance and there are application domains where one or the other can be more important. As a matter of example, the study mentions reusability, extensibility, comprehensibility, and performance as the main quality attributes to work with, although it does not imply that others cannot be added.

As in the other models there are mappings that drive calculations from OO design metrics to quality attributes, but in this case the intermediate elements are OO knowledge elements: *heuristics* and *strategies* (as refactorings and design transformations that guide the application of heuristics). Thus, Barber and Graser incorporate OO knowledge not as a goal, as in Miller et al. (1999), but rather as a tool to achieve quality attribute

enhancement. The quality attributes chosen by the designer and their associated importance weights are quality goals; the metrics calculate the degree of achievement of the heuristics, which are used to calculate the level of achievement of the goals. This tool uses strategies to help the designer to change design in order to increase certain heuristics, and the order in which they are suggested to the designer is driven by the goals.

Unfortunately the study talks about a tool still under construction and, in our searches, we have not seen further notice of it. No quantitative measures are given in the paper about the calculation of each heuristic, nor a list of heuristics and strategies is given for it; although promising, we must discard this study as incomplete.

Marinescu and Ratiu's Factor Strategy Model

In Marinescu and Ratiu (2004),yet another perspective is given, this time an indirect measurement of the quality is proposed, instead of measuring the quality of the design, Marinescu and Ratiu measure the lack of quality by detecting common design flaws. The model is comprised of OO metrics, design flaws and, finally, quality factors and goals. The first two set of elements are tied together through principles, rules, and heuristics and with them design flaws are quantified through *detection strategies.* Each quality factor has an associated formula for calculating its level from the set of design flaws (quantified from metrics through detection strategies) and the total quality is given by the quality goals chosen (ISO 9126 quality attributes such as maintainability or reliability) and the weight given to each one. Quality goals are divided into factors exactly as in ISO 9126, and the study refers to it and gives two example formulae for maintainability from its sub-factors (changeability, testability, analysability, and stability) exemplifying that different weights could be used for each subfactor according to experience. There are similar formulae to associate factors and detection strategies and the relative weights must be provided by the designers.

This is probably the most promising of all the selected studies since it takes into account OO knowledge as a tool for improvement and measurement of the design quality and establishes that quality is decomposed in general software quality attributes that can be derived from those knowledge elements. OO properties are not quantitatively present in the model, although there is a table identifying which design flaws impact on what properties (only coupling, cohesion, complexity, and encapsulation are listed).

We see that, on the one hand, quantification of the different mappings must be provided by the developer and, on the other hand, the design flaws (and detection strategies) are categorised according to the design component they are tied to; however, we see that OO knowledge is not properly classified and there are missing elements (only "bad smells" and some "patterns" are used). Probably a better ontology could be used as rules used instead of detection strategies, for that Garzás and Piattini (2005) could be used.

CONCLUSION AND FUTURE WORK

With all the sources, primary, secondary, and discarded ones, we can conclude that studies dealing with OO quality evaluation and metrics can be classified as follows:

- True quality evaluation models based on quality attributes quantification
- Prediction models for a single or few OO properties or quality factors
- Design flaw detection methods and their associated refactorings or design transformations
- OO metrics suites

The first set is the one that interests us and can be further subdivided in models that favor

specific quality indicators or those that propose a flexible model in which the user (i.e., designer or other stakeholder) must set the quality factors and their weights.

One important conclusion of our review is that maintainability is the most used high-level quality indicator, which is logical given that Object Orientation has been seen as a paradigm that favors flexibility and reuse. Other quality indicators or attributes, such as efficiency or portability are missing in the full models, which can be due to the fact that all the studies are biased by the fact that different application domains are not considered.

Surprisingly reliability is also very important in those studies that are just prediction models.

Only four studies establish a full quality evaluation model and, of them, no one establishes an explicit hierarchy between the high-level indicators. Two of the four studies are flexible models and three of the four use, partially, OO micro architecture knowledge.

In future works two main objective can be targeted, on the one hand, by trying to establish hierarchies between quality attributes, which may be according to different hierarchy sets corresponding to application domains, and on the other hand a better application of OO micro architecture knowledge in the construction of the evaluation methods through the use of ontologies.

REFERENCES

Abran, A., James, W. M., Bourque, P., & Dupuis, R. (2004). *Guide to the software engineering body of knowledge. 2004 version*. SWEBOK: IEEE Press.

ACM Digital Library. (2005). Retrieved from http://portal.acm.org

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transaction on Software Engineering, 28*(1), 4.

Barber, K. S., & Graser, T. J. (2000). *Tool support for systematic class identification in object-oriented software architectures.* Paper presented at the 37th International Conference on Technology of Object-Oriented Languages and Systems, Sydney, NSW, Australia.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering, 22*(10), 751.

Briand, L. C., Wust, J., Daly, J. W., & Porter, D. (2000). Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software, 51*(3), 245.

Brito e Abreu, F., & Melo, W. (1996). *Evaluating the impact of object-oriented design on software quality.* Paper presented at the Software Metrics Symposium, Berlin, Germany.

Brown, W. J., Malveau, R. C., Brown, W. H., McCormick, H. W., & Mowbray, T. J. (1998). *Antipatterns: Refactoring software, architectures and projects in crisis.* NY: John Wiley & Sons.

Chidamber, S. R., & Kemerer, C. F. (1991). *Towards a metrics suite for object-oriented design.* Paper presented at the OOPSLA '91, Conference Proceedings on Object-orientedProgramming Systems, Languages, and Applications, New York.

Deligiannis, I., Shepperd, M., Roumeliotis, M., & Stamelos, I. (2003). An empirical investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software, 65*(2), 127.

Dumke, R. R., & Kuhrau, I. (1994). *Tool-based quality management in object-oriented software development.* Paper presented at the Symposium Assessment of Quality Software Development Tools, Washington, DC.

Fenton, N. E., & Pfleeger, S. L. (1998). *Software metrics: A rigorous and practical approach*. Boston: PWS Publishing Co.

Fowler, M. (2000). *Refactoring: Improving the design of existing code*: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston: Addison-Wesley Longman Publishing.

Garzás, J., & Piattini, M. (2005). An ontology for microarchitectural design knowledge. *IEEE Software, 22*(2), 28.

Henderson-Sellers, B., Constantine, L. L., & Graham, I. M. (1996). Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). *Object-oriented Systems, 3*, 142-158.

Humphrey, W. S. (1989). *Managing the software process*. Boston: Addison-Wesley Longman Publishing.

Humphrey, W. S., Snyder, T. R., & Willis, R. R. (1991). Software process improvement at hughes aircraft. *IEEE Software, 8*(4), 11-23.

IEEE Digital Library. (2005). Retrieved from http://ieeexplore.ieee.org

ISO/IEC. (1999). *Information technology — software product evaluatiom — part 1: General overview*. Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2001). *Software engineering — product quality — part 1: Quality model*. Geneva, Switzerland: ISO/IEC.

Journal of Systems and Software. (2005). Retrieved from http://ees.elsevier.com/jss/

Journal of Software Maintenance and Evolution. (2005). Retrieved from http://www3.interscience.wiley.com/cgi-bin/jhome/77004487

Kitchenham, B. (2004). *Procedures for performing systematic reviews* (Joint Tech. Rep. No. 0400011T.1): Keele University and Empirical Software Engineering National ICT Australia, Software Engineering Group, Department of Computer Science.

Li, W., & Henry, S. (1993). *Maintenance metrics for the object-oriented paradigm*. Paper presented at the Software Metrics Symposium, Baltimore, MD.

Marinescu, R., & Ratiu, D. (2004). *Quantifying the quality of object-oriented design: The factor-strategy model*. Paper presented at the 11th Working Conference on Reverse Engineering.

McCabe, T. J. (1976). A software complexity measure. *IEEE Transactions on Software Engineering, 2*, 308-320.

Miller, B. K., Hsia, P., & Kung, C. (1999). *Object-oriented architecture measures*. Paper presented at the Hawaii International Conference on System Sciences.

Oakland, J. S. (1990). *Statistical process control: A practical guide*. Oxford: Butterworth-Heineman.

Paulk, M., Curtis, B., Chrissis, M., & Weber, C. (1993). *Capability maturity model for software (version 1.1)* (Tech. Rep. No. CMU/SEI-93-TR-024). Carnegie Mellon University, Software Engineering Institute.

Purao, S., & Vaishnavi, V. (2003). Product metrics for object-oriented systems. *ACM Computer Survey, 35*(2), 191-221.

Riel, A. J. (1996). *Object-oriented design heuristics*. Reading, MA: Addison-Wesley.

Software design, part 2. (2004) *IEEE Software, 21*(6), c3.

Software: Practice and Experience. (2005). Retrieved from http://www3.intersci ence.wiley.com/cgi-bin/jhome/1752

Subramanyam, R., & Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering, 29*(4), 297.

This work was previously published in Object-Oriented Design Knowledge: Principles, Heuristicsand Best Practices, edited by M. Piattini, pp. 282-306, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.2
Software Quality and the Open Source Process

Sameer Verma
San Francisco State University, USA

ABSTRACT

This chapter introduces the open source software development process from a software quality perspective. It uses the attributes of software quality in a formal model and attempts to map them onto the principles of the open source process. Many stages of the open source process appear to have an ad-hoc approach. Although open source is not considered to be a formal methodology for software development, it has resulted in the development of very high quality software, both in the consumer and in the enterprise space. In this chapter, we hope to understand the open source process itself, and apply it to other methodologies in order to achieve better software quality. Additionally, this chapter will help in understanding the "Wild West" nature of open source and what it may hold for us in the future.

INTRODUCTION

The concept of quality is an abstract one. While quantity can be specified in standard terms such as a gram or a gallon, quality is usually a relative term. It is relative to the assessment of the product or process, as perceived by an individual or an organization (Barbacci et al., 1995). Quality is sometimes defined as compliance *to* a standard (Perry, 1991). It seems that the implications of quality vary from one task to another, and so does its assessment. However, generally speaking, we can agree on the direction it takes when the quality of an entity improves or degrades. In the case of software, the quality of software can be assessed by its characteristics. Several models exist that either measure software quality using a quantitative surrogate, or in terms of attribute sets. In this chapter, we use a model (Bass et al., 2000), where the quality of software is assessed by its attributes. These attributes include per-

formance, security, modifiability, reliability, and usability of a particular system. We will explore these attributes, the challenges they pose to the open source development process and how the open source community measures up to these challenges.

BACKGROUND

The importance of information systems and technology is well-established in our society, both in personal and professional space. It is difficult to imagine a society without access to information. Software forms an important cog of that system. While hardware such as desktops, laptops, and PDAs provide a platform for implementing computing power, software is the flexible component that is responsible for expression of innovation, creativity, and productivity (Lessig, 2005). Ranging from word-processing to number-crunching, the quality of software influences and impacts our lives in many different ways. It is no surprise that software quality has become the subject of many studies (Halloran & Scherlis, 2002).

Software Quality

While the quantitative side of software can be measured and optimized in terms of its capability to solve mathematical formulations and render graphics (Wong & Gokhale, 2005), it is the qualitative perception that matters to the end user and the organization (Bass et al., 2000). Quality is also harder to assess, so it is often ignored or replaced by quantitative measurements such as lines of code. Obviously, more lines of code do not always imply better quality (Samoladas et al., 2004).

In this chapter, we will use a model that uses five attributes, namely, performance, security, modifiability, reliability, and usability, to assess the quality of software. Performance involves adjusting and allocating resources to meet system timing requirements (Bass et al., 2000). Security can be described as freedom from danger, that is, safety. It may also be viewed as protection of system data against unauthorized disclosure, modification, or destruction (Barbacci et al., 1995). Modifiability is the ability of a system to be changed after it has been deployed. Requests can reflect changes in functions, platform, or operating environment (Bass et al., 2000). Reliability of a system is the measure of the system's ability to keep operating over time (Barbacci et al., 1995). Usability is a basic separation of command from data. It relies on explicit models for task, user, and system (Bass et al., 2001). These attributes put together can assist in assessing the quality of a software project. Of course, these attributes apply to all kinds of software, whether open source or proprietary.

Open Source Process

Next, let us examine the open source process. The purpose of looking at open source as a process, as opposed to a product, is that the open source ideology permeates well beyond software production (DiBona et al., 1999). The proponents of open source believe in a philosophy of open source more so than simply the software. They look upon software as an enabler of change in a society that increasingly relies on information technology for its survival and growth (Lessig, 2005). The following sections are by no means exhaustive, but should provide a substantial background to understanding the tenets of quality in the open source context.

Open source software (OSS) is largely developed by freelance programmers, who create freely distributed source code by collaborating and communicating over the Internet (Moody, 2001; Raymond, 1999; Sharma et al., 2002). A small, but significant proportion of the software is also developed in software companies such as IBM, Sun Microsystems, and Intel. Open source software is generally fashioned in what is often

classified as the *bazaar* style as opposed to the *cathedral* style, which is more commonly observed in proprietary software development. In his collection of essays, Raymond (1999) postulates that the proprietary style of software development follows the cathedral model, complete with plans, schedules, resources, and deliverables. In contrast, the bazaar model lacks an explicit blueprint. Work begins with an idea, followed by a series of short release cycles of software, which is prototypical at best. Over time this software development process gains momentum, in some cases, partly due to the availability of source code and partly due to common interest in the application of the software (Feller & Fitzgerald, 2002).

This development process is akin to shopping at a bazaar-like marketplace and purchasing ingredients as needed. Just as one would shop for olive oil, basil, pine nuts, lemons, parmesan cheese, and garlic to make pesto (Technorati, 2005), one could look for an operating system, a Web server, database server, and a programming language to build a Web application (Associates, 2005). The fact that this style of software development actually works comes as a pleasant surprise to many who are accustomed to working with traditional methods of software development[1]. A more subtle message from Raymond's essay is that while the *cathedral* approach may result in a grand and impressive application, it becomes somewhat outdated by the time it is completed (Raymond & Trader, 1999). The application may be complete, but the nature of the original problem may have changed significantly over time.

Failure to maintain a time schedule and contain the scope are often cited as reasons for the failure of information systems (Applegate et al., 2002). Open source's course-correction approach makes minor changes frequently to continually address the changing nature of the problem at hand. In some ways, the *bazaar* approach might alleviate the ills of the traditional software development process (Feller & Fitzgerald, 2000).

Proprietary vs. Open Source Software

Proprietary software, including the source code used to create it, is often protected by a patent (Perens, 2005). The patent, not to be confused with copyright, is not free of distribution restrictions, and is rarely free of cost. Proprietary software is also defined as the software whose source code is kept secret and belongs to a specific individual or a company (Barahona et al., 1999). In the case of proprietary software, the source code is not distributed. The software is only made available in the form of compiled, object files to end users (Feller & Fitzgerald, 2002). The ownership of the software belongs to a specific company, group, or individual. Software cannot be redistributed, modified, or sold without the explicit permission of the owner. This "permission" is usually specified in the end user licensing agreement, also called EULA.

Enterprises use similar licenses, except that enterprise licenses apply to all end users in an enterprise, and even if the source code is given to the client company for maintenance purposes, the client is not allowed to make it available to third parties, let alone the entire world (Goth, 2001). These descriptions are largely applicable to proprietary software, although many variations do occur and must be treated on a case-by-case basis (Rosen, 2005). The owner entity (usually a software company) decides on the release cycles of the proprietary software, which in most cases is based on market pressures and upgrade revenue (Applegate et al., 2002). Closely guarded source code becomes the basis of this revenue stream.

The development framework for open source leverages communication facilities of the Internet. Independent software developers examine existing source code and make modifications to fulfill immediate needs in their own environment. In the case of open source software, the licensing allows distribution in source code as well as object form. When a product is not distributed with its

source code, there is usually a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloadable via the Internet.

Proprietary Licensing

Companies that practice the traditional software approach typically use proprietary software licenses. The EULA[2] assures that any software given to a customer cannot be copied, redistributed, or modified without the explicit permission of the company. Since commercial software products are in the form of object and not source code, software products can be licensed per-user, per-machine, per-processor, or for an entire organization under closed-source software licensing. A closed-source license may require a royalty fee, and in some versions it has an expiration date by which the customer should purchase the software again or upgrade to a new version of the product. Proprietary licensing is designed to generate a continuous revenue stream, while protecting the source code, and thereby controlling its use in the market.

Open Source Licensing

In contrast, the nature of the open source software is that no right-to-use license fee or right-to-redistribute license fee is solicited from the users. No royalty is charged for businesses or individuals if they sell the source code of any open source software. However, open source must not be confused with public domain (Rosen, 2005). When the developer releases the source code to the public under the terms of an open source license, he or she does not relinquish the copyright. Open source licenses grant conditional rights to members of the general public willing to comply with the terms of the open source license. Developers can still sell private licenses to individuals and companies that are unwilling or unable to comply with the terms of the open source license (Rosen, 2005).

THE OPEN SOURCE MECHANISM

The open source community is a collection of individuals who may have any level of interest in open source software. These interests may lead them to play different roles. The involvement varies from being providers of ideas and code to simply users of the code (Dotzler, 2003). Open source communities pattern their operations off virtual organizations. In most cases, virtual organizations refer to new organizational forms that rely on strategic alliances and external partners to collaborate to achieve business goals or to serve customer needs (Davidow & Malone, 1992; Grenier & Metes, 1995; Lucas, 1996).

In order to achieve "spatial and temporal independence" (Robey et al., 1998), information and communication technologies tend to be heavily utilized to facilitate the coordination across time and space boundaries (Mowshowitz, 1997). Gallivan (2001, p. 281) defines virtual organizations as "goal directed and consisting of geographically distributed agents who may or may not ever meet face to face", which is very apt in the case of open source communities. Additionally, the *gift culture* within the open source community features giving away source code (Bergquist & Ljungberg, 2001), voluntarily testing and debugging the software (Dotzler, 2003), and supporting fellow users by promptly answering their posted questions through mailing lists and Internet Relay Chat sessions (Raymond & Trader, 1999).

Producer of Software

Open source software development is often initiated by individuals or small cliques of people for their own limited purpose (Mui et al., 2005). A phrase often used to describe this approach is "Every good work of software starts by scratching a developer's personal itch" (Raymond, 1999, p. 23). The *itch* metaphor is indicative of a small, yet focused need that an individual developer may have, as opposed to a grand plan for soft-

ware development. If the software that results from this process (*scratch*) proves to be useful, then others join in and contribute. The producer community is usually made up of individuals who are bound together by an open source project and little else. Often, these "producers" never meet in person. It is a modern marvel that entire pieces of software projects are negotiated, developed, managed, and disseminated completely online (Dotzler, 2003).

Based on the control structure of most open source software projects, a handful of individuals are given write-access to the software repository. These are the producers of software, also called maintainers in OSS parlance. Projects in the open source world adopt the walled server approach (Halloran & Scherlis, 2002). The server that hosts the software repository is visible to the entire world. Anybody can read from it. However, only the maintainers hold the right to *implement* recommended changes. Information flows freely from the producers, but not all feature requests or bug fixes are necessarily implemented into the project itself.

This approach seems to work well for controlling software development practices of the project. It does not limit the behavior of developers in their own environment. It does limit changes that the maintainers can make on the project itself. The walled server idea embodies critical aspects of software best practices such as process, management, design, and architecture *as defined by* the project leaders. This enables effective engineering management despite the fact that the engineers are self-selected and largely self-managed. It also allows for a more rigorous approach to quality control and assurance.

Consumer of Software

Oddly enough, the primary consumer base of open source software is the open source community itself. Thus, one way to classify open source development is to study it as an innovation (Verma & Jin, 2004). Considering the traditional methods of software development, there is plenty of evidence that the open source methodology is indeed an innovation. Taking this idea one step further, we can look at this development as a process that happens where the consumers of software are from the same group as the producers. In most production and consumption-based economies, the producer group is quite different from the consumer group.

This holds true in the world of traditional software as well. The stack of communities is vertically connected from producer to distributor to consumer. Consumers of software are rarely skilled in the art and science of software production, hence the segmented innovation approach, where the innovators are different from the end users. However, the open source community is different. A large proportion of the open source community falls in an intersection of producers and consumers of the software (Feller, 2001). This is inherent in the open source process. Users are treated as co-developers. Both parties subscribe to the same community. These shared innovation networks have a great advantage over the segmented innovation systems. Furthermore, individual users do not have to develop everything they need on their own; they can benefit from innovations developed by others and freely shared within and beyond the user network.

Peer Review Process

The software development process is community-driven, where the community size may vary from a small group of 2, to a large group of over 100 developers. An interesting departure from other traditional approaches is that OSS users are treated as co-developers or peers (Dotzler, 2003). Since the source code is freely available, a community of peers is able to partake in a peer review process somewhat similar to the academic peer review process conducted to publish research (Raymond & Trader, 1999). The role of members

of the community varies based on their skill sets, interest, and availability of resources. Different roles can be approximately categorized into four groups, including project owners/core developers, patch submitters, source code testers, and end users (Verma & Jin, 2004). Project owners/core developers are a small group of people who contribute most of the code and control the software releases.

Let us look at two examples to comprehend the proportion of members across different functional roles. The first example is the Apache project, which is an open source project for developing a Web server. It began as a series of patches or fixes to the original Web server developed at National Center for Supercomputing Applications at University of Illinois, Urbana-Champaign (Audris, 2003). Eventually, the patches became so numerous that it was suggested the server be named a "patchy" server; hence the name Apache (Dotzler, 2003). The core developers of Apache account for over 80% of the coding. Patch submitters involve a relatively wider development community who examine the source code in detail and submit bug fixes (Mockus et al., 2000).

Source code testers are comprised of an even larger group who download and compile the source code and report the bugs. Lastly, end users, who may constitute the largest group of all, are only interested in using precompiled object code. End users may also report problems, but the problems are based on precompiled software. Another example is the Mozilla project, which came out of the original Netscape Web browser code (Dotzler, 2003). Mozilla developed into a significantly different browser since its inception in 1998. Mozilla has about 25 core developers, over 400 patch submitters, over 10,000 quality assurance testing contributors, and around over 500,000 end users involved (Dotzler, 2003).

Debugging and Feedback

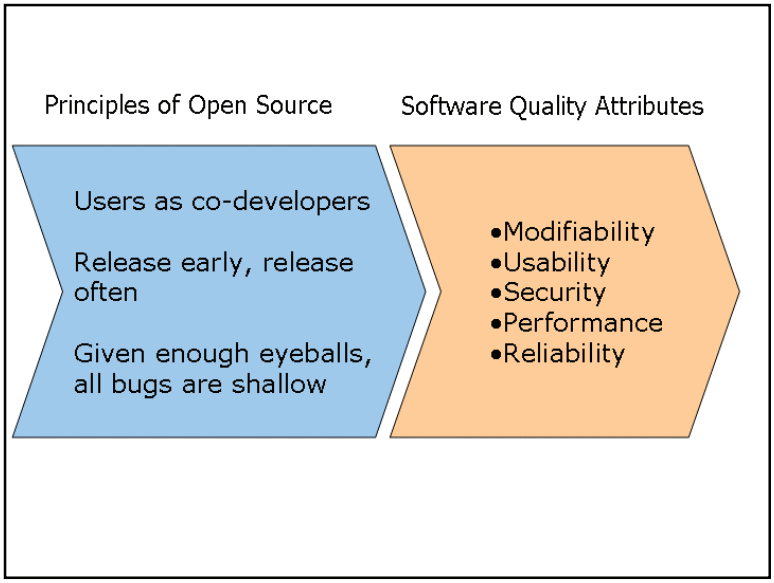
Bug reporting methods adopted by the open source community are designed for public view. All feedback is made public, thereby allowing someone on the outside to review the code and submit a change, usually called a fix or a patch. Such patches are usually maintained in parallel with existing software, so that users may pick and choose patches that are necessary for a particular scenario. The walled server approach (Halloran & Scherlis, 2002) mentioned before, provides an asymmetric channel for flow of bug-related information. The number of inputs from the community outnumbers the number of fixes implemented by the core group. Patches are usually written by one of the core members, or by patch submitters. Only core developers are allowed to make the decision to include a patch into the main software code base. Irrespective of the decision to include the patch into the main code base, the patch is always made available as an option. If a project requires some esoteric feature set provided by a patch, the developers can choose to include it at the time of compilation.

Evolutionary Survival

When the demands for features and bug fixes from the community go unanswered, the project may split or fork into a parallel development stream. Most open source licensing allows for a project fork, which is a split in the code development process into a new branch. A fork allows for independent and parallel development following two different approaches. It is theorized that the most useful branch will thrive while the less useful one will wither away (Torvalds & Diamond, 2001). This is similar to the process of genetic evolution where dominant traits get adopted and recessive ones diminish over generations.

A popular example is PHPNuke, a portal framework written in PHP, with a database back-

Figure 1. Mapping of open source principles and quality attributes



end to store content. PHPNuke started as a creation of one person to fulfill the need for a free and open source content management system. PHPNuke got popular very quickly. The application soon grew to a size that was beyond one person's ability to manage. The maintainer accepted help from other programmers but did not give them credit where it was due. He also failed to fix bugs they repeatedly pointed out, even when they submitted bug fixes. A number of these programmers became annoyed and disappointed with the general unresponsiveness of the project owner and decided to fork PHPNuke (Krubner, 2003). This version came to be known as PostNuke. With its own community, PostNuke implemented all the patches that were previously ignored. As PostNuke became more feature rich and less buggy, PHPNuke picked up momentum in order to compete with its forked creation. While forking does create a division of resources, it also creates a healthy competitive environment. Eventually, which version of the fork will dominate (i.e., used by the majority) will depend on the adaptability of the software to its environment[3].

QUALITY AND THE OPEN SOURCE PROCESS

To reiterate our approach to examining open source via the lens of quality attributes, we propose five key attributes: modifiability, usability, security, performance, and reliability. We would like to map these attributes to the primary principles of open source (Figure 1). Three most widely sited principles of OSS development include:

- Treating users as co-developers.
- Release early and release often.
- Given enough eyeballs, all bugs are shallow.

These three principles explain the essence of open source development model. Other observations are comparatively minor in their impact on the open source process as a whole.

Users As Co-Developers

The first principle of *treating users as co-developers* stems from the fact that many members of the open source community are also involved in the development process. They may be programmers or simply bug testers. Either way, the users are treated as an integral part of the development process. This approach makes it easier to communicate with the community on a deeper level of software development. For example, instead of telling users to wait for a month or two, a project member may simply ask members to download untested code from the software repository. Repeated testing and use by co-developers provides valuable feedback that plays an essential role in the quality assurance process. Often, co-developers test software on a variety of software and hardware combinations. For example, programs written for the 386 class of Intel processors running a form of Linux operating system will typically run without any problems on 486 and Pentium class processors as well. Similarly, software written for the 32-bit version of Windows (Windows 95) will typically run on Windows 98, Windows 2000, and Windows XP.

Each co-developer would often have different setup of software and hardware. The combinations of hardware and software as a testing platform make for a very rich testing process. Working with co-developers has other advantages. Every time a significant problem is solved, a patch is applied to the existing code which is re-released every night in versions dubbed as nightly releases (Dotzler, 2003). Each nightly release is based on the very latest improvements and is targeted toward the co-developer community. Each nightly build is tested and goes through a rigorous testing process conducted by the co-developers.

By treating users as co-developers, the open source community strives to create a better performing piece of software (fewer bugs) with higher levels of usability. A good example of improved performance is the Linux kernel. After its 624 iterative incarnations (Kendrick, 2005), the version 2.6.10 of Linux has come to support many form factors ranging from cell phones to mainframes (Pranevich, 2003). The performance of Linux is equally good in either of these environments. The availability of source code and feedback from co-developers has made it possible to get performance levels on par with industry giants from the proprietary world.

Similarly, the Mozilla Firefox project is an example of improved usability. The Mozilla project was started in 1998 as a spin-off of the Netscape Communicator product from Netscape Communications (Dotzler, 2003). The idea was to open source the code-base in order to foster innovation. Over the last seven years, the Mozilla project has evolved from a sluggish, bloated piece of software to a much more usable product in the stand-alone version of the browser called Firefox[4]. The usability of Firefox is considered so much better than the leading browser (Internet Explorer) from Microsoft that Firefox has been downloaded over 78 million times since its original release in 2004.

The OSS community follows an incremental model of quality and payoff. With each incremental change comes a change in quality. After all, a large proportion of the feedback comes from the user community. This high degree of involvement relies on a legal mechanism of the *right* to modify any piece of software. It is only under the auspices of an open source license, that software creators provide their source code and hence a large share of their intellectual property rights. This sharing is done for a greater good of a much better expected payoff in the long run.

Release Early, Release Often

Open source licenses are primarily designed to encourage modification and redistribution of software in source code form. The process of releasing several versions of the software in rapid cycles is aptly captured in the second principle: *release*

early and release often. For example, over its life cycle of almost 14 years, it is estimated that the Linux kernel has been released 624 times (Kendrick, 2005). Each release attempts to improve upon the previous one. In some projects, versions are released every week during rapid testing and bug fixing. The general thinking is that instead of providing the end user with a feature complete version of the software, the creators release very rudimentary versions to get the discussions going. Small changes are incorporated quickly and released often to the public.

Another such example can be seen with the NoCatAuth project (Ishii et al., 2002), a very visible project that provides open source routing for wireless networks. In its initial incarnation, the software was written in Perl. The first few versions did not even work. However, it presented the idea to a group of people who then worked on improving it. After over 25 revisions, the software was stable enough to run for months on end without a reboot. NoCatAuth gained this stability through rapid release cycles with simple bug fixes implemented in each cycle.

Most open source projects use version numbers that mirror the policy of rapid releases. For example, in case of the Linux kernel, the version numbers take the form of *x.y.z* where *x* begins at 0, and *y* and *z* are nested subcategories (Torvalds, 2005). The first release of Linux was dubbed version 0.01 and has reached version 2.6.10 today[5].

Such incremental versioning supports incremental changes and timely assimilation of feedback through the quality assurance process. Features are therefore implemented and bugs are fixed in a more organized fashion.

Another outcome of rapid release cycles is that over time, the proportion of critical system errors decreases. This observation has been tested using randomly-generated commands (called the Fuzz approach) on different operating systems, including Microsoft Windows Family, UNIX, and Linux (Miller et al., 2000). The outcome is interesting. It shows that over time, the reliability of Linux-based operating systems has improved tremendously. Better stability and increased time between crashes improves stability and reliability of the system as a whole.

The most visible downside of the rapid release process is that it does not allow for long-term thinking in some cases. For example, the Jabber project was initially created to address the problem of instant messaging (Adams, 2001). However, as the project grew, the developer community realized that the project could be extended to much more than just instant messaging. In its next major release beyond the 1.4 series, the Jabber Software Foundation decided to release a version 2.0, a complete rewrite of the design and its code (XMPP.org, 2005). Both series 1.4 and 2.0 co-exist and are actively used based on feature preferences.

Given Enough Eyeballs…

Feedback is a very important cornerstone of the OSS process. Since the source code is made available for peer review, skilled programmers not only find problems but also fix them on their own. Even the non-programmer community can contribute by looking for the odd behavior in performance and user interface design. This is the essence of the third principle: *given enough eyeballs, all bugs are shallow,* also called *Linus' Law* after Linus Torvalds, the founder of Linux (Raymond, 1999).

Easy access to source code facilitates rigorous peer review and parallel debugging among many geographically dispersed programmers, therefore enabling rapid evolution of *high quality* software (Koch & Schneider, 2002; Stamelos et al., 2002). In addition, the mass of software users are directly and actively involved in the development process. Their bug reports tend to be very detailed because they generally cover problems that originated in different kinds of systems with various hardware/software configurations. Their suggestions and feedback regarding features

Table 1. Comparing software methodologies (Adapted from Abrahamsson et al., 2002)

Entity	**A gile Methods**	**Open Source-based (bazaar style)**	**Plan-driven (cathedral style)**
Producers	Rapid, Localized, Collaborative	Geographically distributed, Collaborative, Rapid	Adequate skill-set, plan-oriented, usually localized
Consumers	Dedicated, Knowledgeable, Collaborative, Empowered	Dedicated, Knowledgeable, Collaborative, Empowered	Representative, plan-driven
Design approach	Based on current environment	Based on current environment, but open for change	Based on current and foreseeable environments
Team management	Smaller team dynamics, Face-to-face	Dispersed teams, loose team dynamics	Larger teams
Goal	Rapid Value	Challenging problem (itch metaphor)	High assurance
Time scale	Short	Short	Long

and solutions of the software also promote more functional and user-friendly design.

This approach also goes against the grain of the *security through obscurity* concept that is often leveraged by proprietary software companies (Hissam et al., 2002). This is a controversial principle in security since it attempts to use secrecy to ensure security (Wikipedia, 2005). The essence of this concept is that since proprietary software does not make source code available, the risk of discovering security vulnerabilities is minimized. However, this approach is also akin to the practice of hiding the key under the doormat, so that it remains hidden. It is almost universally known that the first place to look for keys is under the doormat. Once the vulnerability is discovered, all is lost, until the vendor chooses to fix the problem. With open source projects, vulnerabilities get fixed either by the vendor, or by someone in the community. It appears that the open source process fosters better notions of security from the code perspective.

COMPARISON OF SOFTWARE METHODOLOGIES

Many approaches within the open source community are similar to the ones followed within proprietary processes. It is important to understand that the programming language or framework that is used in either case is not significantly different. Programs written as open source or closed are essentially only instructions for a computer. What differs is its visibility and management. Closed source projects have programmers and reviewers, but these members are most likely paid employees of the same organization, where the goal is to maximize profit. OSS communities are not driven by monetary incentives. They are largely driven by ego, trust (Mui et al., 2005), and a desire to produce free code.

Open source software development has formalized in the last few years, but most of the work is done based on the direction from a very small core group as in the *benevolent dictator* model (Hamm, 2004) or by a group of developers who take on roles based on their quality of work and intentions as in the *meritocracy* model (Perens, 1997; Young, 1958). Linux is the primary example of a project where the central authority is Linus Torvalds, who relies on the consensus of a few other core contributors. Linus is therefore considered to be the benevolent dictator. On the other hand, a project like Debian, which is a community-driven distribution of Linux, chooses a team to lead the project every few years. The individuals elected to these positions are chosen based on their merit; hence the term meritocracy. In either case, the projects start with

a vested interest at solving a single problem (the itch metaphor), and then evolves into something larger, if found useful.

Similar methods exist that rely on quick cycles of feedback and very rapid development schedules. Agile methods are primarily fostered by groups that are customer-oriented, and prefer to work in face-to-face environments that are usually well-funded (Alliance, 2001). Extreme programming is perhaps the most well-known of all agile methods. Table 1 presents a side-by-side comparison of open source methods, agile methods, and the classic plan-driven methods (Abrahamsson et al., 2002).

FUTURE TRENDS

In the early days of open source, it was strongly believed that the entire open source movement was being shouldered by altruistic programmers who burned the midnight oil to keep their ideals intact, an ideal of software free from bugs and free from the clutches of the enterprise (Himanen, 2001). Those stereotypes are changing rapidly (Dahlander & Magnusson, 2005). It is no longer a fly-by-night operation. Large organizations such as IBM, Oracle, General Motors, and the U.S. Department of Defense are playing a major role in the creation and maintenance of open source software (MITRE, 2002). Companies such as Sun Microsystems, known for their proprietary software, are testing the waters with different business models. Sun Microsystems recently announced the OpenSolaris project (Sun Microsystems, 2005), which aims at releasing the Solaris version 10 operating system under an open source license. These are big steps for the software industry, and it looks like this trend is moving up.

Other companies are getting into the business of providing a test platform (often called a stack) where they test a combination of software for enterprise level performance and security. The primary complaint with open source software is its lack of a guarantee. The software is provided as-is. Companies such as SpikeSource test combinations of open source Web servers, databases, and scripting environments for performance and security. Such approaches bolster confidence by providing guarantees that enterprises need. In such cases, open source is no longer viewed as unwarranted code written by lone programmers.

Enterprise involvement brings with it the question of legality. Given that code is written by so many different people, it is very difficult to avoid the conflict with an existing patent. In recent cases, SCO has taken IBM and DaimlerChrysler to court over intellectual property claims (Groklaw, 2005). Given that the legal standing of open source software is still largely untested in court, some companies shy away from using open source.

CONCLUSION

While the open source approach is more along the lines of the untamed "Wild West", it presents a compelling alternative to developing high quality software. Projects such as Apache, Mozilla, and Linux have produced equal if not better quality software when compared with their proprietary counterparts. Apache commands over 65% of the world's Web server market share (Netcraft, 2005), while Mozilla Firefox and Linux have demonstrated very rapid growth in their own domains. In spite of the risks of a community-driven project, open source presents a favorable picture from a quality perspective. Over time, open source software improves in usability, performance, security, and reliability. Additionally, it is the *modifiability* of open source that stands out as the prime attribute. It would be well worth the effort to take a closer look at the open source world and attempt to accommodate its idiosyncrasies into more well-established, formalized methods to gain a "best of both worlds" advantage.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis.* Retrieved December 1, 2005, from http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf

Adams, D. (2001). *Programming jabber.* Sebastopol, CA: O'Reilly & Associates.

Alliance, A. (2001). *Principles behind the Agile Manifesto.* Retrieved December 1, 2005, from http://www.agilemanifesto.org/principles.html

Applegate, L. M., Austin, R. D., & McFarlan, W. F. (2002). *Creating business advantage in the information age.* New York: McGraw-Hill Irwin.

Associates, O. R. (2005). *OnLAMP.com.* Retrieved December 1, 2005, from http://www.onlamp.com/

Audris, M. (2003). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Barahona, J. M. G., Quiros, P. d. l. H., & Bollinger, T. (1999). A brief history of free software and open source. *IEEE Software, 16*(1), 32-34.

Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). *Quality attributes.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass, L., Klein, M., & Bachmann, F. (2000). *Quality attribute design primitives.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass, L., Klein, M., & Bachmann, F. (2001, October 4). Quality attribute design primitives and the attribute driven design method. *Proceedings of the 4th Conference on Product Family Engineering,* Bilbao, Spain.

Bergquist, M., & Ljungberg, J. (2001). The power of gifts: Organising social relationships in open source communities. *Information Systems Journal, 11*(4), 305-320.

Dahlander, L., & Magnusson, M. G. (2005). Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy, 34*(4), 481.

Davidow, W. H., & Malone, M. S. (1992). *The virtual corporation.* New York: Harper Collins.

DiBona, C., Ockman, S., & Stone, M. (Eds.). (1999). *Open sources: Voices for the open source revolution.* Sebastapol, CA: O'Reilly.

Dotzler, A. (2003). Getting involved with Mozilla: The people, the tools, the process. In L. U. G. O. Davis (Ed.), *[electronic] Presentation at the Linux User Group of Davis.* Davis, CA. Mozilla.org.

Feller, J. (2001). Thoughts on studying open source software communities. In N. L. Russo, B. Fitzgerald, & J. I. DeGross (Eds.), *Realigning research and practice in information systems development: The social and organizational perspective* (pp. 379-388). Boise, ID: Kluwer.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. *Proceedings of the the 21st International Conference in Information Systems (ICIS 2000),* Brisbane, Queensland, Australia (pp. 58-69).

Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development.* London: Addison-Wesley.

Gallivan, M. J. (2001). Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal, 11*(4), 277-304.

Goth, G. (2001). The open market woos open source. *IEEE Software, 18*(2), 104-107.

Grenier, R., & Metes, G. (1995). *Going virtual:*

Moving your organization into the 21st century. Upper Saddle River, NJ: Prentice Hall.

Groklaw. (2005). *SCO vs. IBM Case: 2:03cv00294.* Retrieved December 1, 2005, from http://www.groklaw.net/staticpages/index.php?page=legal-docs#scovibm

Halloran, T. J., & Scherlis, W. S. (2002, May). High quality and open source software practices. *Proceedings of the International Conference on Software Engineering,* Orlando, FL.

Hamm, S. (2004). *Linus Torvalds' benevolent dictatorship.* Retrieved December 1, 2005, from http://www.businessweek.com/print/technology/content/aug2004/tc20040818_1593.htm

Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. London: Secker & Warburg.

Hissam, S. A., Plakosh, D., & Weinstock, C. (2002). Trust and vulnerability in open source software. *IEE Proceedings — Software, 149*(1), 47-51.

Ishii, H., Chang, P., & Verma, S. (2002). *Implementing secure services over a wireless network* (Tech. Rep. No. BICS 699). San Francisco State University.

Kendrick, B. (2005). *Linux releases as of 2005.02.08.* Retrieved December 1, 2005, from http://www.sonic.net/~nbs/linux-releases.txt

Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12*(1), 27-42.

Krubner, L. (2003). *Editorial history on PHP-Nuke and Post-Nuke.* Retrieved December 1, 2005, from http://www.nukecops.com/article65.html

Lessig, L. (2005). *Free culture: The nature and future of creativity.* New York: The Penguin Press.

Lucas, H. C. (1996). *The T-Form organization*. San Francisco: Jossey-Bass Publishers.

Miller, B. P., Koski, D., Lee, C. P., Maganty, V., Murthy, R., Natarajan, A., et al. (2000). *Fuzz revisited: A re-examination of the reliability of UNIX utilities and services* (Web). Madison: University of Wisconsin, Madison.

MITRE. (2002). *Use of free and open-source software (FOSS) in the U.S. Department of Defense* (Web No. MP 02 W0000101). Defense Information Systems Agency.

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000, June). A case study of open source software development: The Apache Server. *Proceedings of the the 22nd International Conference on Software Engineering,* Limerick, Ireland (pp. 263-272).

Moody, G. (2001). *Rebel code: Linux and the open source revolution.* London: Penguin.

Mowshowitz, A. (1997). Virtual organization. *Communications of the ACM, 40*(9), 30-37.

Mui, L., Verma, S., & Mohtashemi, M. (2005, February). A multi-agents model of the open source development process. *Proceedings of the International Conference on Technology, Knowledge and Society,* Berkeley, CA.

Netcraft. (2005). *Web server market share*. Retrieved December 1, 2005, from http://news.netcraft.com/archives/web_server_survey.html

Perens, B. (1997). *Debian social contract.* Retrieved December 1, 2005, from http://www.debian.org/social_contract

Perens, B. (2005). *The problem of software patents in standards.* Retrieved December 1, 2005, from http://perens.com/Articles/PatentFarming.html

Perry, W. E. (1991). *Quality assurance for information systems.* New York: QED Technical Publishing Group.

Pranevich, J. (2003). *The wonderful world of Linux 2.6*. Retrieved December 1, 2005, from http://kniggit.net/wwol26.html

Raymond, E. S. (1999). *The cathedral and the bazaar* (Vol. 1). Sebastopol, CA: O'Reilly & Associates.

Raymond, E. S., & Trader, W. C. (1999). Linux and open-source success. *IEEE Software, 16*(1), 85-89.

Robey, D., Boudreau, M.-C., & Storey, V. C. (1998). Looking before we leap: Foundations for a research program on virtual organizations and electronic commerce. In G. St-Amant & M. Amami (Eds.), *Electronic commerce: Papers from the Third International Conference on the Management of Networked Organizations* (pp. 275-290).

Rosen, L. (2005). *Open source licensing: Software freedom and intellectual property law*. Upper Saddle River, NJ: Prentice Hall.

Samoladas, I., Stamelos, I., Angelis, L., & Oikonomou, A. (2004). Open source software development should strive for even greater code maintainability. *Communications of the ACM, 47*(10), 83.

Sharma, S., Sugumaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-OSS communities. *Information Systems Journal, 12*(1), 7-26.

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open-source software development. *Information Systems Journal, 12*(1), 43-60.

Sun Microsystems. (2005). *OpenSolaris (Version 10)* [Web]. Palo Alto, CA: Author.

Technorati, I. (2005). *Tag: Pesto*. Retrieved December 1, 2005, from http://www.technorati.com/tag/pesto

Torvalds, L. (2005). Kernel Number Versioning. Retrieved December 1, 2005, from http://kernel-trap.org/mailarchive/1/message/29288/thread

Torvalds, L., & Diamond, D. (2001). *Just for fun*. London: Texere.

Verma, S., & Jin, L. (2004). Diffusion and adoption of open source software within the open source community. *Proceedings of the 35th Annual Meeting of the Decision Sciences Institute*, Boston.

Wikipedia. (2005). *Security through obscurity*. Retrieved December 1, 2005, from http://en.wikipedia.org/wiki/Security_through_obscurity

Wong, W. E., & Gokhale, S. (2005). Static and dynamic distance metrics for feature-based code analysis. *The Journal of Systems and Software, 74*(3), 283.

XMPP.org. (2005). *History of XMPP*. Retrieved December 1, 2005, from http://www.xmpp.org/history.html

Young, M. (1958). *The rise of the meritocracy* (reprint ed.). Somerset, NJ: Transaction Publishers.

ENDNOTES

1. This document was authored using OpenOffice Writer, a component of the open source productivity suite available at http://www.openoffice.org/
2. EULA is the generic term used for proprietary software licenses. Open source licenses are classified based on one of the fifty-six different licenses as certified by the Open Source Initiative. There is no common body that certifies proprietary software licenses.
3. There appears to be some recent discussion amongst the core developers of PHPNuke to completely re-write the application and to avoid all the problems they faced with

the original design. This is a significant drawback of the open source model.

[4] Mozilla still exists as a complete suite (codenamed SeaMonkey), but is used by a smaller proportion of users. The larger and more visible component is the Mozilla Firefox product.

[5] As of version 2.6.11, Linus Torvalds has proposed the use of a quad notation, wherein a release will be versioned as x.y.z.a, thereby adding one more loop to the quality assurance process.

This work was previously published in Measuring Information Systems Delivery Quality, edited by E. Duggan & J. Reichgelt, pp. 291-310, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 7.3
Agile Software Development Quality Assurance:
Agile Project Management, Quality Metrics, and Methodologies

James F. Kile
IBM Corporation, USA

Maheshwar R. Inampudi
IBM Corporation, USA

ABSTRACT

Of great interest to software development professionals is whether the adaptive methods found in agile methodologies can be successfully implemented in a highly disciplined environment and still provide the benefits accorded to fully agile projects. As a general rule, agile software development methodologies have typically been applied to non-critical projects using relatively small project teams where there are vague requirements, a high degree of anticipated change, and no significant availability or performance requirements (Boehm & Turner, 2004). Using agile methods in their pure form for projects requiring either high availability, high performance, or both is considered too risky by many practitioners (Boehm et al., 2004; Paulk, 2001). When one investigates the various agile practices, however, one gets the impression that each may still have value when separated from the whole. This chapter discusses how one team was able to successfully drive software development quality improvements and reduce overall cycle time through the introduction of several individual agile development techniques. Through the use of a common-sense approach to software development, it is shown that the incorporation of individual agile techniques does not have to entail additional risk for projects having higher availability, performance, and quality requirements.

INTRODUCTION

Traditional software development approaches, perhaps best represented by the capability maturity model for software (SW-CMM) (Paulk, Curtis, Chrissis, & Weber, 1993) and its successor the capability maturity model for software integration (CMMI®) (Chrissis, Konrad, & Shrum, 2003), focus on a disciplined approach to software development that is still widely used by organizations as a foundation for project success. While the strength of traditional development methods is their ability to instill process repeatability and standardization, they also require a significant amount of organizational investment to ensure their success. Organizations that have done well using traditional approaches can also fall victim of their success through a strict expectation that history can always be repeated (Zhiying, 2003) when the environment becomes uncertain.

Agile development practices have frequently been presented as revolutionary. There is some evidence, however, that they can offer an alternative common-sense approach when applied to traditional software engineering practices (Paulk, 2001). Perhaps they can be used in part to improve the development processes of projects that do not fit the usual agile model (e.g., critical systems with high availability requirements)? Indeed, it has been suggested that project risk should be the driving factor when choosing between agile and plan-driven methods (Boehm et al., 2004) rather than overall project size or criticality. This implies that certain components of *any* project may be well suited to agility while others may not.

This chapter discusses how agile methods were used on one team to successfully drive software development quality improvements and reduce overall cycle time. This is used as a framework for discussing the impact of agile software development on people, processes, and tools. Though the model project team presented is relatively small (eight people), it has some decidedly non-agile characteristics: It is geographically distributed, it has no co-located developers, the resulting product has high performance and reliability requirements, and the organization's development methodology is decidedly waterfall having gained CMM® Level 5 compliance. Therefore, some of the fundamental paradigms that serve as the basis for successful agile development—extreme programming (Beck & Andres, 2005), for example—do not exist. Nevertheless, they were successfully able to implement several agile practices while maintaining high quality deliverables and reducing cycle time.

Chapter Organization

This chapter is organized as follows:

1. **Background:** Some history is given about our model project team and what led them to investigate agile methods. The concept of using a hybrid plan- and agile-driven method is also introduced.
2. **Approaching Selection:** How did our model project team decide which agile practices to use and which ones to discard? This section discusses the risk-based project management and technical approach used.
3. **Implementation:** This section presents how each selected agile practice was incorporated into the software development process.
4. **Impact:** How did the project team know the implemented agile practices were providing some benefit? This section talks generically about some of the metrics that were used to compare the project to prior projects performed by the same team and the impact the selected methods had on the project.
5. **Future Trends:** A brief discussion about what path will be taken to approach follow-on projects.
6. **Conclusion**.

BACKGROUND

How doth the little busy bee
Improve each shining hour,
And gather honey all the day
From every opening flower!

Isaac Watts, *Divine Songs, 20, Against Idleness and Mischief, 1715*

This chapter introduces several concepts about integrating agile software development techniques into a project that does not have typical agile characteristics. The information provided identifies the conditions that were present at the time our profiled project team began to incorporate agile practices into their decidedly traditional development approach. We begin with a history of a project development team that was unable to meet the expectations of its customer and was unsatisfied with the progress they were making toward meeting their goal of quickly developing a quality product that supported both high availability and high performance. Though the conditions identified are specific to this project and project team, one will most likely find them familiar.

Following an overview of the project and project team, a brief summary is given of some of the existing alternative development methodologies that formed the basis of the team's decision to attempt to integrate agile techniques. Though a short section, it provides some additional insight into the investigatory nature underway to improve the team's results.

This background presents the reader with a contextual overview that will serve to ground the topics discussed later in the chapter. It provides a starting point from which the remaining discussions are based. Because a real project team is being profiled, both the name of the project and the product has been obscured throughout the chapter.

Project and Historical Context

In 2003, a project was undertaken to replace an existing Web application used almost daily by a significant number of individuals (almost 450,000 users). This would not be an ordinary application rewrite, however. When the business analyzed how the product was being used and what its perceived shortcomings were, it became clear that the application needed to be taken in a whole new direction. A project was therefore undertaken to create an entirely new application—one that would incorporate the base functionality of the original application, yet include a significant number of functional improvements, usability enhancements, and external dependencies. This was not the first attempt at replacing this application (a prior attempt ended in failure), but it was certainly the most bold.

This original rewrite project became troubled as requirements seemed to never stabilize and critical milestones were continuously missed. Though it was a medium-sized project with approximately 18 individuals on the development team, there were almost as many analysts, testers, and reviewers and perhaps an equal number of stakeholders. It had the classic characteristics of what Ed Yourdon calls a "death march"—a project in which an unbiased risk assessment would determine that the likelihood of failure is extremely high (Yourdon, 2004). Though the project was considered a success both in delivery and quality, the personal sacrifices were extremely costly. It left the entire team feeling that there needed to be a change in how future projects would be managed and how to adapt to rapid change in the future.

Back to Basics

Interestingly, even though it was recognized that things would have to change, the first change that was made was to be sure the team adhered to what they did not hold fast to the first time: the traditional software engineering life cycle. Though

this may seem somewhat counterintuitive, part of the problems faced during the original "death march" project had to do with not maintaining proper control over changes, agreeing to a scope that could not possibly be contained within the time allotted for the project, and not properly evaluating risks and dependencies. In other words, the project team needed to be able to walk before it could run. Since traditional development methodologies were well known and had generally predictable results, they would provide the basis upon which any future process changes would be based.

Several Small Successes

In our experience, it is a common occurrence that several smaller upgrade releases follow large application enhancements or new application implementations—this was no exception. As the project team was re-learning the basics of the software engineering process, there were two opportunities to immediately put it to work and identify which areas were ripe for true improvement. The first was a 2-month cycle of enhancements. It was a small project, but there was still a significant staff on board to complete the work. Unlike the first project, this one adhered to the traditional software engineering process and was successful with respect to schedule, cost, and quality. The business customer was satisfied and somewhat relieved that the delivery was uneventful.

The second project of enhancements was slightly larger in scope, but used less staff and, therefore, had a longer duration. Again, a traditional software development process was followed and the project was successful with regard to schedule, cost, and quality. This second project became a true proof point for the team and was a source of confidence in their abilities. They proved that they could maintain control over these types of projects and deliver high quality work. On the other hand, even though requirements change activity was similar to what occurred in the original project, their ability to control the change was through rejection or re-negotiation—they were unable to accept late changes that might have improved the overall product. A prime example of this was in the area of end user usability. In the traditional software development process being used, ensuring that an application is usable had to be done after the application was essentially complete (during user acceptance). Unfortunately, this meant that there would be no time remaining in the development cycle to address any changes prior to releasing the upgraded product. The implication was that these types of "enhancements" would always have to wait for a follow-on release to be implemented.

The project team also began to recognize that their integration and subsequent testing phases consumed a significant part of the development schedule. Even though the project was generally under control, integration had become a time of heroic sleep deprivation to ensure the schedule was met. It was not the same situation as occurred in the original rewrite project, but it was significant enough that the team recognized that this part of the development process needed to be addressed differently.

Rapidly Changing Business Needs

Though our profiled project team could now be considered successful—after all, they were able to deliver on a set of scope within a defined period of time at a defined cost and with good quality results—the process modifications that they made did not allow them to keep up with the rapidly changing needs of the business. The business could not afford to have 6-9 month development cycles with no changes to the original scope. The releases they sought to put out were time sensitive. They also wanted the amount of functionality contained within each release to remain somewhat flexible. Instead, as new ideas arose, they would be added to a list of ever-increasing "future requirements" or handled as changes that would adjust the end date of the release. There was also the nagging

problem of not being able to incorporate usability defect corrections easily into the release where the defects were found without adding a separate "usability" test period with corrections prior to the final user acceptance test period. As it was, they were subjecting users to usability issues that would not be corrected until a follow-on release.

Finally, the business was looking for more out of the team and the team was looking for a better way to do things. Traditional software development practices appeared to be only part of the solution. They had learned to walk, but weren't sure yet how to run.

Delivery Challenges

As more and more functional enhancements were requested by the business, the team began to run into additional delivery challenges. Though quality, cost, and schedule were under control, they were unable to build in the most important features fast enough for the business. In fact, they found that their cycle time to complete a project had actually elongated. In essence, they had traded the chaos of the original schedule for its opposite and found that both didn't really solve their problem (though not being in chaos was infinitely better). They also found that just "following the process" had a chilling effect on their customer relationship. The practice of locking down requirements and stopping change made them appear unresponsive and prone to not delivering value. Though the initial releases following the large rewrite were successful, the sense of pending frustration was becoming palpable. Again, the team recognized that they needed to do something different.

Technical Challenges

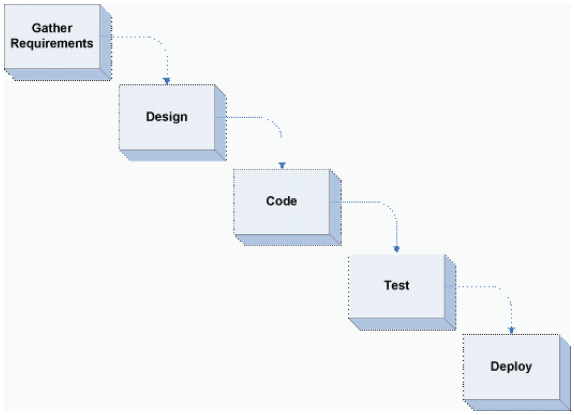
Technical challenges do not always get the same attention as other facets of software development when discussing the speed of delivery or quality for the final product, but it was a real concern to our profiled project team. Their customer was not only business-savvy, but had a keen interest in directing which technologies were used. This meant that some portion of the technical solution was imparted to the development team through the requirements gathering process. This could include individual technologies or, in one instance, the final production platform's specifications. To accommodate these types of requirements required a bit of experimentation to ensure they would work. This was something that the traditional development process did not easily support since some of the requirements themselves would derive additional requirements once investigated.

Hybrid Methodologies

Using a hybrid of adaptive and traditional software development methodologies is not as new and radical as it may at first appear. Though some of the concepts related to iterative development and other agile-like techniques can be traced back to at least two decades before the first mass-produced computer was even built (Larman & Basili, 2003), the "traditional" waterfall software development model had gained acceptance by the late 1960s when it was proposed that engineering disciplines should be used to tame wild software schedules (Naur & Randell, 1968). It derives its name from the fact that each step has distinct input and exit criteria that is supported by the surrounding steps (Figure 1). Unfortunately, the model assumes that a project goes through the process only once and that the implementation design is sound (Brooks, 1995).

Soon after being proposed, enhancements started to appear. Over time, several evolutionary techniques were developed as a compliment or replacement to the basic waterfall model including modified waterfalls, evolutionary prototyping, staged delivery, and the spiral model (Boehm, 1988; McConnell, 1996). Each enhancement recognized a failing in the original waterfall ap-

Figure 1. A traditional waterfall development model



proach and proceeded to address them within the replacement models.

Why Use a Hybrid?

Why use a hybrid development model and not adopt a single approach? The answer to this question is related to the amount of risk one can afford in their project schedule, cost, and quality. Pure waterfall models operate best with systems that require high reliability and need to be scaleable (McConnell, 1996). Our profiled project team and application has high reliability and high performance as key requirements, but they also have a highly volatile business environment in which the priority of functional enhancements frequently changes.

There is also a bit of a comfort factor in altering something one already understands; One need only learn the new techniques that replaces the original rather than an entirely new process. Over time as new techniques are introduced, the old process will no longer exist in its original form and the organization may be following a totally new methodology—one that meets their needs.

APPROACHING SELECTION

Guess if you can, choose if you dare.

Pierre Corneille, *Héraclius, act IV, sc. IV, 1674*

Great deeds are usually wrought at great risks.

Herodotus, *Histories, VII, 50, c. 485 – c. 425 B. C.*

One of the most difficult things when implementing process change is deciding which changes to make. The entire exercise is a study in risk management since choosing the wrong thing may impact the team's ability to deliver. Recall that after the tumultuous project of 2003, our profiled project team was able to deliver on time, on cost, and at a reasonable level of quality—though there was some room for improvement in the area of quality. Their challenge was to deliver faster and be more adaptable to changes that were brought forward within the development cycle. They recognized that changes needed to be made to make the team's delivery better, but they wanted to be sure that those changes did not undo the predictability they had worked so hard to attain.

The team approached these changes from two perspectives: Project management and technical. From a project management perspective, selected changes would need to be those that would enhance the delivery or quality of the project. From a technical perspective, the changes would need to be reasonable and able to enhance practitioner productivity and delivery. Making changes to one's development process is a unique experience; No two projects are the same. However, there seems to be at least two constants that we will address in the following sections prior to discussing process selection: Fear of change and overcoming that fear.

Fearing Change

Though our profiled project team recognized that there was something that needed to be done to make them a better team that could adapt to changes, deliver more quickly, and produce high quality results, some feared that tinkering with what was working could push them toward the ad hoc development process that they had already rejected. Even though they were not delivering quickly and the customer could not be characterized as completely happy, their projects seemed under control and they were no longer working 90-hour weeks.

The fear of change was manifest in several dimensions for our profiled project team. Each one, though, could be counterbalanced with a fear of not changing. This made for an interesting dance of conflicting emotions around what should be changed and what should be left alone. On one hand, they had proven their competence to their executive management. If they changed the way they do things and failed, they risked something that was tied directly to their self worth. Countering that emotion was the fear of not changing: If their customer was unhappy, the view of their competence may erode regardless.

Overcoming Fear

Fortunately for our profiled project team, their fear of not changing eventually outweighed their fear of change. They were able to recognize that if they did nothing, the situation they would find themselves in would be far worse than if they had not tried at all. Their customer was looking for something new and if the changes could be presented in that light, small bumps in the road may be looked upon more as a learning experience, than failure.

The project management and technical leadership team began to brainstorm. They came up with a plan that would make any change they implemented participative at all levels of the project and conservative so that they could assess the impact and determine if the change was good for the project. Agile practices seemed to make a lot of sense, but a piecemeal approach to change (advocated by those same agile practices) also seemed prudent. They decided that before they implemented any change, they would make sure their customer understood what they were doing and was supportive. In a sense, this approach helped them bridge the chasm between fear of change and the consequences of not changing.

It should be noted that although the project team was able to come to the conclusion that they should change and was able to overcome their fears by making some practical decisions, this was not an easy or quick process. It developed over time and with the help of the relationships they had built with others in the organization.

Process Selection

Implementing changes within any organization takes time and must be participative at all levels to be successful (Manns & Rising, 2005). To overcome the fear of making changes, the team had decided to do it in small steps—a conservative approach that would assist their evaluation of the change when the project was complete. They began by addressing two areas that seemed to cause the most trouble: Requirements prioritization and putting out a version of the release to the customer early so that initial tests—and more importantly usability tests—could be completed in time to provide feedback that could then be incorporated into the code base prior to deployment. Changes would still be controlled, but because there were to be multiple iterations, there would also be multiple integrations and system tests; they would have some flexibility to incorporate small changes from the first cycle into the second assuming they could keep the quality of the release high and they planned enough time for these anticipated changes.

When the team found they had some suc-

cess (see "Impact") with their initial changes, they became more emboldened. They suggested and implemented more changes. We discuss the areas that were addressed earliest in the next several sections. They are presented along with the reasoning behind each so that the reader can understand why each was chosen by the project team. Later in the chapter, a discussion ensues about how each practice was specifically implemented from a technical standpoint and the cycle time and quality impacts of each.

Prioritizing Requirements

One of the most difficult things facing our profiled project team was their joint ability with their customer to prioritize their requirements. On any given day, the priority may change. What seemed to be missing was a way to quantify the requirements in a manner that would permit a reasonable prioritization. In some cases, a requirement may be highly desired, but its cost would make implementation prohibitive. In other cases, a requirement may be somewhat desired, but its cost would make implementation highly desirable. A process was needed to quickly assess requirements and have the customer prioritize them so that the team was always aware of what features and functions were desired next.

Iterative Development

Partially to address their overall product quality and to gain early feedback on how a release was progressing, the team decided that some form of iterative development should be implemented. Creating products iteratively goes back to an invention theory from the 1920s and 1930s (Larman et al., 2003). It is a proven technique for addressing product quality and change. As you will see, the team's first foray into iterative development was only partially successful and required some additional process changes.

Continuous Integration

Perhaps the most frustrating part of the development process for our profiled project team was the "integration" cycle. This was where the system was put together so that it could be functionally tested as a whole. Part of the frustration with this process was that there was no easy way to see the entire system in operation from end to end without going through a lot of tedious build steps. To address this, the team decided that they would need to automate their builds and would need to permit any team member to create a locally running version of the full system at any time.

Addressing integration took on additional importance with respect to iterative development. If the team wished to create rapid iterations in the future, they could not do so without addressing the integration and build process.

Automation

One area the team thought they could gain improvements in both quality and cycle time was in the area of automation. It has long been understood that design and code inspections could significantly and positively impact the quality of a product, but the time to perform the inspections could be prohibitive for a large product. Indeed, testing also fell into this same category—large benefit, but time consuming. To address the latter concerns, the team identified automating critical reviews and testing as one of their top priorities. Tools such as JUnit, JTest, Rational Performance Tester, Findbugs (http://findbugs.sourceforge.net/), SA4J (http://www.alphaworks.ibm.com/tech/sa4j), and Parasoft's WebKing would be used (and re-used) to reduce their cycle time while improving quality.

IMPLEMENTATION

For the things we have to learn before we can do them, we learn by doing them.

Aristotle, *Nicomachean Ethics, II, 1, ca. 384-322 B. C.*

Deciding which processes to alter as discussed in "Approaching Selection" was only one facet of implementing change. Each area that was selected required a corresponding implementation action (or actions). This section of our chapter focuses on those actions that were taken to address overall quality and cycle time. Of interest are some of the reasons why certain actions were taken. As you will see, the way agility was worked into our profiled project team's project can serve as a model for other project using a hybrid development methodology where teams are looking for incremental or evolutionary (rather than revolutionary) process improvements.

Improving Quality

Perhaps one of the most vexing problems faced after the tumultuous 2003 project and even in the small step enhancement projects undertaken in 2004, was the fact that defects were being discovered and corrected late in the development cycle when they were most time consuming and most expensive to correct. Adjusting the defect detection curve such that it afforded an earlier indication into what was wrong and provided the ability to make early corrections was considered of paramount importance to improving overall code and product quality.

After taking a retrospective look back at how the product itself was developed, it became clear that not everything had been created in a manner that would be considered optimal. There were architectural and design flaws that were not necessarily apparent when the original version of the application was created, but began to impose limitations on development as enhancements were being considered—limitations that had the result of increasing the amount of time and money it would take to make those enhancements.

In addition, the original project team that worked on the first version of the product in 2003 was quite large. Due to the ad hoc nature of that project, no coding standards had been defined or adhered to. This meant that each functional area of the application was implemented and behaved differently. In effect, the application had internal vertical silos of discrete functionality.

Changes surrounding the quality of the application needed to address each of these issues: Defect detection and correction, architectural and design dependencies, and the silo effect of independently created functions. The sections that follow provide a general overview of what was implemented to address each of these concerns. We begin with a discussion about the project's quality management plan. From there, we introduce the concept of "technical stories" as a way the project team codified the refactoring of existing inefficient architectural and design constructs. Next is a description of what was done to move the defect detection and correction curve earlier in the development cycle. This is followed by a description of some of the methods and tools that would be used to enforce best coding practices. Finally, an explanation of how continuous integration might be used to improve overall code quality is given.

Quality Management Plan

Being a traditional waterfall project with a structured approach to development meant that our profiled project team had implemented a quality management plan for each of their projects. Typically, this plan would identify, along industry standard lines, the percentage and aggregated number of defects that one could expect by project phase, how those defects would be detected (e.g., inspection, testing, etc.) and how they would be

removed.

Rather than discard the quality management plan, the team thought it important to take the time to update it with the strategy they would employ to address the quality of the product. Though such a document may be derided as defying the "barely sufficient" nature of an agile project, the team found it useful to document an internal goal for the overall detection and correction of defects and the strategy they were going to use for their early elimination from the product. This document also gave them a baseline from which they could measure their intended results with their actual results.

The quality management plan, therefore, became a document that identified the goals the team would strive to achieve and the techniques they would employ to integrate agile practices into their development process. It no longer specified only industry standard information to which the project would attempt to comply, but a much higher standard to which the project team wished to be held. These changes are evident in each of he implementation practices outlined in the following sections. Each, however, was first identified as part of the project's quality management plan.

Technical Stories

One thing that was identified quickly by the profiled project team was that innovation could often be introduced into a project that would also satisfy a business requirement. In other words, the way new function was added began to have both a technical component in addition to the business component. These so-called "technical stories" became part of the requirements gathered after each release and, later on, iteration. They were influenced by a retrospective look at what went well and what did not go so well during each development cycle. As a result of these reflections, the architecture of the application was reduced and simplified through refactoring. This had the net effect of reducing the cost of ownership and change while improving the overall quality of the application by consolidating change points. The approach the team took is similar to the "user stories" concept in extreme programming.

A few possible outcomes of these "technology stories" include:

- Cost reduction as a result of simplification.
- Architecture simplification through refactoring.
- Improvement in the throughput or performance of an individual application module or area.
- Architectural changes that will re-align the application with the long-term strategy of the organization.

Defect Detection and Correction

The continuous feedback provided to the development team through the use of iterative development and continuous integration paired with automation and reuse supplied them with the opportunity to detect and correct defects earlier in the development cycle and improve overall quality. Recalling some of the difficulties the project team had with late usability changes and the difficulty they had integrating the system, two practices were introduced: Test case reuse and test case automation.

Test Case Reuse

When a project is undertaken to enhance an existing product, a common scenario that many developers face is the re-introduction of defects from prior releases when a new feature is added without understanding the overall purpose of the module (or function). One way to combat this is to retain the unit and functional test cases from release to release and execute them prior to and during each build of the system. By integrating

Figure 2. Technology-based proposals in release planning

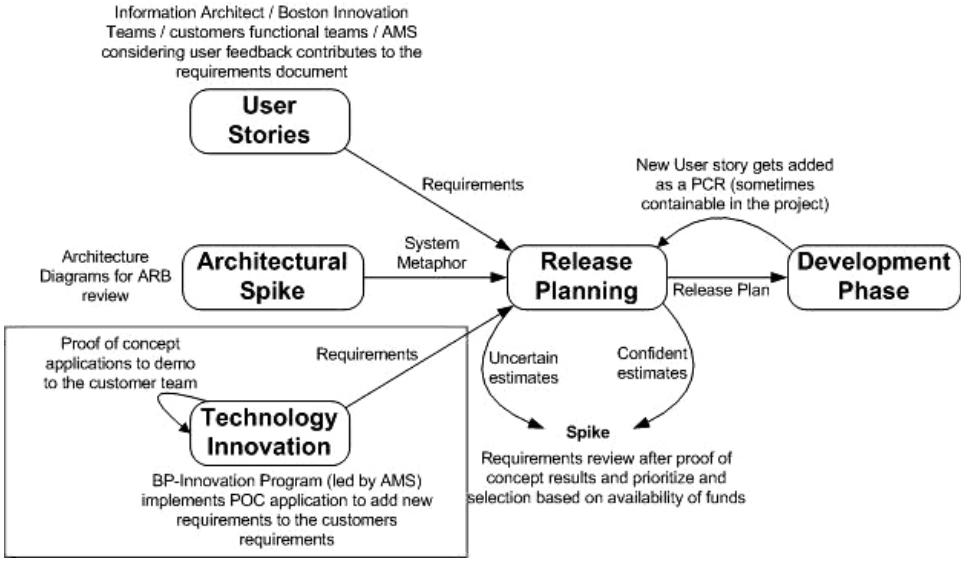
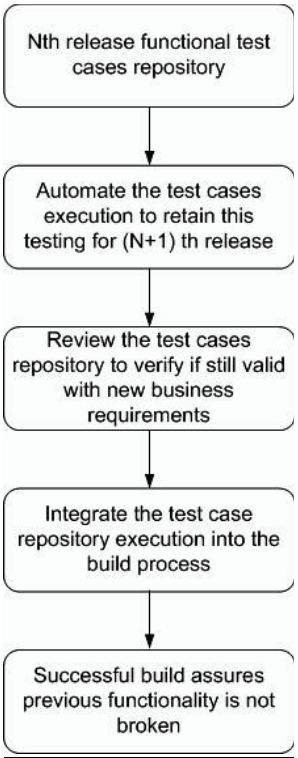


Figure 3. Test case reuse process

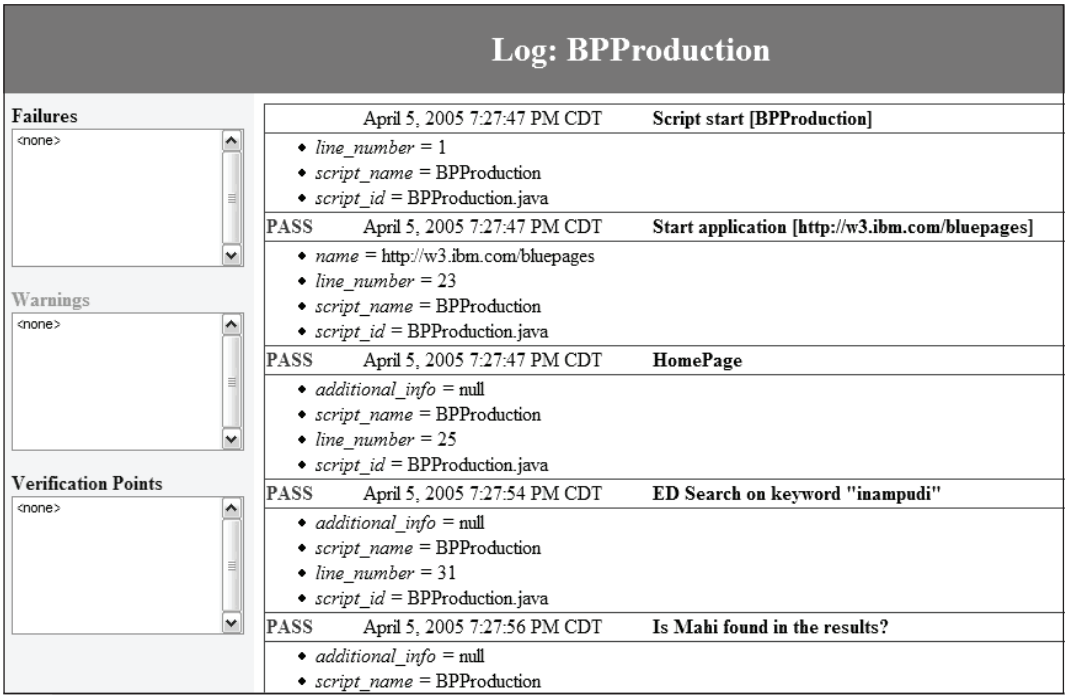


execution of the test cases with the build process, one can be assured that existing functionality is not compromised by changes made to the product; Either the new changes are incorrect or the test case is no longer valid. This reduces the number of undetected defects in a release and improves the overall quality of the application. Instead of finding issues during integration or system testing, they can be found and corrected prior to or during each system build. The theory is that the closer the defect is detected to when the change is made, the easier it will be to recall what was changed and fix it. An example of the process followed appears in Figure 3.

Automated Test Case Execution

Agile principles encourage developers to adopt test-driven development. Whether a project follows a pure agile approach, a hybrid approach (as was used here), or a traditional methodology, there is value in testing code in an automated fashion at the unit and function level. Retaining these test cases so that all developers can execute them in

Figure 4. Automated functional test cases using Rational Functional Tester



an automated fashion to ensure that their changes do not break the system is an agile principle that was implemented for this project team to measure project progress, address interim code quality, and assist in the development of new classes or methods. It should be noted that since these test cases were being built upon an existing product that did not have them initially, they were first built against those areas that required change. Those cases remained available as subsequent projects were undertaken.

Two tools were used to automate test case execution. The first, not surprisingly, was JUnit for unit testing. For functional testing, IBM's Rational Function Tester was used. This latter tool easily integrates with the build process and provides an automated functional regression testing platform for client-based and Web-based applications. A sample report appears in Figure 4.

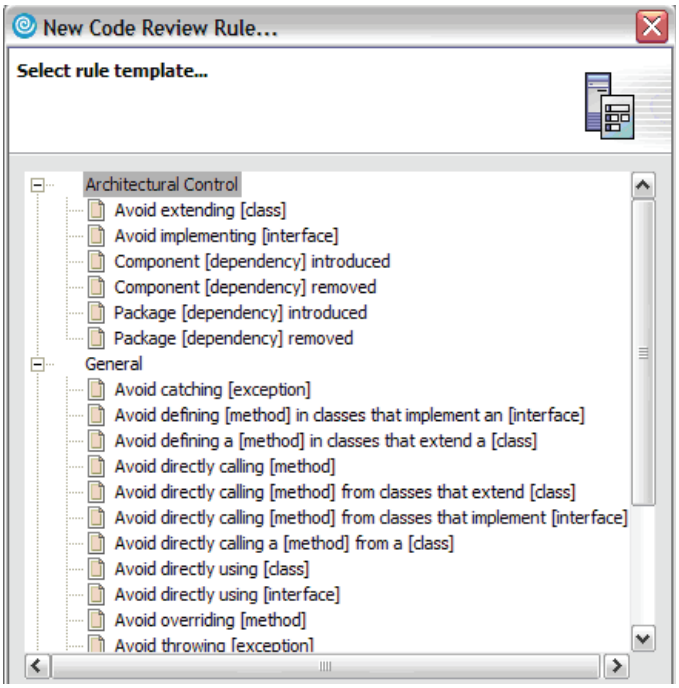
Enforce Coding Practices

One area of quality that is oftentimes not addressed by a project team is the way code will be written. Documenting the coding standards up front is helpful, but it will not ensure that an individual will not violate the project's standards or best coding practices in general. Rather than implement a series of manual code inspections, several tools were implemented to ensure best practice compliance.

Automated Code Reviewers

Tools such as Parasoft's JTest, RAD Code Reviewer, and WebKing can be plugged right into the developer's IDE. They help ensure that code is written according to a standard the team has set. They also can catch common mistakes and identify problem areas that may need to be ad-

Figure 5. Example automated code review rules



dressed. Each developer on the project team was required to install the plug-ins into their development environment and execute the review process prior to checking the code in or integrating it into the system build. An example of some of the rules used appears in Figure 5.

Tools such as IBM's Rational Application Developer Code review tool can be used to show the details and the nature of a violation including the class name and the line number of the code where the violation occurred (see Figure 6).

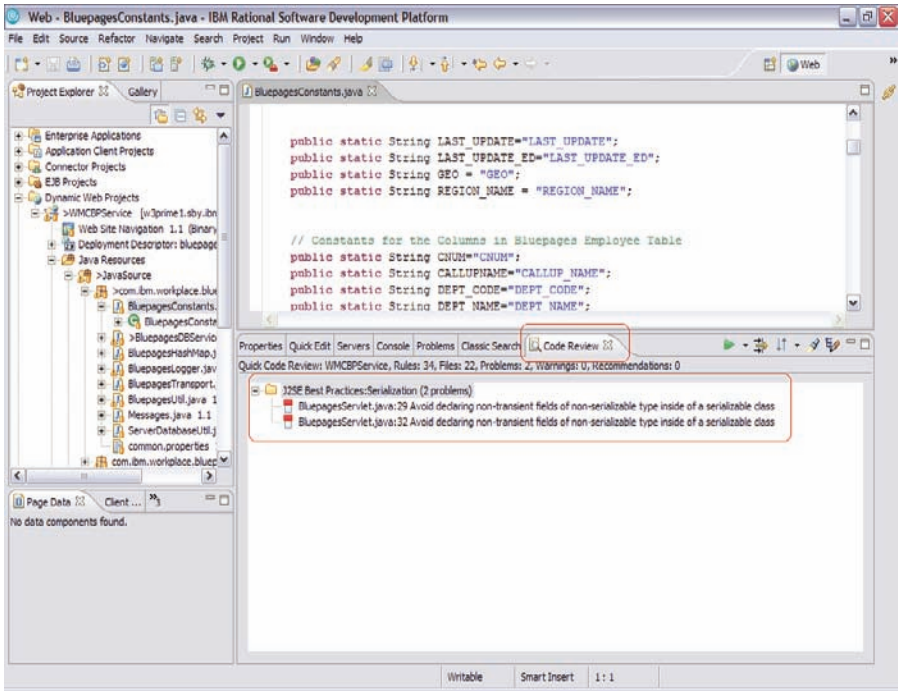
Automated Static and Stability Analysis

Static analysis tools such as Findbugs (http://findbugs.sourceforge.net/) and Structural Analyzer for Java (SA4J) (http://www.alphaworks.ibm.com/tech/sa4j) can be embedded into the build process to verify the quality of the build. These tools produce reports for the development team that help them understand potential run time defects either due to the way something was implemented or by finding architectural anti-patterns that can reduce the application's stability in the production environment.

"Continuous" Integration

One of the extreme programming practices that our profiled project team readily adopted was the concept of continuous integration. Recall that one of the most difficult project activities was the integration of created components into a functioning whole. It was felt that if the integration of the application could become more continuous—more agile—it would be possible to see the system working earlier, identify and remove integration defects in closer proximity to when they were introduced, and enforce the view that the system must always be kept in an operational state.

Figure 6. Example code review automation (IBM Rational Code Reviewer)



Automating the Build

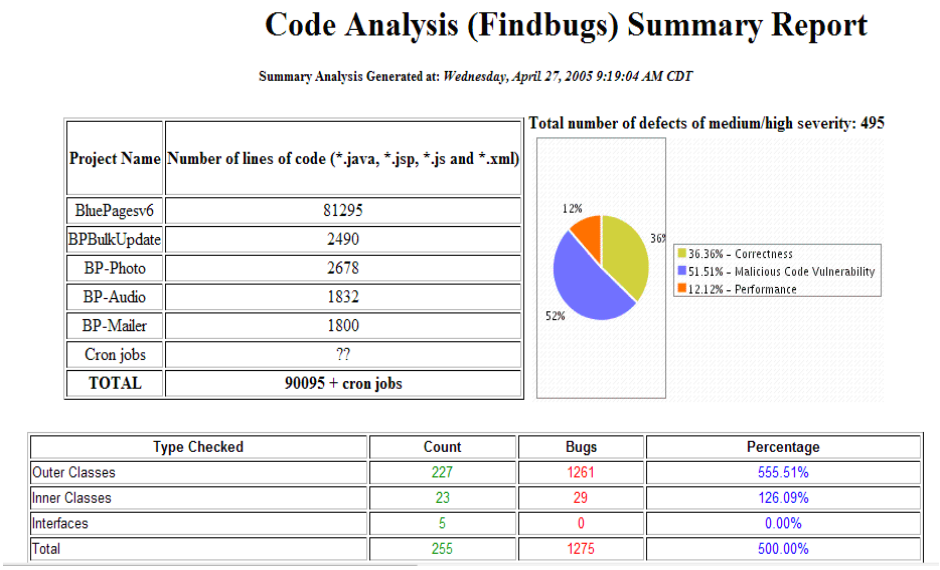
The primary barrier to continuous integration was an onerous build process. One of the technical stories, therefore, was to automate the product build such that it could be started by any developer in their own workspace and in the integration environment (under the proper controls). Apache's ANT (http://ant.apache.org), as the de facto standard for build automation, would be used as the foundation for this automated build process. In addition to automating the build itself, the script would also incorporate several of the code verification steps identified earlier: functional analysis, structural analysis, functional test verification, coding practices, etc.

The Build Process

The following process provides a general overview of the steps to be followed by the automated build process identified in several technical stories for the application.

- Pull the latest checked-in source software from the library control system (CVS).
- Execute automated code analysis tools on the extracted code to verify the code's look and feel and identify any anti-patters violations of best practices in the code base.
- Build an EAR package and execute all existing JUnit test cases against the code package and push the results to a build status Web page.
- Install the application into the runtime environment.
- Execute the automated functional test cases using Rational Functional Tester and publish the results to the build status Web page.
- Execute an overall architectural quality check using structural analysis tools (e.g., SA4J).

Figure 7. Example static analysis report



Code Analysis (Findbugs) Summary Report

Summary Analysis Generated at: *Wednesday, April 27, 2005 9:19:04 AM CDT*

Project Name	Number of lines of code (*.java, *.jsp, *.js and *.xml)
BluePagesv6	81295
BPBulkUpdate	2490
BP-Photo	2678
BP-Audio	1832
BP-Mailer	1800
Cron jobs	??
TOTAL	**90095 + cron jobs**

Total number of defects of medium/high severity: 495

Type Checked	Count	Bugs	Percentage
Outer Classes	227	1261	555.51%
Inner Classes	23	29	126.09%
Interfaces	5	0	0.00%
Total	255	1275	500.00%

Figure 8. Automated build tools stack

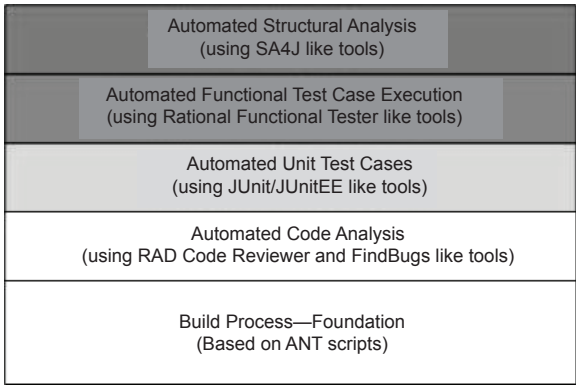
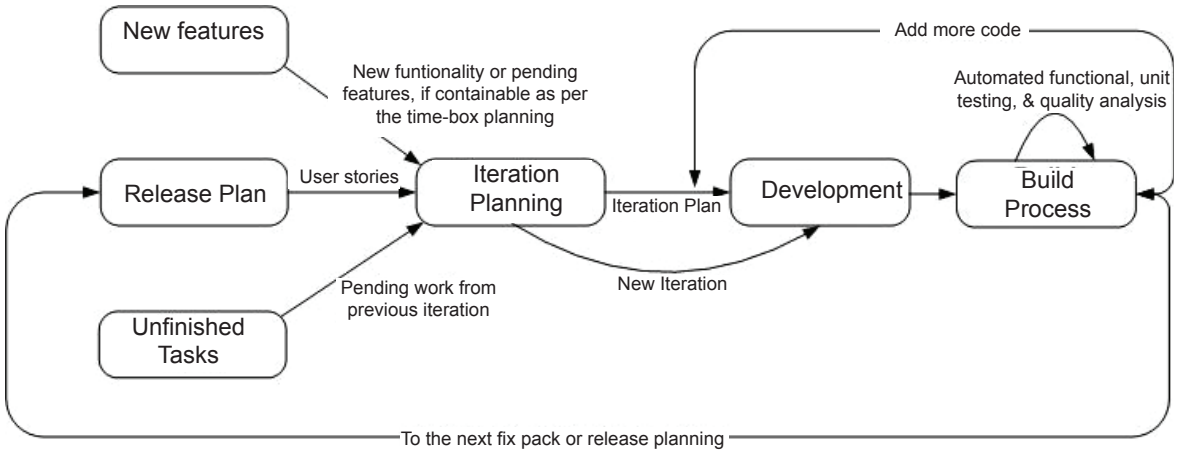


Figure 9. Continuous integration using automated tools



A graphical representation of this process appears in Figure 8.

Mapping the Build Framework to the Development Process

Every time a developer adds code to the system either as part of an iteration or as part of a release, the overall integration with existing code (and code written by others) is taken care of by the build process. This process is depicted in Figure 9.

IMPACT

Nothing quite new is perfect.

Marcus Tullius Cicero, *Brutus, 71, c. 106 B.C.-43 B.C.*

What was the overall impact of the changes that were made? By and large, the impact was positive. The team proved they could successfully integrate agile techniques into a traditional development process. What follows is a summary of some of the results. It should be noted that these are results from one team and that the experiment would need to be expanded to others to assess its validity in a broader context. Regardless, some of the results are rather remarkable.

Requirements Prioritization: Time Boxing the Schedule

As identified in the "Approaching Selection" section, instead of beginning with a fixed set of requirements from which a project sizing and project plan was derived, the most important requirements were identified and given a rough sizing. Based upon this rough sizing, the requirements were re-ordered. The schedule was broken down into discrete "time boxes" that dictated how much would be spent and how many features could be contained within a particular iteration or a project. Anything that could not fit would be re-prioritized into a follow-on project (or iteration). This method permitted the customer to introduce what they considered the most important features into the product and regularly deliver function to the business. Since the estimation at the beginning was by necessity a rough order of magnitude, as the team began the work additional adjustments would be made to the scope. If the size of a feature grew beyond what could be contained within a project, it would be discarded (or, if truly important, the date would change). If the size of a feature was smaller than what was originally anticipated, additional features would be added from the prioritized list (see Figure 10).

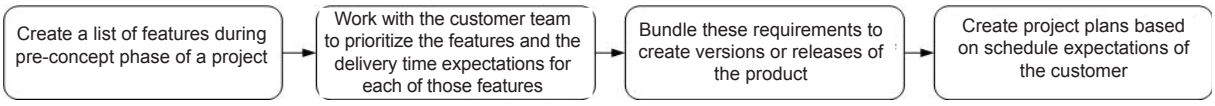
Using this approach, customer satisfaction increased from essentially failing on the first project to 97.3% on the first project that used this approach—they recognized that the project team was willing to help them meet their goals and change the way they do things in support of those goals. (Note: This satisfaction rating was for the entire release and most likely reflects the improvement in quality—discussed later—as well as the way requirements were prioritized.)

Iterative Development

Breaking each release into smaller iterations had a three-fold impact on the projects. First, the customer was able to see the results of built-in features earlier. This allowed them to re-assess the priority of the remaining requirements against changes that they may wish to implement. From a customer satisfaction perspective, the team was beginning to be seen as much more flexible—yet, the project was still under control.

The second area of positive impact area was the team's ability to have usability testing done on the parts of the application as they were produced. This meant that any usability defects identified could be rolled in to the next iteration and would be in the final build rather than waiting six months for a follow-on project to be commissioned. This had

Figure 10. Time boxed requirements prioritization



the net effect of improving end user satisfaction to above 85%—which was a significant improvement from 76% given the size of the population.

The third area of impact was in the quality of the final product. Since the system was being "put together" more frequently (also see the discussion on continuous integration results), the amount of time spent cleaning up integration errors was significantly reduced. While the initial 2003 project had greater than 400 user acceptance defects, less than a year later the user acceptance phase for all of the iterations combined had three defects (one of severity 3 and two of severity 4).

As we mentioned, not everything was positive. The way the team initially implemented iterations was not as clean as it could be. They were rushing at the end of the first iteration to get function working. This lesson learned was built into their second project—they more discretely planned the iterations so that they would have enough time to put the system together prior to the interim review by the customer. Interestingly, when they automated the build and began to use a more continuous form of integration, this additional planning was no longer required.

Continuous Integration

Perhaps one of the biggest gains the project team saw from a quality perspective was as a result of implementing their version of continuous integration. As previously discussed, this involved automating the build process such that testing occurred at each run and the system always remained in a workable state. Creating the build process cost approximately 100 labor hours to the team. The amount of time saved in integration and integration testing, however, was 300 hours and almost two weeks cycle time. On the project it was implemented in, the additional time was used to contain additional features that originally did not fit in the planned time boxes. For future projects, the additional time will be factored into the schedule as available for general development.

Automation

Although automation took on various forms including the creation of the automated build used for continuous integration, there were some additional positive impacts to cost and quality. For example, even though there was a cost to modifying automated test cases from release to release, that cost was minimal compared to creating a new test suite each time or executing all of the test cases manually. Some interesting statistics were gathered from project to project. The original project in 2003 used 12.7% of its overall budget (hours) to conduct functional and system testing (not user acceptance testing where most of the defects were eventually found). Through automation and reuse of the test case library, the two subsequent similarly sized projects consumed 5.8% and 5.2% of their budget on function and system testing respectively.

Recall that automation also added several tools that checked the stability and quality of the code. Perhaps the best measure of impact on the project is that after incorporating the recommendations for coding standard best practices and addressing structural weaknesses, the amount of time

required to maintain it was reduced by almost 10%. In a world where operation budgets are constrained, this was considered a significant under run. Most of it related to the reduced amount of call support from people who were having trouble using the application or finding obscure errors that had not been caught in the project team's own testing.

FUTURE TRENDS

In our opinion, the trend toward using agile software development practices in general and as a way to enhance the quality of products developed using traditional practices will continue. As with the profiled project team used as the basis for this chapter, we also see a trend toward using risk to identify which practices may work best in a particular environment. That will mean that projects that are not thought of as being able to easily use agile practices to enhance quality or reduce cycle time and cost today—such as those with high availability requirements or high performance requirements—may have an agile component in the future.

Smaller and More Frequent Releases

Developing a product in a piecemeal fashion predates computing by several decades. The concept of creating incremental releases to products initially grew from a desire to improve quality (Larman et al., 2003). Recent evidence has continues to show that smaller, more frequent releases have a positive impact on the overall quality of a software development project (see Madsen, 2005, for example). Several "heavier" methodologies such as the rational unified process always embraced iterations and even that has had its share of agile practices introduced as process improvements (Ambler, 2006). We expect this trend toward smaller, incremental releases with agile components to continue.

Reviews

Another future trend in agile quality management seems to be the return of peer reviews. Agile practices typically rely on up front test cases to ensure quality, but some of the current literature indicates that peer reviews still play an important role in software development. Some recent research has been conducted on focusing reviews on the most important aspects of a particular project based upon risk and the perceived value of a particular review (Lee & Boehm, 2005). This suggests that reviews themselves may also be moving toward a sufficiency model similar to agile. It will be interesting to see if a review structure will appear as part of pure agile practices.

More Hybrids

As with our profiled project team, not everyone is willing or able to move to completely agile approaches for their software development either due to perceived complexity or performance and availability requirements. We believe that the evolutionary introduction of agile practices into traditional organizations will continue, but alterations may be required for an organization to derive value as in Svensson and Host (2005). Perhaps the largest focus area in the next couple of years will be in project management. Project managers will need to not only drive the implementation of agile practices, but also need to understand their impact on their project(s) (Coram & Bohner, 2005). In all of these cases, we believe risk will most likely be the deciding factor for when agile methods are used and when they are not.

CONCLUSION

This chapter discussed how one team was able to successfully drive software development quality improvements while reducing overall cycle time through the introduction of several individual

agile development techniques. Through piecemeal change to their existing development processes, they were able to make significant improvements over time. This common-sense approach to software development showed that the incorporation of agile techniques does not have to entail additional risks for projects that have high availability, performance, and quality requirements.

REFERENCES

Ambler, S. W. (2006). *The agile edge: Unified and agile.* Software Development Retrieved January 8, 2006, from http://www.sdmagazine.com/documents/s=9947/sdm0601g/0601g.html

Beck, K., & Andres, C. (2005). *Extreme programming explained: Embrace change* (2nd ed.). Boston: Addison-Wesley.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Boston: Addison-Wesley.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Brooks, F. P. (1995). *The mythical man-month* (Anniversary Edition). Boston: Addison-Wesley.

Chrissis, M. B., Konrad, M., & Shrum, S. (2003). *CMMI: Guidelines for process integration and product improvement.* Boston: Addison-Wesley.

Coram, M., & Bohner, S. (2005, April 3-8). *The impact of agile methods on software project management.* Paper presented at the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), Greenbelt, Maryland, USA.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer, 36*(6), 47-56.

Lee, K., & Boehm, B. (2005, May 17). *Value-based quality processes and results.* Paper presented at the 3rd Workshop on Software Quality (3-WoSQ), St. Louis, Missouri.

Madsen, K. (2005, October 16-20). *Agility vs. stability at a successful start-up: Steps to progress amidst chaos and change*. Paper presented at the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05), San Diego, CA,.

Manns, M. L., & Rising, L. (2005). *Fearless change: Patterns for introducing new ideas.* Boston: Addison-Wesley.

McConnell, S. (1996). *Rapid development: Taming wild software schedules.* Redmond, Washington: Microsoft Press.

Naur, P., & Randell, B. (1968, October 7-11). *Software engineering: Report on a Conference Sponsored by the NATO Science Committee.* Paper presented at the 1st NATO Software Engineering Conference, Garmisch, Germany.

Paulk, M. C. (2001). Extreme programming from a CMM perspective. *IEEE Software, 18*(6), 19-26.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). *Capability maturity model for software, Version 1.1. Software Engineering Institute: Capability Maturity Modeling,* 82.

Svensson, H., & Host, M. (2005, March 21-23). *Introducing an agile process in a software maintenance and evolution organization.* Paper presented at the 9th European Conference on Software Maintenance and Reengineering (CSMR'05), Manchester, UK.

Yourdon, E. (2004). *Death march* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Zhiying, Z. (2003). CMM in uncertain environments. *Communications of the ACM*, *46*(8), 115-119.

This work was previously published inAgile Software Development Quality Assurance, edited by I. Stamelos & P. Sfetsos, pp. 186-205, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.4
Teaching Agile Software Development Quality Assurance

Orit Hazzan
Technion–Israel Institute of Technology, Israel

Yael Dubinsky
IBM Haifa Research Lab, Israel
Technion–Israel Institute of Technology, Israel

ABSTRACT

This chapter presents a teaching framework for agile quality—that is, the way quality issues are perceived in agile software development environments. The teaching framework consists of nine principles, the actual implementation of which is varied and should be adjusted for different specific teaching environments. This chapter outlines the principles and addresses their contribution to learners' understanding of agile quality. In addition, we highlight some of the differences between agile software development and plan-driven software development in general, and with respect to software quality in particular. This chapter provides a framework to be used by software engineering instructors who wish to base students learning on students' experiences of the different aspects involved in software development environments.

INTRODUCTION

Quality assurance (QA) is an integral and essential ingredient of any engineering process. Though there is a consensus among software practitioners about its importance, in traditional software development environments conflicts may still arise between software QA people and developers (Van Vliet, 2000, p. 125).

Agile software development methods emerged during the past decade as a response to the characteristics problems of software development processes. Since the agile methods introduced a different perspective on QA, we will call the agile approach toward quality issues *agile quality*—AQ, and will focus, in this chapter, on the teaching of AQ. By the term AQ, we refer to all the activities (e.g., testing, refactoring, requirement gathering) that deal with quality as they are manifested and applied in agile software development environments. It is important to emphasize that the term

AQ does not imply that quality changes. To the contrary, the term AQ reflects the high standards that agile software methods set with respect to software quality.

Based on our extensive experience of teaching agile software development methods both in academia and in the software industry[1], we present a teaching framework for AQ. The teaching framework consists of nine principles, the actual implementation of which is varied and should be adjusted for different specific teaching environments (e.g., academia and industry to different sizes of groups). This chapter outlines the principles and addresses their contribution to learners' understanding of AQ.

In the next section, we highlight some of the differences between agile software development and plan-driven[2] software development in general, and with respect to software quality in particular. Then, we focus on the teaching of AQ. We start by explaining why quality should be taught and, based on this understanding, we present the teaching framework for AQ, which suggests an alternative approach for the teaching of AQ. Finally, we conclude.

Agile vs. Plan-Driven Software Development

In this section, we highlight some of the main differences between agile software development and traditional, plan-driven software development. Before we elaborate on these differences, we present our perspective within which we wish to analyze these differences.

Traditional software development processes mimic traditional industries by employing some kind of production chain. However, the failure of software projects teaches us that such models do not always work well for software development processes. In order to cope with problems that result from such practices, the notion of a production chain is eliminated in agile software development environments and is replaced by a more network-oriented development process (Beck, 2000). In practice, this means that in agile teams, the task at hand is *not* divided and allocated to several different teams according to their functional description (for example, designers, developers, and testers), each of which executes its part of the task. Rather, all software development activities are intertwined and there is no passing on of responsibility to the next stage in the production chain. Thus, all team members are equally responsible for the software quality. We suggest that this different concept of the development process results, among other factors, from the fact that software is an intangible product, and therefore it requires a different development process, as well as a different approach toward the concept of software quality, than do tangible products.

Agile Development Methods vs. Plan-Driven Development Methods

During the 1990s, the agile approach toward software development started emerging in response to the typical problems of the software industry. The approach is composed of several methods and it formalizes software development frameworks that aim to systematically overcome characteristic problems of software projects (Highsmith, 2002). Generally speaking, the agile approach reflects the notion that software development environments should support communication and information sharing, in addition to heavy testing, short releases, customer satisfaction, and sustainable work-pace for all individuals involved in the process. Table 1 presents the manifesto for agile software development (http://agilemanifesto.org/).

Several differences exist between agile software development methods and plan-driven methods. Table 2 summarizes some of these differences.

Table 1. Manifesto for agile software development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

Table 2. Several differences between agile and plan-driven software development methods

	Agile Software Development Methods	**Plan-Driven Software Development Methods**
Process orientation	The development process is formulated in terms of activities that all team members apply on a daily basis.	The development process is formulated in terms of stages, in which each team member has one defined role in the process.
Formulation of requirements	Requirements are formulated in a gradual process during which customers and developers improve their understanding of the developed product. This process enables natural evolution.	Requirements are formulated in one of the first stages of the project. Therefore, the cost of implementing a change in requirements increases the later in the process it is introduced.
Customer involvement	Customers are available for discussion, clarifications, etc., in all stages of the software development process.	Primary contact with the customers occurs at the beginning of the development process.

AQ vs. Plan-Driven QA

In plan-driven software development environments, the main concept related to software quality is quality assurance, which, according to Sommerville (2001), is "The establishment of a framework of organizational procedures and standards which lead to high-quality software" (p. 537). Though this definition inspires an organizational roof for quality assurance processes, in reality, in many software organizations quality assurance is associated with a specific stage of a typical software development process and is usually carried out by the QA people who are not the developers of the code whose quality is being examined.

To illustrate the agile software development approach toward quality, we quote Cockburn (2001), who describes quality as a team characteristic:

Quality may refer to the activities or the work products. In XP, the quality of the team's program is evaluated by examining the source code work product: "All checked-in code must pass unit tests at 100% at all times." The XP team members also evaluate the quality of their activities: Do they hold a stand-up meeting every day? How often do the programmers shift programming partners? How available are the customers for questions? In some cases, quality is given a numerical value, in other cases, a fuzzy value ("I wasn't happy with the team moral on the last iteration") (p. 118).

As can be seen, within the framework of agile software development, quality refers to the *entire team* during the *entire process* of software development and it measures the code as well as the actual activities performed during the development process, both in quantitative and in qualitative terms. Accordingly, the term qual-

Table 3. Some differences between AQ and plan-driven QA

	Agile Quality (AQ)	**Plan-Driven QA**
Who is responsible for software quality?	All development team members	The QA team
When are quality-related topics addressed?	During the entire software development process; quality is one of the primary concerns of the development process	Mainly at the QA/testing stage
Status of quality-related activities relatively to other software development activities	Same as other activities	Low (Cohen, Birkin, Garfield, & Webb, 2004)
Work style	Collaboration between all role holders	Developers and QA people might have conflicts (Cohen et al., 2004)

ity assurance does not appear in agile software development as a specific stage.

In Table 3, we summarize some of the noticeable differences between the attitude toward quality of agile software development methods and of plan-driven methods, as it is manifested in many software organizations.

We note that these previous perspectives are clearly also reflected in the cultures of the two approaches toward software development. While in the context of plan-driven development, conferences are held that are dedicated to QA issues, conferences that target the community of agile software developers subsume all aspects of the development process, including AQ. This difference might, of course, be attributed to the maturity of the plan-driven software development approach; still, the observation is interesting by itself.

TEACHING AGILE SOFTWARE DEVELOPMENT QUALITY

Why Teach QA?

Naturally, software engineers should be educated for quality. The importance of this belief is reflected, for example, in the Software Engineering volume[3] of the Computing Curricula 2001, in which software quality is one of the software engineering education knowledge areas (p. 20), and is described as follows:

Software quality is a pervasive concept that affects, and is affected by all aspects of software development, support, revision, and maintenance. It encompasses the quality of work products developed and/or modified (both intermediate and deliverable work products) and the quality of the work processes used to develop and/or modify the work products. Quality work product attributes include functionality, usability, reliability, safety, security, maintainability, portability, efficiency, performance, and availability. (p. 31)

Furthermore, the software engineering code of ethics and professional practice[4], formulated by an IEEE-CS/ACM Joint Task Force, addresses quality issues and outlines how software developers should adhere to ethical behavior. Table 4 presents the eight principles of the Code. Note especially Principle 3, which focuses on quality.

Based on the assumption that the concept of quality should be taught as part of software engineering education, the question that we should ask at this stage is, How should quality be taught? Later in this section, we present our perspective on this matter. We suggest that the nature of the software

Table 4. Principles of the software engineering code of ethics and professional practice

1. Public: Software engineers shall act consistently with the public interest.
2. Client and Employer: Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
3. Product: Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. Judgment: Software engineers shall maintain integrity and independence in their professional judgment.
5. Management: Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. Profession: Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. Colleagues: Software engineers shall be fair to and supportive of their colleagues.
8. Self: Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

development methods that inspire a curriculum is usually reflected in the curriculum itself. For example, in traditional software engineering and computer science programs, QA is taught as a separate course, similar to the way in which it is applied in reality in plan-driven software development processes. Based on our teaching experience of agile software development methods, we propose that when teaching the concept of quality is integrated into a software engineering program that is inspired by agile software development, quality-related issues should and are integrated and intertwined in all topics. This idea, as well as others, is illustrated in the next section in which we present the teaching framework we have developed for teaching agile software development and illustrate how AQ integrated naturally into this teaching framework.

Teaching Framework for AQ

This section is the heart of our chapter. In what follows, we introduce our teaching framework, which is composed of nine principles, presented in Table 5 as pedagogical guidelines. Each of the principles is illustrated with respect to the teaching of AQ.

As can be seen, all principles put the learners at the center of the discussion while referring to two main aspects—cognitive and social. Specifically, Principles 1, 2, 3, and 7 emphasize the learning process from a cognitive perspective while Principles 4, 5, 6, 8, and 9 highlight the learning process from a social perspective. We note that this is not a dichotomy, but rather, each principle addresses both aspects to some degree. Accordingly, in what follows, the principles are presented in such an order that enables a gradual mental construction of the learning environment that this teaching framework inspires.

Specifically, for each principle we first describe how it is expressed when agile software development concepts are taught, and then how it is applied in the teaching of AQ.

This presentation style is consistent with our perspective of the teaching of AQ. As mentioned previously, agile software development inspires a development environment in which all activi-

Table 5. Teaching framework

• **Principle 1:** Inspire the agile concept nature.
• **Principle 2:** Let the learners experience the agile concept as much as possible.
• **Principle 3:** Elicit reflection on experience.
• **Principle 4:** Elicit communication.
• **Principle 5:** Encourage diverse viewpoints.
• **Principle 6:** Assign roles to team members.
• **Principle 7:** Be aware of cognitive aspects.
• **Principle 8:** Listen to participants' feelings toward the agile concept.
• **Principle 9:** Emphasize the agile concept in the context of the software world.

ties involved in software development processes are intertwined, and the notion of a production chain is eliminated. Accordingly, when we teach AQ we do not separate it from the teaching of the software development process (in our case, agile software development) but, rather, AQ is taught as part of the software development process in the same spirit in which the entire agile development process is taught.

This section presents, in fact, the application of our teaching framework for software development methods (presented in Dubinsky & Hazzan, 2005 and in Hazzan & Dubinsky, 2006) for the case of AQ. In Dubinsky and Hazzan (2005), we also outline the evolutionary emergence of the teaching framework and describe its implementation in a specific course (including detailed schedule and activities).

Principle 1: Inspire the Agile Concept Nature

This is a meta-principle that integrates several of the principles described later on in this section and, at the same time, is supported by them. It suggests that complex concepts in software development, such as quality or a software development method, should not be lectured about, but rather, their spirit should be inspired. In other words, the teaching of a complex (agile) concept should not be based solely on lecturers but rather, the learning of the main ideas of such concepts is more valuable if a "learning by doing" approach is applied and the (agile) concept is applied, performed, and used by the learners. Such an experience improves the learners experience and skills in the said agile concept, and at the same time, provides the teacher with opportunities to elicit reflection processes.

The application of this principle is expressed by active learning (Silberman, 1996) on which the next principle elaborates, and should take place in an environment that enables the actual performance of the agile concept.

In the case of teaching AQ, this principle implies that the learning occurs in an environment in which it would be natural to illustrate and feel the interrelation between AQ and the other activities that take place in agile software development environments. For example, the extreme programming practice of *whole team*, which states that "a variety of people work together in interlinking ways to make a project more effective" (Beck & Andres, 2004, p. 73), should be applied in order to inspire agile software development. In such software development environments, when the teacher asks the learners to expose and reflect on the relationships between AQ and the other activities, connections between AQ and other activities performed in this environment become clear.

Principle 2: Let the Learners Experience the Agile Concept as Much as Possible

This principle is derived directly from the previous one. In fact, these two principles stem from the importance attributed to the learners' experimental basis, which is essential in learning processes of complex concepts. This assertion stands in line with the constructivist perspective of learning (Davis, Maher, & Noddings, 1990; Confrey, 1995; Kilpatrick, 1987), the origins of which are rooted in Jean Piaget's studies (Piaget, 1977).

Constructivism is a cognitive theory that examines learning processes that lead to mental constructions of knowledge based upon learners' knowledge and experience. According to this approach, learners construct new knowledge by rearranging and refining their existing knowledge (Davis et al., 1990; Smith, diSessa, & Roschelle, 1993). More specifically, the constructivist approach suggests that new knowledge is constructed *gradually*, based on the learner's existing mental structures and in accordance with feedback that the learner receives both from other people with whom he or she interacts and from the different artifacts that constitute the learning environments. In this process, mental structures are developed in steps, each step elaborating on the preceding ones. Naturally, there may also be regressions and blind alleys.

We suggest that quality in general, and AQ in particular, are complex concepts. Therefore, their gradual learning process should be based on the learners' experience. One way to support and enhance such a gradual mental learning process is to adopt an active-learning teaching approach according to which learners are *active* to the extent that enables a reflective process (which is addressed by another principle later on in this chapter).

We do not claim that lecturing should be absolutely avoided in the process of teaching AQ; in fact, some aspects of AQ can and should be taught by means of lectures. Our experience, however, teaches us that the more learners *experience* AQ and *reflect* upon it, the more they improve their understanding of the essence of the topic, as well as their professional skills.

To illustrate how this principle is applied in the case of AQ, we focus on acceptance tests. Here, active learning is expressed in several ways. First, learners are active in the definition of the software requirements. Second, learners define the acceptance tests and verify that they meet the requirements. Third, they develop the acceptance tests. And fourth, they are guided to reflect both on each individual step and on the entire process. Such a complete process provides learners with a comprehensive message that both highlights each element of the AQ process and at the same time connects each of its elements to the others.

Principle 3: Elicit Reflection on Experience

The importance of introducing reflective processes into software development processes has been already discussed (Hazzan, 2002; Hazzan & Tomayko, 2003). This approach is based on Schön's *Reflective Practitioner* perspective (Schön, 1983, 1987). Indeed, it is well known in the software industry that a reflective person, who learns both from the successes and failures of previous software projects, is more likely to improve his or her own performance in the field (Kerth, 2001).

According to this principle, learners should be encouraged to reflect on their learning processes as well as on different situations in the software development process in which they participated. We note that reflection processes should not be limited to technical issues, but rather should also address feelings, work habits, and social interactions related to the software development processes.

In order to elicit learners' reflective processes, learners should be offered verbal and written means for self-expression. The ability to express one's reflections and impressions gives learners the feeling that their thoughts and feelings are of interest to the instructors. Naturally, such reflective processes might also elicit criticism and complaints. In this spirit, learners should be encouraged to express not only positive ideas, but also negative feelings and suggestions for improvement.

The teaching of AQ is a good opportunity to illustrate this principle since it allows us to address the different facets of AQ. First, we can address the technical aspect of AQ, asking learners to reflect on the actual processes of applying AQ. Specifically, learners can be asked to describe the process they went through, to indicate actions that improved their progress and actions that blocked progress and should be improved, and to suggest how the AQ process itself could be improved. Second, affective aspects can be referred to during the reflection process. For example, learners can be asked to describe their feelings during the AQ process and specifically indicate actions that encouraged them, as well as actions that discouraged them, in their pursuit of the AQ process. Finally, social issues can be addressed in such reflection processes. For example, learners can be asked to indicate what teamwork actions supported the AQ process and which interfered with that process and to suggest how such interactions should be changed so as to support the AQ process. Furthermore, experience learners can be asked to reflect both during the AQ process and after it is completed—processes that Schön calls in-action and on-action reflection, respectively.

Principle 4: Elicit Communication

Communication is a central theme in software development processes. Indeed, the success or failure of software projects is sometimes attributed to communication issues. Accordingly, in all learning situations we aim at fostering learner-learner, as well as learner-teacher communication.

When communication is one of the main ingredients of the learning environment, the idea of knowledge sharing becomes natural. Then, in turn, knowledge sharing reflects back on communication. This principle can be applied very naturally in the context of AQ since it is a multifaceted concept. During the AQ learning process, learners can be asked to identify its different facets (such as, the developer perspective, the customer perspective, its fitness to the organizational culture) and to allocate the learning of its facets to different team members—first learning them, and then subsequently teaching them to the other team members in the stage that follows. In the spirit of agile software development, it is appropriate to assign the different aspects that are to be learned to pairs of learners (rather than to individuals) in order to foster learning processes. When the team members present what they have learned to their teammates, not only do they share their knowledge, but further communication is enhanced.

Another way to foster communication is to use metaphors or "concepts from other worlds." Metaphors are used naturally in our daily life, as well as in educational environments. Generally speaking, metaphors are used in order to understand and experience one specific thing using the terms of another thing (Lakoff & Johnson, 1980; Lawler, 1999). Communication, which is based on the metaphor's concept-world, refers not only to instances in which both concept-worlds correspond to one another, but also to cases in which they do not. If both concept-worlds are identical, the metaphor is not a metaphor of that thing, but rather the thing itself. Specifically, metaphors can be useful even without specifically mentioning the concept of metaphor. For example, the facilitator may say: "Can you suggest another concept-world that may help us understand this unclear issue." Our experience indicates that learners have no problem suggesting a varied collection of con-

cept-worlds, each highlighting a different aspect of the said problem and together supporting the comprehension of the topic under discussion.

Principle 5: Encourage Diverse Viewpoints

This perspective is based on management theories that assert the added value of diversity (cf. the American Institute for Managing Diversity, http://aimd.org). In the context of agile software development, it is appropriate to start by quoting Beck et al. (2004):

Teams need to bring together a variety of skills, attitudes, and perspectives to see problems and pitfalls, to think of multiple ways to solve problems, and to implement the solutions. Teams need diversity. (p. 29)

We argue that this perspective is correct also with respect to AQ, as explained next.

Naturally, the more diverse a team is, the more diverse the perspectives elicited are. These diverse viewpoints may improve software development processes in general, and the execution of AQ in particular. Specifically, in this context diversity has several benefits. First, learners are exposed to different perspectives that they can use when communicating with people from different sectors and of different opinions. Second, the developed software product itself may be improved because when different perspectives are expressed with respect to a specific topic, the chances that subtle issues will emerge are higher. Consequently, additional factors are considered when decisions are made. Third, the creation process is questioned more when diverse opinions are expressed and, once again, this may result in a more argument-based process based on which specific decisions are made. Finally, we believe that diversity reduces resistance to new ideas and creates an atmosphere of openness toward alternative opinions. In the case of learning AQ, which inspires different work habits than the ones most learners are familiar with, such openness to a different perspective is especially important.

Principle 6: Assign Roles to Team Members

This principle suggests that each team member should have an individual role in addition to the personal development tasks for which he or she is responsible. Based on our agile teaching and research practice, we have identified 12 roles, each of which is related to at least one aspect of software development, several of which are related to AQ (Dubinsky & Hazzan, 2004a). See Table 6.

The role assignment serves as a means for distributing the responsibility for the project progress and quality among all team members. The rationale for this practice stems from the fact that one person (or a small number of practitioners) can not control and handle the great complexity involved in software development projects. When accountability is shared by all team members, each aspect of the entire process is treated by single team member, yet, at the same time, each team member feels personally responsibility for every such aspect. Indeed, both the software project and all team members benefit from this kind of organization.

More specifically, our research shows that the accumulative impact of these roles increases the software quality both from the customer's perspective and from the development perspective, for several reasons. First, the roles address different aspects of the development process (management, customer, code) and together encompass all aspects of a software development process. Second, such a role assignment increases the team members' *commitment* to the project. In order to carry out one's role successfully, each team member must gain a global view of the developed software, in addition to the execution of his or her personal development tasks. This need, in turn, increases one's responsibility toward the

Table 6. Roles in agile teams

Role	Description
Leading Group	
Coach	Coordinates and solves group problems, checks the Web forum and responds on a daily basis, leads development sessions.
Tracker	Measures the group progress according to test level and task estimations, manages studio boards, manages the group diary.
Customer Group	
End user	Performs on-going evaluation of the product, collects and processes feedback received from real end-users.
Customer	Tells customer stories, makes decisions pertaining to each iteration, provides feedback, defines acceptance tests.
Acceptance tester	Works with the customer to define and develop acceptance tests, learns and instructs test-driven development.
Maintenance Group	
Presenter	Plans, organizes, and presents iteration presentations, demos, and time schedule allocations.
Documenter	Plans, organizes, and presents project documentation: process documentation, user's guide, and installation instructions.
Installer	Plans and develops an automated installation kit, maintains studio infrastructure.
Code Group	
Designer	Maintains current design, works to simplify design, searches for refactoring tasks and ensures their proper execution.
Unit tester	Learns about unit testing, establishes an automated test suite, guides and supports others in developing unit tests.
Continuous integrator	Establishes an integration environment, publishes rules pertaining to the addition of new code using the test suite.
Code reviewer	Maintains source control, establishes and refines coding standards, guides and manages the team's pair programming.

development process. Third, the need to perform one's role successfully increases the team members' *involvement* in all parts of the developed software and leads him or her to become familiar with all software parts. If team members have only a limited view and are aware only of their own personal development tasks, they will not be able to perform their personal roles properly. Alternatively, the proposed role scheme supports knowledge sharing, participants' involvement and enhanced performances.

The software quality and the quality of the development process are reflected by three measures that serve as AQ teaching-metrics. The first is the role time measure (RTM). The RTM measures the development-hours/role-hours ratio, or in other words, the time invested in development tasks relative to the time invested in role activities. The second measure is the role communication measure (RCM), which measures the level of communication in the team at each development stage. The third measure is the role management measure (RMM), which measures the level of the project management. Data illustration of these metrics, taken from a specific academic project, can be found in Dubinsky and Hazzan (2004b).

Principle 7: Be Aware of Cognitive Aspects

This principle addresses two issues. The first deals with the idea of inspiring a process of on-going

and gradual improvement. The second addresses the fact that software development should be addressed by the individuals involved on different levels of abstraction.

It is clear that software development is a gradual process conducted in stages, each one improving upon those preceding it. In many cases, this improvement takes place in parallel to an improvement in the developers understanding of the developed application. Indeed, this principle is closely derived from the constructivist approach presented in the previous discussion of Principle 2. Accordingly, the learning environment should specifically inspire that feeling of gradual learning and elicit reflection processes when appropriate (cf. Principle 3).

We briefly present two illustrative scenarios that describe how this principle can be applied in practice. When learners try to achieve a consensus with respect to a topic of which their current knowledge is insufficient, the instructor/facilitator should guide them to postpone their final decision until a later stage. Sometimes, the instructor should guide the team to make a temporary decision based on their current knowledge, and explicitly state that in the future they will be able to update, refine, and even change the decision just made. In other cases when learners are deadlocked the moderator/instructor can stop the discussion, reflect on what has transpired, and suggest to move on, explaining that it might make more sense to readdress the issue currently blocking the development progress at a later stage when the learners' background and knowledge can solve the said problem.

As mentioned before, this principle is also related to thinking on different levels of abstraction. In a previous paper (Hazzan & Dubinsky, 2003), we suggested that during the process of software development, developers are required to think on different abstraction levels and to shift between abstraction levels, and explain how several agile practices (such as, refactoring and planning game) support this shift between abstraction level. In other words, developers must shift from a global view of the system (high level of abstraction) to a local, detailed view of the system (low level of abstraction), and vise versa. For example, when trying to understand customers' requirements during the first stage of development, developers should have a global view of the application (high level of abstraction). When coding a specific class, a local perspective (on a lower abstraction level) should be adopted. Obviously, there are many intermediate abstraction levels in between these two levels that programmers should consider during the process of software development. However, knowing how and when to move between different levels of abstraction does not always come naturally, and requires some degree of awareness. For example, a developer may remain at an inappropriate level of abstraction for too long a time, while the problem he or she faces could be solved immediately if the problem were viewed on a different (higher or lower) level of abstraction. The required shift to that different abstraction level might not be made naturally, unless one is aware that this may be a possible step toward a solution.

This principle suggests that instructors or workshop facilitators who teach agile AQ should be aware of the abstraction level on which each stage of each activity is performed. Based on this awareness, they then should decide whether to remain on this abstraction level, or, alternatively, whether there is a need to guide the participants to think in terms of a different level of abstraction. For example, when learners are engaged in design activities and tend to move to details related to the code level, it is important to guide them to stay at the appropriate (higher) level of abstraction. It is further suggested that the instructor or facilitator explicitly highlight the movement between abstraction levels and discuss with the learners the advantages that can be gained from such moves.

We note that the role assignment mentioned in the discussion of Principle 6 can also be viewed

as a means to encourage learners to look, think and examine the development process from different abstraction levels. More specifically, if a team member wishes to perform his or her individual role successfully, that is, to lead the project in the direction that the role specifies, he or she must gain a more global (abstract) view of the developed application.

Principle 8: Listen to Participants' Feelings Toward the Agile Concept

The adoption of AQ requires a conceptual change with respect to what a software development process is. In practice, when learners express emotional statements against some aspect of AQ, we propose to take advantage of this opportunity and encourage participants to describe the subject of the said statement as it is manifested in their current software development environment. As it turns out, in many cases these descriptions elicit problems in the currently used approach. Then, we explain how AQ attempts to overcome the problematic issues just raised. For example, when a statement is made against the test-driven development approach, it is a good opportunity to ask the person making this statement to describe the testing process that he or she is currently using. In some cases, this in itself is sufficient: The question highlights the test-driven development approach toward the discussed issue, and consequently, in many cases, the facial expression of the person expressing the objection immediately changes.

In all teaching situations, we propose to try sympathizing with and legitimizing learners' feelings, and being patient until learners start becoming aware of the benefits that can be gained from the new approach. In many cases, learners' objections disappeared in part after a short while. One plausible explanation is that they begin to realize that the new approach might actually support their work and improve the quality of their developed products.

Principle 9: Emphasize the Agile Concept in the Context of the Software World

This principle closes the circle that opened with the first principle—Inspire the nature of the learned concept, in our case—AQ. We believe that part of this inspiration is related to the connections made between the concept taught and the world of software engineering. Since the world of software engineering has witnessed relatively many cases in which new terms emerged and shortly after turned out to be no more than buzzwords, when teaching a new concept that requires developers to adopt a different state of mind, it is preferable to connect the new idea to the world of software development, and in our case, to connect AQ to other agile ideas. This can be done, for example, by presenting the learners with specific problems faced by the software industry (for example, the high rate of software projects that do not fit customer requirements), illustrating how the taught idea may help overcome them. Learners will then, hopefully, feel that, on the one hand, they are being introduced to a new idea that is not detached from the software industry world and is not just a passing fashion, and on the other hand, that the new approach toward quality issues emerged as a timely answer to the needs of the software industry and that it will be useful to them in the future.

In the case of teaching AQ, the need for AQ may be first explained and some problems related to traditional QA processes may be outlined. Such a broad perspective enables learners to understand the place of the agile approach in the software industry in general, and in particular, to observe that AQ is a topic that is still undergoing development.

SUMMARY

The set of principles presented in this chapter aims to establish a teaching framework within which we teach agile software development in general, and AQ in particular. A closer look at the teaching framework reveals that, in fact, its nature is similar to that of agile software development environments. Specifically, as agile software development inspires the notion of a single comprehensive framework in which all activities are performed by all team members in short cycles, with the different activities mutually contributing to one another, the framework described in this chapter also inspires an integrative teaching framework in which all principles should be adhered to at the same time, with different focuses as appropriate. Furthermore, as the assimilation of agile software development takes place in stages, the adoption of this teaching framework should also be carried out gradually, according to the culture of the environments into which the teaching framework is assimilated.

ACKNOWLEDGMENTS

Our thanks are extended to the Technion V.P.R. Fund—B. and the G. Greenberg Research Fund (Ottawa) for their support of this research.

REFERENCES

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change* (2nd ed.). Boston: Addison-Wesley.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline*. Reading, MA: Pearson Education Inc.

Cockburn, A. (2001). *Agile software development.* Boston: Addison-Wesley.

Cohen, C. F., Birkin, S. J., Garfield, M. J., & Webb, H. W. (2004). Managing conflict in software testing, *Communications of the ACM, 47*(1), 76-81.

Confrey J. (1995). A theory of intellectual development. *For the Learning of Mathematics, 15*(2), 36-45.

Davis, R. B., Maher, C. A., & Noddings, N. (1990). Constructivist views on the teaching and learning of mathematics. *Journal for Research in Mathematics Education*, Monograph Number 4, The National Council of Teachers of Mathematics.

Dubinsky, Y., & Hazzan, O. (2004a). Roles in agile software development teams. The *5th International Conference on Extreme Programming and Agile Processes in Software Engineering* (pp. 157-166). Garmisch-Partenkirchen, Germany.

Dubinsky, Y., & Hazzan, O. (2004b). Using a roles scheme to derive software project metrics. *Quantitative Techniques for Software Agile Processes Workshop, Proceedings (and selected for the Post-Proceedings) of SIGSOFT 2004*, Newport Beach, CA.

Dubinsky, Y., & Hazzan, O. (2005). A framework for teaching software development methods. *Computer Science Education*, *15*(4), 275-296.

Fowler, M., & Beck, K. (2002). *Planning extreme programming.* Boston.

Hazzan, O. (2002). The reflective practitioner perspective in software engineering education. *The Journal of Systems and Software*, *63*(3), 161-171.

Hazzan, O., & Dubinsky, Y. (2003). Bridging cognitive and social chasms in software development using extreme programming. *Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering* (pp. 47-53). Genova, Italy.

Hazzan, O., & Dubinsky, Y. (2006). Teaching framework for software development methods. Poster presented at the ICSE Educator's Track. *Proceedings of ICSE (International Conference of Software Engineering)* (pp. 703-706), Shanghai, China.

Hazzan, O., & Tomayko, J. (2003). The reflective practitioner perspective in eXtreme programming. *Proceedings of the XP Agile Universe 2003* (pp. 51-61). New Orleans, LA.

Highsmith, J. (2002). *Agile software developments ecosystems.* Boston: Addison-Wesley.

Kerth, N. (2001). *Project retrospective.* New York: Dorset House Publishing.

Kilpatrick, J. (1987). What constructivism might be in mathematics education. In J. C. Bergeron, N. Herscovics, & C. Kieran (Eds.), *Proceedings of the 11th International Conference for the Psychology of Mathematics Education (PME11)* (Vol. I, pp. 3-27). Montréal.

Lakoff, G., & Johnson, M. (1980). *Metaphors we live by.* The University of Chicago Press.

Lawler, J. M. (1999). Metaphors we compute by. In D.J. Hickey (Ed.), *Figures of thought: For college writers.* Mountain View, California: Mayfield Publishing.

Piaget, J. (1977). Problems of equilibration. In M. H. Appel, & L. S. Goldberg (Eds.), *Topics in cognitive development, volume 1: Equilibration: Theory, research, and application* (pp. 3-13). New York: Plenum Press.

Schön, D. A. (1983). *The reflective practitioner.* New York: BasicBooks.

Schön, D. A. (1987). *Educating the reflective practitioner: Toward a new design for teaching and learning in the profession.* San Francisco: Jossey-Bass.

Silberman, M. (1996). *Active learning: 101 strategies to teach any subject.* Boston: Pearson Higher Education.

Sommerville, I. (2001). *Software engineering* (6th ed.). Reading, MA: Addison-Wesley.

Smith, J. P., diSessa, A. A., & Roschelle, J. (1993). Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The Journal of the Learning Sciences, 3*(2), 115-163.

Van Vliet, H. (2000). *Software engineering: Principles and practice.* New York: John Wiley & Sons.

ENDNOTES

1. For further information about our work, please visit our Web site *Agile Software Development Methods and Extreme Programming* (http://edu.technion.ac.il/Courses/cs_methods/eXtremeProgramming/XP_Technion.htm).
2. The term "plan-driven" was introduced by Boehm et al. (2004), who divide the software development methods prevalent today into "agile" and "plan-driven."
3. This volume is part of the Joint Task Force on Computing Curricula 2001 carried out by the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM): http://sites.computer.org/ccse/SE2004Volume.pdf
4. ACM Code of Ethics and Professional Conduct: http://www.acm.org/constitution/code.html

This work was previously published in Agile Software Development Quality Assurance, edited by I. Stamelos & P. Sfetsos, pp. 171-184, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.5
Software Quality Modeling with Limited Apriori Defect Data

Naeem Seliya
University of Michigan, USA

Taghi M. Khoshgoftaar
Florida Atlantic University, USA

ABSTRACT

In machine learning the problem of limited data for supervised learning is a challenging problem with practical applications. We address a similar problem in the context of software quality modeling. Knowledge-based software engineering includes the use of quantitative software quality estimation models. Such models are trained using apriori software quality knowledge in the form of software metrics and defect data of previously developed software projects. However, various practical issues limit the availability of defect data for all modules in the training data. We present two solutions to the problem of software quality modeling when a limited number of training modules have known defect data. The proposed solutions are a semisupervised clustering with expert input scheme and a semisupervised classification approach with the expectation-maximization algorithm. Software measurement datasets obtained from multiple NASA software projects are used in our empirical investigation. The software quality knowledge learnt during the semisupervised learning processes provided good generalization performances for multiple test datasets. In addition, both solutions provided better predictions compared to a supervised learner trained on the initial labeled dataset.

INTRODUCTION

Data mining and machine learning have numerous practical applications across several domains, especially for classification and prediction problems. This chapter involves a data mining and machine learning problem in the context of software quality modeling and estimation. Software measurements and software fault (defect) data have been used in the development of models that predict software quality, for example, a software quality

classification model (Imam, Benlarbi, Goel, & Rai, 2001; Khoshgoftaar & Seliya, 2004; Ohlsson & Runeson, 2002) predicts the fault-proneness membership of program modules. A software quality model allows the software development team to track and detect potential software defects relatively early-on during development.

Software quality estimation models exploit the software engineering hypothesis that software measurements encapsulate the underlying quality of the software system. This assumption has been verified in numerous studies (Fenton & Pfleeger, 1997). A software quality model is typically built or trained using software measurement and defect data from a similar project or system release previously developed. The model is then applied to the currently under-development system to estimate the quality or presence of defects in its program modules. Subsequently, the limited resources allocated for software quality inspection and improvement can be targeted toward low-quality modules, achieving cost-effective resource utilization (Khoshgoftaar & Seliya, 2003).

An important assumption made during typical software quality classification modeling is that fault-proneness labels are available for all program modules (instances) of training data, that is, supervised learning is facilitated because all instances in the training data have been assigned a quality-based label such as fault-prone (*fp*) or not fault-prone (*nfp*). In software engineering practice, however, there are various practical scenarios that can limit availability of quality-based labels or defect data for all the modules in the training data, for example:

- The cost of running data collection tools may limit for which subsystems software quality data is collected.
- Only some project components in a distributed software system may collect software quality data, while others may not be equipped for collecting similar data.
- The software defect data collected for some program modules may be error-prone due to data collection and recording problems.
- In a multiple release software project, a given release may collect software quality data for only a portion of the modules, either due to limited funds or other practical issues.

In the training software measurement dataset the fault-proneness labels may only be known for some of the modules, that is, labeled instances, while for the remaining modules, that is, unlabeled instances, only software attributes are available. Under such a situation following the typical supervised learning approach to software quality modeling may be inappropriate. This is because a model trained using the small portion of labeled modules may not yield good software quality analysis, that is, the few labeled modules are not sufficient to adequately represent quality trends of the given system. Toward this problem, perhaps the solution lies in extracting the knowledge (in addition to the labeled instances) stored in the software metrics of the unlabeled modules.

The above described problem represents the labeled-unlabeled learning problem in data mining and machine learning (Seeger, 2001). We present two solutions to the problem of software quality modeling with limited prior fault-proneness defect data. The first solution is a semisupervised clustering with expert input scheme based on the *k-means* algorithm (Seliya, Khoshgoftaar, & Zhong, 2005), while the other solution is a semisupervised classification approach based on the expectation maximization (EM) algorithm (Seliya, Khoshgoftaar, & Zhong, 2004).

The semisupervised clustering with expert input approach is based on implementing constraint-based clustering, in which the constraint maintains a strict membership of modules to clusters that are already labeled as *nfp* or *fp*. At the end of a constraint-based clustering run a domain expert is allowed to label the unlabeled clusters, and the semisupervised clustering process is iter-

ated. The EM-based semisupervised classification approach iteratively augments unlabeled program modules with their estimated class labels into the labeled dataset. The class labels of the unlabeled instances are treated as missing data which is estimated by the EM algorithm. The unlabeled modules are added to the labeled dataset based on a confidence in their prediction.

A case study of software measurement and defect data obtained from multiple NASA software projects is used to evaluate the two solutions. To simulate the labeled-unlabeled problem, a sample of program modules is randomly selected from the JM1 software measurement dataset and is used as the initial labeled dataset. The remaining JM1 program modules are treated (without their class labels) as the initial unlabeled dataset. At the end of the respective semisupervised learning approaches, the software quality modeling knowledge gained is evaluated by using three independent software measurement datasets.

A comparison between the two approaches for software quality modeling with limited apriori defect data indicated that the semisupervised clustering with expert input approach yielded better performance than EM-based semisupervised classification approach. However, the former is associated with considerable expert input compared to the latter. In addition, both semisupervised learning schemes provided an improvement in generalization accuracy for independent test datasets.

The rest of this chapter is organized as follows: some relevant works are briefly discussed in the next section; the third and fourth sections respectively present the semisupervised clustering with expert input and the EM-based semisupervised classification approaches; the empirical case study, including software systems description, modeling methodology, and results are presented in the fifth section. The chapter ends with a conclusion which includes some suggestions for future work.

RELATED WORK

In the literature, various methods have been investigated to model the knowledge stored in software measurements for predicting quality of program modules. For example, Schneidewind (2001) utilizes logistic regression in combination with Boolean discriminant functions for predicting *fp* program modules. Guo, Cukic, and Singh (2003) predict *fp* program modules using Dempster-Shafer networks. Khoshgoftaar, Liu and Seliya (2003) have investigated genetic programming and decision trees (Khoshgoftaar, Yuan, & Allen, 2000), among other techniques. Some other works that have focused on software quality estimation include Imam et al. (2001), Suarez and Lutsko (1999) and Pizzi, Summers, and Pedrycz (2002).

While almost all existing works on software quality estimation have focused on using a supervised learning approach for building software quality models, very limited attention has been given to the problem of software quality modeling and analysis when there is limited defect data from previous software project development experiences. In a machine learning classification problem when both labeled and unlabeled data are used during the learning process, it is termed as semisupervised learning (Goldman, 2000; Seeger, 2001). In such a learning scheme the labeled dataset is iteratively augmented with instances (with predicted class labels) from the unlabeled dataset based on some selection measure. Semisupervised classification schemes have been investigated across various domains, including content-based image retrieval (Dong & Bhanu, 2003), human motion and gesture pattern recognition (Wu & Huang, 2000), document categorization (Ghahramani & Jordan, 1994; Nigam & Ghani, 2000), and software engineering (Seliya et al., 2004). Some of the recently investigated techniques for semisupervised classification include the EM algorithm (Nigam, McCallum, Thrun, & Mitchell, 1998), cotraining (Goldman

& Zhou, 2000; Mitchell, 1999; Nigam & Ghani, 2000), and support vector machine (Demirez & Bennett, 2000; Fung & Mangasarian, 2001).

While many works in semisupervised learning are geared toward the classification problem, a few studies investigate semisupervised clustering for grouping of a given set of text documents (Zeng, Wang, Chen, Lu, & Ma, 2003; Zhong, 2006). A semisupervised clustering approach has some benefits over semisupervised classification. During the semisupervised clustering process additional classes of data can be obtained (if desired) while the semisupervised classification approach requires the prior knowledge of all possible classes of the data. The unlabeled data may form new classes other than the pre-defined classes for the given data. Pedrycz and Waletzky (1997) investigate semisupervised clustering using fuzzy logic-based clustering for analyzing software reusability. In contrast, this study investigates semisupervised clustering for software quality estimation.

The labeled instances in a semisupervised clustering scheme have been used for initial seeding of the clusters (Basu, Banerjee, & Mooney, 2002), incorporating constraints in the clustering process (Wagstaff & Cardie, 2000), or providing feedback subsequent to regular clustering (Zhong, 2006). The seeded approach uses the labeled data to initialize cluster centroids prior to clustering. The constraint-based approach keeps a fixed-grouping of the labeled data during the clustering process. The feedback-based approach uses the labeled data to adjust the clusters after executing a regular clustering process.

SEMISUPERVISED CLUSTERING WITH EXPERT INPUT

The basic purpose of a semisupervised approach during clustering is to aid the clustering algorithm in making better partitions of instances in the given dataset. The semisupervised clustering approach presented is a constraint-based scheme that uses labeled instances for initial seeding (centroids) of some clusters among the maximum allowable clusters when using *k-means* as the clustering algorithm. In addition, during the semisupervised iterative process a domain (software engineering) expert is allowed to label additional clusters as either *nfp* or *fp* based on domain knowledge and some descriptive statistics of the clusters.

The data in a semisupervised clustering scheme consists of a small set of labeled instances and a large set of unlabeled instances. Let D be a dataset of labeled (*nfp* or *fp*) and unlabeled (*ul*) program modules, containing the subsets L of labeled modules and U of unlabeled modules. In addition, let the dataset L consist of subsets L_nfp of *nfp* modules and L_fp of *fp* modules. The procedure used in our constraint-based semisupervised clustering approach with *k-means* is summarized next:

1. **Obtain initial numbers of *nfp* and *fp* clusters:**
   - An optimal number of clusters for the *nfp* and *fp* instances in the initial labeled dataset are obtained using the C_g criterion proposed by Krzanowski and Lai (1988).
   - Given L_nfp, execute the C_g criterion algorithm to obtain the optimal number of *nfp* clusters among $\{1, 2, \ldots, Cin\_nfp\}$ number of clusters, where Cin_nfp is the user-defined maximum number of clusters for L_nfp. Let p denote the obtained number of *nfp* clusters. Given L_fp, execute the C_g criterion algorithm to obtain the optimal number of *fp* clusters among $\{1, 2, \ldots, Cin\_fp\}$ number of clusters, where Cin_fp is the user-defined maximum number of clusters for L_fp. Let q denote the obtained number of *fp* clusters.
2. **Initialize centroids of clusters:** Given the maximum number of clusters, C_{max}, allowed

during the semisupervised clustering process with *k-means*,

- The centroids of p clusters out of C_{max} are initialized to centroids of the clusters labeled as *nfp*.
- The centroids of q clusters out of $\{C_{max} - p\}$ are initialized to centroids of the clusters labeled as *fp*.
- The centroids of the remaining r (i.e., $C_{max} - p - q$) clusters are initialized to randomly selected instances from U. We randomly select 5 unique sets of r instances each for initializing centroids of the unlabeled clusters. Thus, centroids of the $\{p + q + r\}$ clusters can be initialized using 5 different combinations.
- The sets of *nfp*, *fp*, and unlabeled clusters are thus,

$C\_nfp = \{c\_nfp_1, c\_nfp_2, \ldots, c\_nfp_p\}$,
$C\_fp = \{c\_fp_1, c\_fp_2, \ldots, c\_nfp_q\}$
$C\_ul = \{c\_ul_1, c\_ul_2, \ldots, c\_ul_r\}$

3. **Execute constraint-based clustering:**
   - The *k-means* clustering algorithm with the Euclidean distance function is run on D using the initialized centroids for the C_{max} clusters, and under the constraint that *the existing membership of a program module to a labeled cluster remains unchanged*. Thus, at a given iteration during the semisupervised clustering process, if a module already belongs (initial membership or expert-based assignment from previous iterations) to a *nfp* (or *fp*) cluster, then it cannot move to another cluster during the clustering process of that iteration.
   - The constraint-based clustering process with *k-means* is repeated for each of the 5 centroid initializations, and the respective SSE (sum-of-squares-error) values are computed.
   - The clustering result associated with the median SSE value is selected for continuation to the next step. This is done to minimize the likelihood of working with a lucky/unlucky initialization of cluster centroids.
4. **Expert-based labeling of clusters:**
   - The software engineering expert is presented with descriptive statistics of the r unlabeled clusters, and is asked to label them as either *nfp* or *fp*. The specific statistics presented for attributes of instances in each cluster depends on the expert's request, and include data such as minimum, maximum, mean, standard deviation, and so forth.
   - The expert labels only those clusters for which he/she is very confident in the label estimation.
   - If the expert labels at least one of the r (unlabeled) clusters, then go to Step 2 and repeat, otherwise continue.
5. **Stop semisupervised clustering:** The iterative process is stopped when the sets *C_nfp*, *C_fp*, and *C_ul* remain unchanged. The modules in the *nfp* (*fp*) clusters are labeled and recorded as *nfp* (*fp*), while those in the *ul* clusters are not assigned any label. In addition, the centroids of the $\{p + q\}$ labeled clusters are also recorded.

SEMISUPERVISED CLASSIFICATION WITH EM ALGORITHM

The expectation maximization (EM) algorithm is a general iterative method for maximum likelihood estimation in data mining problems with incomplete data. The EM algorithm takes an iterative approach consisting of replacing missing data with estimated values, estimating model parameters, and re-estimating the missing data values. An iteration of EM consists of an E or Expectation

step and an M or Maximization step, with each having a direct statistical interpretation.

We limit our EM algorithm discussion to a brief overview, and refer the reader to Little and Rubin (2002) and Seliya et al. (2004) for a more extensive coverage. In our study, the class value of the unlabeled software modules is treated as missing data, and the EM algorithm is used to estimate the missing values. Many multivariate statistical analysis, including multiple linear regression, principal component analysis, and canonical correlation analysis are based on the initial study of the data with respect to the sample mean and covariance matrix of the variables. The EM algorithm implemented for our study on semisupervised software quality estimation is based on maximum likelihood estimation of missing data, means, and covariances for multivariate normal samples (Little et al., 2002).

The E and M steps continue iteratively until a stopping criterion is reached. Commonly used stopping criteria include specifying a maximum number of iterations or monitoring when the change in the values estimated for the missing data reaches a plateau for a specified epsilon value (Little et al., 2002). We use the latter criteria and allow the EM algorithm to converge without a maximum number of iterations, that is, iteration is stopped if the maximum change among the means or covariances between two consecutive iterations is less than 0.0001. The initial values of the parameter set are obtained by estimating means and variances from all available values of each variable, and then estimating covariances from all available pairwise values using the computed means.

Given the *L* (labeled) and *U* (unlabeled) datasets, the EM algorithm is used to estimate the missing class labels by creating a new dataset combining *L* and *U* and then applying the EM algorithm to estimate the missing data, that is, the dependent variable of *U*. The following procedure is used in our EM-based semisupervised classification approach:

1. Estimate the dependent variable (class labels) for the labeled dataset. This is done by treating *L* also as *U*, that is, the unlabeled dataset consists of the labeled instances but without their fault-proneness labels. The EM algorithm is then used to estimate these missing class labels. In our study the *fp* and *nfp* classes are labeled 1 and 0, respectively. Consequently, the estimated missing values will approximately fall within the range 1 and 0.
2. For a given significance level α, obtain confidence intervals for the predicted dependent variable in Step 1. The assumption is that the two confidence interval boundaries delineate the *nfp* and *fp* modules. Record the upper boundary as *ci_nfp* (i.e., closer to 0) and the lower boundary as *ci_ fp* (i.e., closer to 1).
3. For the given *L* and *U* datasets, estimate the dependent variable for *U* using EM.
4. An instance in *U* is identified as *nfp* if it's predicted dependent variable falls within (i.e., is lower than) the upper boundary, that is, *ci_nfp*. Similarly, an instance in *U* is identified as *fp* if it's predicted dependent variable falls within (i.e., is greater than) the lower bound, that is, *ci_ fp*.
5. The newly labeled instances of *U* are used to augment *L*, and the semisupervised classification procedure is iterated from Step 1. The iteration stopping criteria used in our study is such that if the number of instances selected from *U* is less than a specific number (that is, 1% of initial *L* dataset), then stop iteration.

EMPIRICAL CASE STUDY

Software System Descriptions

The software measurements and quality data used in our study to investigate the proposed semisupervised learning approaches is that of a large

NASA software project, JM1. Written in C, JM1 is a real-time ground system that uses simulations to generate certain predictions for missions. The data was made available through the Metrics Data Program (MDP) at NASA, and included software measurement data and associated error (fault or defect) data collected at the function level.

A program module for the system consisted of a function or method. The fault data collected for the system represents, for a given module, faults detected during software development. The original JM1 dataset consisted of 10,883 software modules, of which 2,105 modules had software defects (ranging from 1 to 26) while the remaining 8,778 modules were defect-free, that is, had no software faults. In our study, a program module with no faults was considered *nfp* and *fp* otherwise.

The JM1 dataset contained some inconsistent modules (those with identical software measurements but with different class labels) and those with missing values. Upon removing such modules, the dataset was reduced from 10,883 to 8,850 modules. We denote this reduced dataset as JM1-8850, which consisted of 1,687 modules with one or more defects and 7,163 modules with no defects.

Each program module in the JM1 dataset was characterized by 21 software measurements (Fenton et al., 1997): the 13 metrics as shown in Table 1 and 8 derived Halstead metrics (Halstead length, Halstead volume, Halstead level, Halstead difficulty, Halstead content, Halstead effort, Halstead error estimate, and Halstead program time. We used only the 13 basic software metrics in our analysis. The eight derived Halstead metrics were not used. The metrics for the JM1 (and other datasets) were primarily governed by their availability, internal workings of the projects, and the data collection tools used. The type and numbers of metrics made available were determined by the NASA Metrics Data Program. Other metrics, including software process measurements, were not available. The use of the specific software metrics does not advocate their effectiveness, and a different project may consider a different set of software measurements for analysis (Fenton et al., 1997; Imam et al., 2001).

Table 1. Software measurements

Line Count Metrics	Total Lines of Code Executable LOC Comments LOC Blank LOC Code And Comments LOC
Halstead Metrics	Total Operators Total Operands Unique Operators Unique Operands
McCabe Metrics	Cyclomatic Complexity Essential Complexity Design Complexity
Branch Count Metrics	Branch Count

In order to gauge the performance of the semisupervised clustering results, we use software measurement data of three other NASA projects, KC1, KC2, and KC3, as test datasets. These software measurement datasets were also obtained through the NASA Metrics Data Program. The definitions of what constituted a *fp* and *nfp* module for these projects are the same as those of the JM1 system. A program module of these projects also consisted of a function, subroutine, or method. These three projects were characterized by the same software product metrics used for the JM1 project, and were built in a similar software development organization. The software systems of the test datasets are summarized next:

- The KC1 project is a single CSCI within a large ground system and consists of 43 KLOC (thousand lines of code) of C++ code. A given CSCI comprises of logical groups of computer software components (CSCs). The dataset contains 2107 modules, of which 325 have one or more faults and 1782 have zero faults. The maximum number of faults in a module is 7.

- The KC2 project, written in C++, is the science data processing unit of a storage management system used for receiving and processing ground data for missions. The dataset includes only those modules that were developed by NASA software developers and not commercial-of-the-shelf (COTS) software. The dataset contains 520 modules, of which 106 have one or more faults and 414 have zero faults. The maximum number of faults in a software module is 13.
- The KC3 project, written in 18 KLOC of Java, is a software application that collects, processes, and delivers satellite meta-data. The dataset contains 458 modules, of which 43 have one or more faults and 415 have zero faults. The maximum number of faults in a module is 6.

Empirical Setting and Modeling

The initial *L* dataset is obtained by randomly selecting *LP* number of modules from JM1-8850, while the remaining *UP* number of modules were treated (without their fault-proneness labels) as the initial *U* dataset. The sampling was performed to maintain the approximate proportion of *nfp*:*fp* = 80:20 of the instances in JM1-8850. We considered different sampling sizes, that is, *LP* = {100, 250, 500, 1000, 1500, 2000, 3000}. For a given *LP* value, three samples were obtained without replacement from the JM1-8850 dataset. In the case of *LP* = {100, 250, 500}, five samples were obtained to account for their relatively small sizes. Due to space consideration, we generally only present results for *LP* = {500, 1000}; however, additional details are provided in (Seliya et al., 2004; Seliya et al., 2005).

When classifying program modules as *fp* or *nfp*, a Type I error occurs when a *nfp* module is misclassified as *fp*, while a Type II error occurs when a *fp* module is misclassified as *nfp*. It is known that the two error rates are inversely proportional (Khoshgoftaar et al., 2003; Khoshgoftaar et al., 2000).

Semisupervised Clustering Modeling

The initial numbers of the *nfp* and *fp* clusters, that is, *p* and *q*, were obtained by setting both *Cin_nfp* and *Cin_ fp* to 20. The maximum number of clusters allowed during our semisupervised clustering with *k-means* was set to two values: $C_{max} = \{30, 40\}$. These values were selected based on input from the domain expert and reflects a similar empirical setting used in our previous work (Zhong, Khoshgoftaar, & Seliya, 2004). Due to similarity of results for the two C_{max} values, only results for $C_{max} = 40$ are presented.

At a given iteration during the semisupervised clustering process, the following descriptive statistics were computed at the request of the software engineering expert: minimum, maximum, mean, median, standard deviation, and the 75, 80, 85, 90, and 95 percentiles. These values were computed for all 13 software attributes of modules in a given cluster. The expert was also presented with following statistics for JM1-8850 and the *U* dataset at a given iteration: minimum, maximum, mean, median, standard deviation, and the 5, 10, 15, 20, 25, 30, 35, 40, 45, 55, 60, 70, 75, 80, 85, 90 and 95 percentiles. The extent to which the above descriptive statistics were used was at the disposal of the expert during his labeling task.

Semisupervised Classification Modeling

The significance level used to select instances from the *U* dataset to augment the *L* dataset is set to $\alpha = 0.05$. Other significance levels of 0.01 and 0.10 were also considered; however, their results are not presented as the software quality estimation performances were relatively similar for the different α values. The iterative semisupervised classification process is continued until the number of instances added to *U* is less than 1% of the initial unlabeled dataset.

Semisupervised Clustering Results

The predicted class labels of the labeled program modules obtained at the end of each semisupervised clustering run are compared with their actual class labels. The average classification performance across the different samples for each *LP* and $C_{max} = 40$ is presented in Table 2. The table shows the average Type I, Type II, and Overall misclassification error rates for the different *LP* values. It was observed that for the given C_{max} value, the Type II error rates decreases with an increase in the *LP* value, indicating that with a larger initial labeled dataset, the semisupervised clustering with expert input scheme detects more *fp* modules.

In a recent study (Zhong et al., 2004), we investigated unsupervised clustering techniques on the JM1-8850 dataset. In that study, the *k-means* and *Neural-Gas* (Martinez, Berkovich, & Schulten, 1993) clustering algorithms were used at $C_{max} = 30$ clusters. Similar to this study, the expert was given descriptive statistics for each cluster and was asked to label them as either *nfp* or *fp*. In (Zhong et al., 2004), the *Neural-Gas* clustering technique yielded better classification results than the *k-means* algorithm.

For the program modules that are labeled after the respective semisupervised clustering runs, the corresponding module classification performances by the *Neural-Gas* unsupervised clustering technique are presented in Table 2. The semisupervised clustering scheme depicts better false-negative error rates (Type II) than the unsupervised clustering method. The false-negative error rates of both techniques tend to decrease with an increase in *LP*. The false-positive error rates (Type I) of both techniques tends to remain relatively stable across the different *LP* values.

A *z-test* (Seber, 1984) was performed to compare the classification performances (populations) of semisupervised clustering and unsupervised clustering. The Overall misclassifications obtained by both techniques are used as the response variable in the statistical comparison at a 5% significance level. The proposed semisupervised clustering approach yielded significantly better Overall misclassifications than the unsupervised clustering approach for *LP* values of 500 and greater.

The KC1, KC2, and KC3 datasets are used as test data to evaluate the software quality knowledge learnt through the semisupervised clustering process as compared to unsupervised clustering with *Neural-Gas*. The test data modules are classified based on their Euclidean distance from centroids of the final *nfp* and *fp* clusters at the end a semisupervised clustering run. We report

Table 2. Average classification performance of labeled modules with semisupervised clustering.

Sample Size	Semisupervised			Unsupervised		
	Type I	Type II	Overall	Type I	Type II	Overall
100	0.1491	0.4599	0.2058	0.1748	0.5758	0.2479
250	0.1450	0.4313	0.1989	0.1962	0.5677	0.2661
500	0.1408	0.4123	0.1913	0.1931	0.5281	0.2554
1000	0.1063	0.4264	0.1630	0.1778	0.5464	0.2431
1500	0.1219	0.4073	0.1759	0.1994	0.5169	0.2595
2000	0.1137	0.3809	0.1641	0.1883	0.5172	0.2503
2500	0.1253	0.3777	0.1725	0.1896	0.4804	0.2440
3000	0.1361	0.3099	0.1687	0.1994	0.4688	0.2499

the averages of the respective number of random samples for *LP* = {500, 1000}. A similar classification is made using centroids of the *nfp* and *fp* clusters labeled by the expert after unsupervised clustering with the *Neural-Gas* algorithm.

The classification performances obtained by unsupervised clustering for the test datasets are shown in Table 3. The misclassification error rates of all test datasets are rather unbalanced with a low Type I error rate and a relatively high Type II error rate. Such a classification is obviously not useful to the software practitioner since among the program modules correctly detected as *nfp* or *fp*, most are *nfp* instances—many *fp* modules are not detected.

The average misclassification error rates obtained by the respective semisupervised clustering runs for the test datasets are shown in Table 4. In comparison to the test data performances obtained with unsupervised clustering, the semisupervised clustering approach yielded noticeable better classification performances. The Type II error rates obtained by our semisupervised clustering approach were noticeably lower than those obtained by unsupervised clustering. This was accompanied, however, with higher or similar Type I error rates compared to unsupervised clustering. Though the Type I error rates were generally higher for semisupervised clustering, they were comparable to those of unsupervised clustering.

Table 3. Data performances with unsupervised clustering

Dataset	Type I	Type II	Overall
KC1	0.0617	0.6985	0.1599
KC2	0.0918	0.4151	0.1577
KC3	0.1229	0.5116	0.1594

Semisupervised Classification Results

We primarily discuss the empirical results obtained by the EM-based semisupervised software quality classification approach in the context of a comparison with those of the semisupervised clustering with expert input scheme presented in previous section. The quality-of-fit performances of the EM-based semisupervised classification approach for the initial labeled datasets are summarized in Table 5. The corresponding misclassification error rates for the labeled datasets after the respective EM-based semisupervised classification process is completed are shown in Table 6.

As observed in the Tables 5 and 6, the EM-based semisupervised classification approach improves the overall classification performances for the different *LP* values. It is also noted that the final classification performance is (generally) inversely proportional to the size of the initial labeled dataset, that is, *LP*. This is perhaps indicative of the presence of excess noise in the JM1-8850 dataset. A further insight into the presence of noise in JM1-8850 in the context of the two semisupervised learning approaches is presented in (Seliya et al., 2004; Seliya et al., 2005).

The software quality estimation performance of the semisupervised classification approach for

Table 4. Average test data performances with semisupervised clustering

Dataset	Type I	Type II	Overall
LP = 500			
KC1	0.0846	0.4708	0.1442
KC2	0.1039	0.3302	0.1500
KC3	0.1181	0.4186	0.1463
LP = 1000			
KC1	0.0947	0.3477	0.1337
KC2	0.1304	0.2925	0.1635
KC3	0.1325	0.3488	0.1528

the three test datasets is shown in Table 7. The table shows the average performance of the different samples for the *LP* values of 500 and 1000. In the case of *LP* = 1000, semisupervised clustering (see previous section) provides better prediction for the KC1, KC2, and KC3 test datasets. The noticeable difference between the two techniques for these three datasets is observed in the respective Type II error rates. While providing relatively similar or comparable Type I error rates, semisupervised clustering with expert input yields much lower Type II error rates than the EM-based semisupervised classification approach.

For *LP* = 500, the semisupervised clustering with expert input approach provides better software quality prediction for the KC1 and KC2 datasets. In the case of KC3, with a comparable Type I error rate the semisupervised clustering approach provided a better Type II error rate. In summary, the semisupervised clustering with expert input generally yielded better performance than EM-based semisupervised clustering.

We note that the preference of selecting one of the two approaches for software quality analysis with limited apriori fault-proneness data may also be based on criteria other than software quality estimation accuracy. The EM-based semisupervised classification approach requires minimal input from the expert other than incorporating the desired software quality modeling strategy. In contrast, the semisupervised clustering approach requires considerable input from the software engineering expert in labeling new program modules (clusters) as *nfp* or *fp*. However, based on our study it is likely that the effort put into the semisupervised clustering approach would yield fruitful outcome in improving quality of the software product.

Table 5. Average (initial) performance with semisupervised classification

LP	Type I	Type II	Overall
100	0.1475	0.4500	0.2080
250	0.1580	0.4720	0.2208
500	0.1575	0.4820	0.2224
1000	0.1442	0.5600	0.2273
1500	0.1669	0.5233	0.2382
2000	0.1590	0.5317	0.2335
3000	0.2132	0.4839	0.2673

Table 6. Average (final) performance with semisupervised classification

LP	Type I	Type II	Overall
100	0.0039	0.0121	0.0055
250	0.0075	0.0227	0.0108
500	0.0136	0.0439	0.0206
1000	0.0249	0.0968	0.0428
1500	0.0390	0.1254	0.0593
2000	0.0482	0.1543	0.0752
3000	0.0830	0.1882	0.1094

CONCLUSION

The increasing reliance on software-based systems further stresses the need to deliver high-quality software that is very reliable during system operations. This makes the task of

Table 7. Average test data performances with semisupervised classification

Dataset	Type I	Type II	Overall
LP = 500			
KC1	0.0703	0.7329	0.1725
KC2	0.1072	0.4245	0.1719
KC3	0.1118	0.5209	0.1502
LP = 1000			
KC1	0.0700	0.7528	0.1753
KC2	0.1031	0.4465	0.1731
KC3	0.0988	0.5426	0.1405

software quality assurance as vital as delivering a software product within allocated budget and scheduling constraints. The key to developing high-quality software is the measurement and modeling of software quality, and toward that objective various activities are utilized in software engineering practice including verification and validation, automated test case generation for additional testing, re-engineering of low-quality program modules, and reviews of software design and code.

This research presented effective data mining solutions for tackling very important yet unaddressed software engineering issues. We address software quality modeling and analysis when there is limited apriori fault-proneness defect data available. The proposed solutions are evaluated using case studies of software measurement and defect data obtained from multiple NASA software projects, made available through the NASA Metrics Data Program.

In the case when the development organization has experience in developing systems similar to the target project but has limited availability of defect data for those systems, the software quality assurance team could employ either the EM-based semisupervised classification approach or semisupervised clustering approach with expert input. In our comparative study of these two solutions for software quality analysis with limited defect data, it was shown that semisupervised clustering approach generally yielded better software quality prediction that the semisupervised classification approach. However, once again, the software quality assurance team may also want to consider the relatively higher complexity involved in the semisupervised clustering approach when making their decision.

In our software quality analysis studies with the EM-based semisupervised classification and semisupervised clustering with expert input approaches, an explorative analysis of program modules that remain unlabeled after the different semisupervised learning runs provided valuable insight into the characteristics of those modules. A data mining point of view indicated that many of them were likely noisy instances in the JM1 software measurement dataset (Seliya et al., 2004; Seliya et al., 2005). From a software engineering point of view we are interested to learn why those specific modules remain unlabeled after the respective semisupervised learning runs. However, due to the unavailability of other detailed information on the JM1 and other NASA software projects a further in-depth analysis could not be performed.

An additional analysis of the two semisupervised learning approaches was performed by comparing their prediction performances with software quality classification models built by using the C4.5 supervised learner trained on the respective initial labeled datasets (Seliya et al., 2004; Seliya et al., 2005). It was observed (results not shown) that both semisupervised learning approaches generally provided better software quality estimations compared to the supervised learners trained on the initial labeled datasets.

The software engineering research presented in this chapter can lead to further related research in software measurements and software quality analysis. Some directions for future work may include: using different clustering algorithms for the semisupervised clustering with expert input scheme, using different underlying algorithms for the semisupervised classification approach, and incorporating the costs of misclassification into the respective semisupervised learning approaches.

REFERENCES

Basu, S., Banerjee A., & Mooney, R. (2002). Semisupervised clustering by seeding. In *Proceedings of the 19th International Conference on Machine Learning*, Sydney, Australia (pp. 19-26).

Demirez, A., & Bennett, K. (2000). Optimization approaches to semisupervised learning. In M. Ferris, O. Mangasarian, & J. Pang (Eds.), *Applications and algorithms of complementarity*. Boston: Kluwer Academic Publishers.

Dong, A., & Bhanu, B. (2003). A new semisupervised EM algorithm for image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (pp. 662-667). Madison, WI: IEEE Computer Society.

Fenton., N. E., & Pfleeger, S. L. (1997). *Software metrics: A rigorous and practical approach* (2nd ed.). Boston: PWS Publishing Company.

Fung, G., & Mangasarian, O. (2001). Semisupervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, *15*, 29-44.

Ghahramani, Z., & Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 120-127). San Francisco.

Goldman, S., & Zhou, Y. (2000). Enhancing supervised learning with unlabeled data. In *Proceedings of the 17th International Conference on Machine Learning*, Stanford University, CA (pp. 327-334).

Guo, L., Cukic, B., & Singh, H. (2003). Predicting fault prone modules by the dempster-shafer belief networks. In *Proceedings of the 18th International Conference on Automated Software Engineering*, Montreal, Canada (pp. 249-252).

Imam, K. E., Benlarbi, S., Goel, N., & Rai, S. N. (2001). Comparing case-based reasoning classifiers for predicting high-risk software components. *Journal of Systems and Software, 55*(3), 301-320.

Khoshgoftaar, T. M., Liu, Y., & Seliya, N. (2003). Genetic programming-based decision trees for software quality classification. In *Proceedings of the 15th International Conference on Tools with Artificial Intelligence*, Sacramento, CA (pp. 374-383).

Khoshgoftaar, T. M., & Seliya, N. (2003). Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering Journal, 8*(4), 325-350. Kluwer Academic Publishers.

Khoshgoftaar, T. M., & Seliya, N. (2004). Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering Journal, 9*(3), 229-257. Kluwer Academic Publishers.

Khoshgoftaar, T. M., Yuan, X., & Allen, E. B. (2000). Balancing misclassification rates in classification tree models of software quality. *Empirical Software Engineering Journal, 5*, 313-330.

Krzanowski, W. J., & Lai, Y. T. (1988). A criterion for determining the number of groups in a data set using sums-of-squares clustering. *Biometrics, 44*(1), 23-34.

Little, R. J. A., & Rubin, D. B. (2002). *Statistical analysis with missing data* (2nd ed.). Hoboken, NJ: John Wiley and Sons.

Martinez, T. M., Berkovich, S. G., & Schulten, K. J. (1993). Neural-gas: Network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks, 4*(4), 558-569.

Mitchell, T. (1999). The role of unlabeled data in supervised learning. In *Proceedings of the 6th International Colloquium on Cognitive Science*, Donostia. San Sebastian, Spain: Institute for Logic, Cognition, Language and Information.

Nigam, K., & Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, McLean, VA (pp. 86-93).

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (1998). Learning to classify text from labeled and unlabeled documents. In *Proceedings of the 15th Conference of the American Association for Artificial Intelligence*, Madison, WI (pp. 792-799).

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2-3), 103-134.

Ohlsson, M. C., & Runeson, P. (2002). Experience from replicating empirical studies on prediction models. In *Proceedings of the 8th International Software Metrics Symposium*, Ottawa, Canada (pp. 217-226).

Pedrycz, W., & Waletzky, J. (1997a). Fuzzy clustering in software reusability. *Software: Practice and Experience, 27,* 245-270.

Pedrycz, W., & Waletzky, J. (1997b). Fuzzy clustering with partial supervision. *IEEE Transactions on Systems, Man, and Cybernetics, 5,* 787-795.

Pizzi, N. J., Summers, R., & Pedrycz, W. (2002). Software quality prediction using median-adjusted class labels. In *Proceedings of the International Joint Conference on Neural Networks*, Honolulu, HI (Vol. 3, pp. 2405-2409).

Schneidewind, N. F. (2001). Investigation of logistic regression as a discriminant of software quality. In *Proceedings of the 7th International Software Metrics Symposium*, London (pp. 328-337).

Seber, G. A. F. (1984). *Multivariate observations.* New York: John Wiley & Sons.

Seeger, M. (2001). *Learning with labeled and unlabeled data* (Tech. Rep.). Scotland, UK: University of Edinburgh, Institute for Adaptive and Neural Computation.

Seliya, N., Khoshgoftaar, T. M., & Zhong, S. (2004). Semisupervised learning for software quality estimation. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, Boca Raton, FL (pp. 183-190).

Seliya, N., Khoshgoftaar, T. M., & Zhong, S. (2005). Analyzing software quality with limited fault-proneness defect data. In *Proceedings of the 9th IEEE International Symposium on High Assurance Systems Engineering*, Heidelberg, Germany (pp. 89-98).

Suarez, A., & Lutsko, J. F. (1999). Globally optimal fuzzy decision trees for classification and regression. *Pattern Analysis and Machine Intelligence, 21*(12), 1297-1311.

Wagstaff, K., & Cardie, C. (2000). Clustering with instance-level constraints. In *Proceedings of the 17th International Conference on Machine Learning*, Stanford University, CA (pp. 1103-1110) .

Wu, Y., & Huang, T. S. (2000). Self-supervised learning for visual tracking and recognition of human hand. In *Proceedings of the 17th National Conference on Artificial Intelligence*, Austin, TX (pp. 243-248) .

Zeng, H., Wang, X., Chen, Z., Lu, H., & Ma, W. (2003). CBC: Clustering based text classification using minimal labeled data. In *Proceedings of the IEEE International Conference on Data Mining*, Melbourne, FL (pp. 443-450).

Zhong, S. (2006). Semisupervised model-based document clustering: A comparative study. *Machine Learning, 65*(1), 2-29.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems, 19*(2), 22-27.

This work was previously published in Knowledge Discovery and Data Mining: Challenges and Realities, edited by X. Zhu and I. Davidson, pp. 1-15, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.6
Integrating Quality Criteria and Methods of Evaluation for Software Models

Anna E. Bobkowska
Gdańsk University of Technology, Poland

ABSTRACT

Successful realization of the model-driven software development visions in practice requires high quality models. This chapter focuses on the quality of models themselves. It discusses context-free and context-dependent quality criteria for models and then moves on to methods of evaluation which facilitate checking whether a model is good enough. We use linguistic theories to understand groups of criteria and their impact on other models, software product and the process of software development. We propose a strict distinction of the impacts of visual modeling languages, models of the system and tools for quality criteria. This distinction is helpful when designing the methods of evaluation and making decision about the point in time, scope and personnel responsible for quality assessment. As the quality criteria and several methods of evaluation has usually been considered separately we propose a methodology which integrates them. Such an integrated approach provides the following benefits. It allows for designing methods of evaluation based on quality criteria and elements of the model (or modeling language) in the context of specific needs. It can be applied for management of the scope of evaluation with quality criteria as well as configuration of the method to a specific situation. It allows for flexible and efficient conduct of the evaluation with selection of the methods of evaluation. Finally, this chapter presents case studies which illustrate the approach.

INTRODUCTION

Models play the central role in model-driven software development approaches. However, only good models really support work of analysts, managers and developers. Only good models have the potential to facilitate dealing with complexity, direct micro-process of software development, enable more efficient performance of the difficult

tasks, facilitate management and increase satisfaction at work. On the opposite, "bad" models may cause mistake, waste time and be annoying in use. Despite that, much research has been done in the area of technology of model transformations and tools supporting this task. We argue that there is a need to focus on the quality of visual modeling languages (VML), which are used for creating the models as well as quality of models themselves.

This chapter takes a practical perspective to quality in modeling. In order to improve the quality of models one needs to understand quality criteria for models, know efficient methods for conducting evaluation and understand relations between them for effective configuration and management. The level of maturity of the technology is not satisfactory nowadays and one can observe several problems from the point of view of project and general knowledge.

From the point of view of the software project, there are several unrelated methods with limited scope of application. It is unclear what could be benefits of using them and how they could be integrated. Perhaps the most popular method for improving model quality are inspections or reviews with checklists. However, one often needs to customize the checklist provided with methodologies for a specific project. It is easy to remove unnecessary checkpoints, but the problem remains when deciding which checkpoints need to added. The problem of inflexible methods of evaluation appears even more clearly in the perspective of the current trend of software process configuration. It assumes that companies do not choose a process but rather they make a configuration of roles, artifacts and tasks for a given type of project within a framework. It means that model artifacts, modeling activities and model checking activities are objects of configuration. Thus, we need objective criteria to make decisions about the scope of modeling as well as methods of evaluation for software models.

From the point of view of maturity of modeling technology, some objective criteria and practical methods of evaluation are needed in order to provide frameworks for customization of the concrete software projects. There is a need to understand criteria why modeling languages should have such elements, diagrams, visualizations, etc. The methods of evaluation are useful for making efficient evaluation and quality criteria behind such decision allow for trade-offs when compromises are necessary. They could be applied by Object Management Group (OMG) members working on modeling standards, e.g., Unified Modeling Language (UML) and its profiles or Business Process Modeling Notation (BPMN). As these standards influence advances in practice, the increase of the maturity of modeling technology should result in better perception of this technology and more successful application in practice. This approach could be useful also for committees which work on application-type modeling languages, e.g., internet applications; special type of models, e.g., task models for user behavior modeling; and models for emerging technology, e.g., aspect-oriented technology. Finally, they could be applied in the area of domain specific modeling. The tasks of designing and evaluating modeling languages in this case are performed in software companies which means a universal application of the methods of visual modeling language evaluation.

The objective of this chapter is to present an approach to dealing with model quality in a way which integrates quality criteria and methods of evaluation. The quality criteria are useful for understanding which aspects are evaluated and for managing the scope of evaluation. The methods of evaluation are necessary for efficient conduct of evaluation. The integrated approach should provide the following benefits. It should allow for designing methods of evaluation based on quality criteria and elements of the model (or modeling language) in the context of specific

needs. It should be applicable for management the scope of evaluation with quality criteria as well as configuration of the method to a specific situation. It should allow for flexible and efficient conduct of the evaluation with selection of the methods of evaluation. The most important features of the solution include:

- Improved understanding of the quality of models in the perspective of selected linguistic theories and separating impacts modeling language, model and modeling tools for the quality criteria;
- Proposition of a methodology for designing methods of evaluation based on quality criteria and elements of the model (or modeling language).

BACKGROUND

Although several quality criteria and some methods for evaluating models have been already proposed, the practical integrated approach is still missing. They solve one kind of problems but fail to satisfy other requirements. This section discusses related work in the area of identifying quality criteria for both models and visual modelling languages, and then available methods of evaluation.

Traditionally, software engineering researchers proposed technical quality criteria for models and visual modeling languages (VML). It was typical, that they were related to modeling technology in general. It can be illustrated with the following examples. Models should be expressive enough to allow developers to capture all interesting strategic and tactical decision (Booch, 1994) and notation should allow to create complete, correct and consistent models (McGregor, 1998). The diagrams should be executable (Martin, 1993). A set of characteristics of a good notation include: clear and uniform mapping of concepts to symbols, ease to draw by hand, good look when printed, faxed and copied using monochrome images, consistency with past practice and self-consistency, simplicity of common case modeling, not-to-subtle distinctions, no overloading of symbols and suppressible details (Rumbaugh, 1999). More attempts to answer the question of the quality of models can be found (Firesmith et al. 1996; Hong, 1993). The set of desirable criteria for models also include: precision, constructability, expressiveness, executability, traceability, inspectability, and usability. A more recent research proposes the following key characteristics of engineering models (Selic 2003):

- Abstraction – possibility to remove or hide details irrelevant for a given viewpoint, it constitutes the means for coping with complexity;
- Understandability – presenting in a form that directly appeals to our intuition and thus reduces intellectual effort;
- Accuracy – true-to-life representation of the system;
- Predictiveness – possibility to predict system's features;
- Small cost – models should be significantly cheaper to construct and analyze than modeled system.

Analysis of the proposed criteria suggests that, in general, models should be easy to understand for software developers, integrators and maintainers, and on the other hand, they should be precise enough to allow for automatic transformations by the tools. However, it is not clear how these criteria could be satisfied. Separating impacts of visual modeling language, model and tools can be helpful in the analysis and allow for higher level of precision and more practical application. Not much work on systematic approaches has been done. The only exception was an attempt to distinguish between goals and means in conceptual

modeling and to define syntactic, semantic and pragmatic quality (Lindland et al. 1994).

Many factors have impact on the models. The process of creating them, education and experience of designers, their purpose of use, and developer's subjective decisions are just a few examples. However, models quality depends, first of all, on the quality of visual modelling languages, which define syntax of correct models, their semantics and notation. In the area of VML evaluation, several methods of evaluation or just evaluations has been described in the literature, e.g. a method based on cognitive dimensions (Cox, 2000), comparative studies (De Champeaux et. al. 1992; Gu et.al. 2002; Kutar et al. 2002), comparing to ontology of systems (Opdahl et al. 2002) or evaluation based on a set of model quality criteria for business process modelling notation (Hommes et.al. 2003).

Several approaches to checking model quality has been proposed, e.g. checklists (Rational Unified Process, 2001), heuristics (Grotehen), object-oriented metrics with thresholds (Lorenz et al. 1994). Usually when applying them one finds out they are not complete and one misses a kind of framework for checking completeness. The problem is one does not know according to which quality criteria evaluations are made. Another problem in current approaches is that it is difficult to manage scope of evaluation when scope of models in the documentation or type of system under development changes.

The level of understanding of the quality criteria for models and usefulness of the proposed methods of evaluation is not satisfactory nowadays and the progress in this area is expected. But even assuming that universal lists of the quality criteria for models are possible to be defined, it is very likely that concrete evaluation will require only limited scope of them. Therefore, instead of attempting to define yet another "final set of quality criteria", we want to present a kind of framework for dealing with these issues.

CONFIGURATION OF METHODS OF EVALUATION BASED ON QUALITY CRITERIA

The objective of our research was to provide a practical solution for increasing quality in the modeling technology. We assume that in order to achieve it one can manipulate on models or visual modeling languages. In this section we identify the requirements for the methods and their application area and then we describe a methodology for designing methods of evaluation.

With the current trend in the area of software processes to provide a framework with best practices, e.g. IBM Rational Unified Process, which could be customized and configurated for a given type of projects depending on its characteristics, particular models, modeling and model checking activities are just a result of decisions made by process engineer. This observation leads to formulation of the first requirement for the methods of evaluation. They should be configurable for artifacts under evaluation.

The practice of software project shows that most projects are performed in the tough market conditions, they must meet limitation of resources and increasing quality requirements. On the other hand, it is known that quality assessment activities are essential, but there is an optimal point on the curve describing cost-to-effect. This perspective leads to the next requirements for the methods of evaluation. They should be as effective and efficient as possible. Since more advanced methods require more time to be performed, the methods of evaluation should be additionally manageable with respect to the scope and time of evaluation.

Furthermore, process engineer should be delivered with a solid knowledge about the consequences of using a given method of evaluation. Such knowledge would enable making rational decisions instead of using a coincidental set of checkpoints.

The above requirements can be fulfilled when delivering a methodology for designing methods

of evaluation customizable for selected objects under evaluation which check out these objects against desired quality criteria. The methodology can be summarized operationally in the following steps:

1. Define objectives for your method of evaluation
2. Identify means you can manipulate – objects under evaluation
3. Define quality criteria
4. Make configuration of the method of evaluation.

This methodology applied to design of the method of evaluation for models during the software project can be used by software process engineers once during the configuration of the process for a given type of projects or when the changes to this process are introduced. Then, several evaluations of models can be made by members of the quality assurance group in several projects of this type. The methodology applied for designing methods of evaluation for visual modeling languages can be used by the members of the standard committees as well as language engineers working on visual modeling languages. The application in the area of support for improvement of the standard modeling languages is not a common activity but it has a great impact on usefulness of technology used by many users. With the increasing popularity of domain specific languages, the design of the modeling languages and methods of their evaluation becomes a more common activity.

Definition of the Objectives and Identification of Objects Under Evaluation

The first important thing when designing a method of evaluation is to define objectives. It requires making realistic decisions about expected results and elements to be manipulated. Hardly ever the process engineer has the freedom to make configuration of the process from scratch without any limitations. Without this specific point of reference, hardly any evaluation can be successful. The distinction between quality of language and quality of models increases precision in evaluation. It is useful to understand a language as an expression space and a model as a kind of expression. Thus, examples of objectives for models include:

- Find defects in the design diagrams,
- Analyze fit to the purpose of the documented system,
- Support domain expert in validation of analysis documents,
- Evaluate understandability of documentation for a project,
- Predict functional quality of a system on the basis of its documentation.

Depending on the expected results different methods would be useful.

As examples of objectives for visual modeling languages one can state:

- Evaluate application of a subset of UML for a given type of projects with respect to the fit to needs,
- Evaluate usability of UML diagrams,
- Compare expressiveness of several propositions for a domain specific language,
- Compare maturity of modeling technology using BPMN and business use case modeling including tool support and fit to software development process.

It is important to have realistic expectation what can be achieved by manipulating VML, models and tools. For example, in order to achieve "executable diagrams" one needs a VML which allows to make executable specifications, then models which are precise enough to deliver data for transformations, and finally the tools which make

the transformations to the executable form. When attempting to assess correctness with respect to syntax it is useful to understand that VML defines correct models, then one can check models against the syntax, and tools support checking correctness against the syntax. They can disable incorrect constructions but additional check is needed in order to assess lack of incomplete work-in-progress ones. When evaluating expressiveness, VML defines what is possible to express and it can be evaluated against the fit of the expression space to the domain or purpose of modeling; models are kinds of expression within VML and they can be evaluated against how they use possibilities of VML to represent a system, and the role of tools is neutral – they just facilitate making models. When evaluating inspectability, VML can be checked against ease of defining criteria for inspection and relating them the model elements, the most important for models is understandability for inspectors and tools can be evaluated whether they support making inspections. Depending on the quality aspect the role of VML, models and tools changes.

Quality Criteria

Once decided about objectives and objects under evaluation the next step is to define quality criteria. It is difficult to define an orthogonal and complete set of quality criteria. The solution depends on several circumstances as well as internal aspects of VMLs. In order to understand better impact of quality criteria for models we use the following linguistic inspirations:

- Distinction into content and expression areas - which enables to predict consequences of several types of defects;
- Model of syntax, semantics and notation - and its application to evaluation of several aspects of VML; and
- Pragmatics – a promising approach which allows to define context of model usage and make more precise context-dependent evaluation.

For describing the quality model one can use, for example, McCall's framework or Goal Question Metric (GQM) approach.

Content and Expression Areas

Distinction between content and expression areas was inspired by the work of Hjelmslev (semiotician and linguist living in 1899-1965). In his perception of a language, a sign is a function between two forms, a content form and an expression form and every sign function is manifested by two substances: the content substance and the expression substance. The content substance is the psychological and conceptual manifestation of the sign, whereas the expression substance is the material substance wherein a sign is manifested. In simple words, it is a distinction between what one communicates and the means one uses to do it, e.g. sound, text, pictures, sign language, gestures, etc.

The implication for model quality evaluation is we can evaluate separtely content of models from their expression form. It allows to split apart these characteristics which have impact on solution from those which are only concerned with understanding. In the first group there are: completeness, correctness with respect to goals of the system etc. While the second group includes understandability, precision, adequate symbols and simplicity.

The content has direct impact on the system and these criteria can be used in order to predict quality of software. For example, the lack of completeness of use case diagram will result in the lack of functions in the system, the consequence of incorrect class diagram will be incorrect structure of the system, incorrect dynamic models will propagate to incorrect interactions between the user and the system. The expression criteria are indirectly concerned with the system quality, but

they indicate for likely problems with understanding by their users and resulting defects.

Syntax, Semantics and Notation

The distinction between syntax, semantics, notation and pragmatics is inspired by the work of semioticians such as Morris and Peirce and chapters of contemporary books on linguistics which study separately syntax (grammar) when dealing with relations between signs, semantics concerned with relations between signs and reality, different forms of expressions and their rules (speech, writing, gestures) and pragmatics—dependence of communication on interpreters. The distinction between syntax, semantics and notation is successfully applied in formulation of UML description. An attempt to use distinction between syntax, semantics and pragmatics for understanding model quality has already been proposed (Lindland et al. 1994). However, this distinction has much greater potential for performing more precise methods of evaluation from the perspective of a given area as well as the perspective of relationships between given two areas.

Pragmatics

Pragmatics is the study of language which focuses attention on the users and the context of language use rather than on reference, truth, or grammar' (The Oxford Companion to Philosophy). It studies the use of language in context, and the context-dependence of various aspects of linguistic interpretation. Context is the situation in which language is used and it includes extralinguistic factors: social, psychological and environmental ones. One of the topics of debate in linguistics is the semantics-pragmatics distinction (Bach). Semantics concerns context-independent meaning (the relation of signs to objects) and pragmatics deals with context-dependent meaning (the relation of signs to their interpreters). This distinction is best reflected in the following pairs of expressions: sentence vs. utterance; meaning vs. use; context-invariant vs. context-sensitive meaning; linguistic vs. speaker's meaning; literal vs. nonliteral use; or saying vs. implying.

Pragmatics in linguistics allows to deal with the context of use. It enables understanding aspects dependent on people who take part in conversations and their situations. It could play similar role in research on modeling in software engineering. VMLs also have several kinds of users with different social, psychological and environmental factors. The perception of models might differ depending on expectations related to the roles they play in software development activities. Application of pragmatics in practice would require definition and description of pragmatics profiles and evaluation of VML in the context of these pragmatics profiles.

Design of the Method of Evaluation

The input for the method of evaluation are objectives, quality criteria and objects under evaluation which can be changed depending on the results of evaluation. The results depend on the objectives. They include: lists of defects, metrics, suggestions of improvement, comparison results etc. The essence of the design of the method of evaluation is combination of the all relevant quality criteria with objects under evaluation. This can be as simple as intersection of criteria and objects under evaluation or as difficult as using several basic methods supporting the quality improvement and integrating them.

In our opinion, the method of evaluation is basically a support for human information processing. Although some tools automatically collect diagram metrics or automatically discover some kinds of low-level defects, still majority of model evaluation tasks and almost all VML evaluation tasks must be performed by professionals.

Thus, it is useful to have a look at a cognitive inspiration (without going into details of cognitive modeling.) One can notice that within

an evaluation without any support two different types of information processing processes are performed:

- An organizational process – which is concerned with issues 'What shall I do next? What should be checked? How to do it? Is it enough?'
- A checking process – which performs several basic checks according to the organizational process.

These processes take place concurrently and can be informal, implicit and unordered. They might be more or less conscious and casual. Main problems and most of 'awaiting' are associated with the organizational process. The role of the method of evaluation is to deliver a pattern for the organizational process, and, thus, to make it more explicit, conscious, ordered and repeatable. As the most waste of time is concerned with this process, the method should result in improvement of the efficiency of evaluation.

Furthermore, modeling concepts and quality concepts are usually placed in different conceptual models in our minds. The role of the methods of evaluation which combine model elements with quality criteria is to drive evaluator's attention step by step to facilitate checking all important aspects. Any of these combinations is less prone for forgetting when there are lots of them or evaluator is working in hurry or has another motivation to finish the evaluation soon. Thus, the method supports effectiveness by forcing complete check.

Additionally, some strategies are useful when designing methods of evaluation, e.g. 'most important things first' which takes into consideration that people get tired during evaluation and their productivity can decrease, or the idea of 'building evaluators' knowledge' and thus checking related aspects together in time. A good practice is to design a space for 'other comments' just in case of finding by evaluators defects which are not covered by the method or to allow them to express their comments on the method.

One more important group of aspects to be taken into account when designing methods of evaluation are human factors: evaluators, their characteristics, their knowledge and their context of work. The differences might depend on culture or language. The evaluators might prefer e.g. command style or question style, long questionnaires with very simple questions or shorter questionnaires with more open and complex questions, discussions with authors or just formal reports from evaluation.

An interesting issue related to the design of the methods of evaluation is whether it is possible to automatically generate appropriate checklists. As tools can be designed to support any task, it is useful to pose additional questions about the quality of the generated checklists and effectiveness of generation, i.e. comparison of the effort required for the development of such tools and inserting necessary data with the effort required for re-designing the checklists when context of application or requirements change. Large benefits can be achieved with low-cost manually-made methods of evaluation. The tools can add the value of easy generation of multiple variants of checklists and allow for relating results of evaluation with the items of the method and processing them. However, one should be skeptical to automatically generated checklists and review them carefully for accuracy and style.

The main idea of the method to support organizational process during evaluator's work and to drive their attention seems to be universal, however more research on strategies and evaluator's preferences would be useful.

CASE STUDIES

In order to demonstrate the practical application of the methodology of designing methods of evaluation, we present three case studies with design of

diverse methods of evaluation. Two of them are related to models, but they have different objectives. The first method is a simple manageable checklist for the purpose of finding defects and the second uses every information on the models in order to deliver quality predictions (Bobkowska, 2001). The third case study summarizes a method for evaluating visual modeling languages from a cognitive perspective (Bobkowska, 2005).

Quality Criteria-Based Checklist

The objective of this method design was to provide a simple checklist for verification of analysis models to be applied in teaching classes of systems modeling and analysis. It was required that the method fits well to the diagrams under evaluation and quality criteria for models should be explicitly stated. Additionally, the method should be efficient and easy to modify.

The object under evaluation was documentation of the analysis, which consisted of the vision of the system (goals, scope, context) in the textual form, use case diagram for presentation of the system's functionality, class diagram and sequence diagrams for all use cases. Apart from the diagrams, non-functional requirements were specified in the textual form.

The quality criteria reflected the distinction between content and expression areas. The content group included: completeness, correctness, consistency and fit to the vision of the system. The expression group included: understandability of diagrams, their elegance, precision, simplicity and adequate level of abstraction.

While designing the method we have used additionally two 'strategies' which allowed for effective and efficient evaluation: 'direct users to check most important aspects first' and 'order questions around the diagrams not the quality criteria'. This resulted in a checklist to check content of the use case diagram, then content of the class diagram and sequence diagrams. Later consistency between all diagrams and fit to the vision were evaluated, and finally - expression criteria for all diagrams. The checklist ended with some space for summary. Each subsection related to checking a given diagram with a selected quality criterion included direct instructions for users informing them what they should do.

For example, section '1. Content of use case diagram' consists of two subsections '1.1 Completeness of use case diagram' with instruction to identify missing actors, missing use cases and missing relationships and subsection '1.2 Correctness of use case diagram' with instruction to identify incorrect actors, use cases and relationships, i.e. spurious, these with wrong scope, wrong labels, wrong directions of arrows in relationships. Section '6. Expression of use case diagram' consists of the following subsections '6.1. Precision and understandability of labels and descriptions' with instruction to identify unclear, vague and difficult to understand labels and descriptions, '6.2. Elegance of icon placement on the diagram' with instruction to find defects related to placements and aesthetics of the diagram, '6.3. Simplicity' with instruction to identify unnecessarily complex descriptions and constructions on the diagram and finally subsection '6.4 Adequate level of abstraction' with instruction to identify mismatch of elements in respect to level of abstraction, e.g. some elements too general or others too detailed.

This checklist satisfies the requirements formulated when stating objectives. It is simple, it fits exactly for the diagrams under evaluation and it explicitly indicates for the criteria of evaluation. It is as efficient as possible and easy to modify when documentation or quality criteria change. We collected opinions of users of this checklist who have also used fragments of general purpose RUP checklist for these diagrams. The users said that clear instructions in context of these subsections facilitate finding defects and give a good understanding according to which criteria evaluation is made. The checklist was leading them through the review and without it they

wouldn't find so many defects. They claimed it suited better to the diagrams under evaluation then the RUP checklist.

Software Quality Prediction

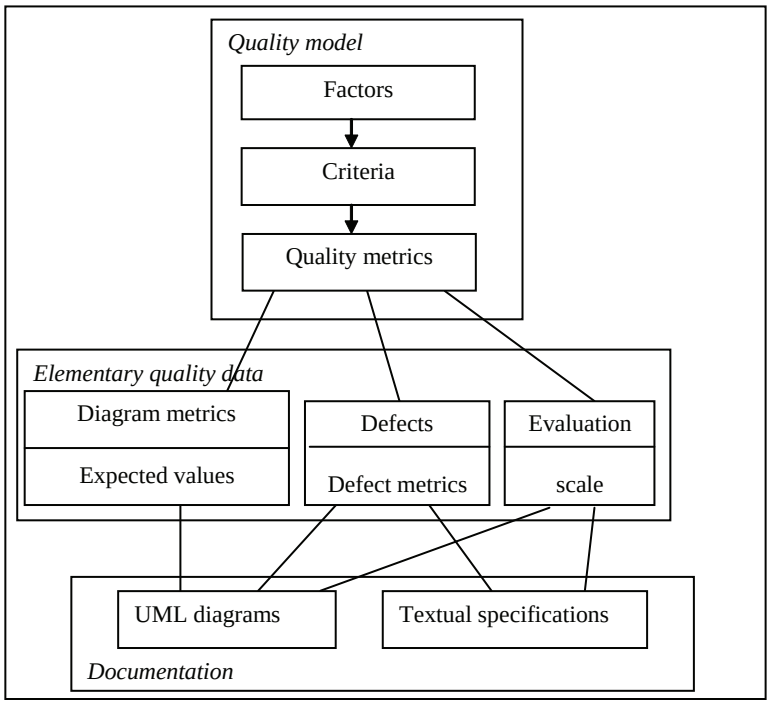
The objective of this method design was to use every information on the models indicating for quality of the final software to be developed and use it in order to make software quality predictions in early phases of software development. Such predictions should allow managers to control the development process with quantitative measures of software product (together with measures of software process taken from other sources).

The objects under evaluation were documents of software analysis and design. The scope of analysis was similar to one described in the previous section, and the documentation of design consisted of systems design which consisted of subsystem diagram and interactions between subsystems, decisions about environment, boundary conditions, style and trade-offs. Other documents contained user interface design, database design and subsystems design.

Since the objective was to predict quality factors of system under development, the meaningful attributes were functionality, maintainability and usability. They needed to be related to quality criteria of models such as completeness, correctness, precision, adequate structure, flexibility to probable changes, adequacy of designed algorithms etc. Additionally, the requirement of quantitative measurement leaded to introduction of quality metrics for both quality factors and quality criteria in scale 0.1.

The schema of the relationships between items of the method for software quality prediction is shown in Fig. 1. The design integrates the quality model (made with the use of McCalls framework

Figure 1. Elementary quality data in context of the quality model and documentation with UML (©2007 Anna Bobkowska. Used with permission)



with its terms of quality factors, criteria and metrics) with UML model elements and related textual specifications, e.g. these which describe vision of the system or system design decisions.

The link between them consists of elementary quality data, which can be diagram metrics and their expected values, model evaluation in a scale and the lists of defects together with their metrics. Diagram metrics cover the idea of measurement of the UML diagrams and use of this information to reason about quality by comparing the results of measurements with expected values, which can be derived from historical data about similar projects. Diagram metrics are also the basis for local size calculation which can be used for defect metrics calculation. The diagram metrics are example of optimal features, and expected values indicate the right range. Evaluations with comments are concerned with the criteria that are difficult to measure objectively. Evaluations are numbers that represent subjective feelings about some aspects of the work, e.g. ease of understanding, aesthetics, precision, etc. They can be given in the scale and described as comments. They represent positive features. Defect collection according to the defect classification is an instantiation of the negative features in the model. These defects can be then counted and combined with the diagram size metrics.

Two steps are performed during the quality prediction:

- Elementary quality data collection which allows to gather elementary quality data on the basis of documentation;
- Reasoning about quality, which allows to transform elementary quality data into quality metrics, which can be made manually by quality expert or can be calculated according to formulas with parameters which represent the impact of given metrics or evaluations.

This method requires more work comparing to the previous case study. However, it satisfies its goal of delivering quantitative predictions of the software quality in early phases of software development. It allows to see consequences of defects in the documentation for the software quality and supports the control of the process of software development. The problem of software quality prediction is quite complicated and when designing a solution one meets the questions of missing data and thus uncertainty of predictions as well as the problem of subjectivity of evaluations and reasoning. However, with questionnaires supporting elementary quality data collection and reasoning, even students of experimental group have performed predictions with satisfying results.

Evaluating VMLs from a Cognitive Perspective

The objective of this method design was to propose a tool for visual modeling language (VML) evaluation from a cognitive perspective and then evaluate with this method use case diagram. It is believed that cognitive fit of the technology to its users can increase efficiency, decrease cost, improve software quality and allow for easier learning, use and maintenance.

Since the methodology is configurable the object under evaluation can be any visual modeling language or any of its diagrams. In this case study we present customization for use case diagram, thus we operate are at meta-model level and objects under evaluation are UML model elements.

The quality criteria for this methodology were defined by a set of cognitive dimensions (Blackwell, Green, 2000):

- Viscosity - resistance to change,
- Visibility - ability to view components easily,
- Premature commitment - constraints on the order of doing things,

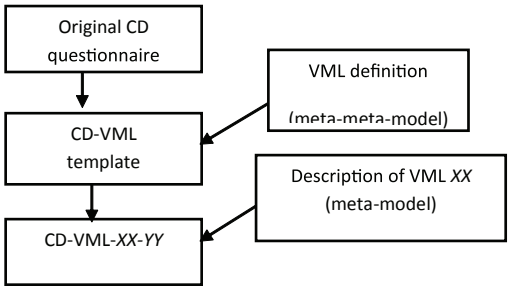
- Hidden dependencies - important links between entities are not visible,
- Role-expressiveness - the purpose of an entity is readily inferred,
- Error-proneness - the notation invites mistakes and the system gives little protection,
- Abstraction - types and availability of abstraction mechanisms,
- Secondary notation - extra information in means other than formal syntax,
- Closeness of mapping - closeness of representation to domain,
- Consistency - similar semantics are expressed in similar syntactic forms,
- Diffuseness - verbosity of language,
- Hard mental operations - high demand on cognitive resources,
- Provisionality - degree of commitment to actions or marks,
- Progressive evaluation - work-to-date can be checked at any time,

A schema of the approach to the design of the method is presented in Fig. 2. CD-VML template is a product of customization of the original cognitive dimensions (CD) questionnaire for visual modeling languages. It required some modification to increase precision of questions and their fit to visual modeling language terminology. Then, the next step in design is customization for the method of evaluation for a specific visual modeling language or one of its diagrams. We have designed a CD-VML-UC questionnaire - a product of the CD-VML methodology for the use case (UC) model in the default context of use of creating models, their usage and change.

In order to give an example, we present questions from the section of the questionnaire related to error-proneness. It starts with questions to find common mistakes whilst modeling and using diagrams: 'Whilst modeling, what kinds of mistakes are particularly common or easy to make? Which kinds of mistakes you must be careful not to make? Whilst using diagrams, which misunderstandings are very likely to happen?' They were followed by a table with listing of all model elements and notation elements and space for problem descriptions. The table was headed with the questions: 'Which model elements or constructions or notation element or visual mechanisms are these mistakes concerned with? What are the problems?' Below the table there was a space for explanations, examples, comments and suggestions for improvement of the VML or suggestions for special features of the CASE tool.

This method was verified in an experiment with students as participants. Students confirmed simplicity and usefulness of use cases, but they also discovered a large number of problems, gave reasonable explanations to them and quite often made suggestions for improvements. Their discoveries covered all the problems reported in the

Figure 2. Schema of the CD-VML methodology (©2007 Anna Bobkowska. Used with permission)



related work. The level of detail of the individual answers was satisfactory: problems usually were described with details and examples and even simple diagrams for illustration of the problem were added. In the section of comments they stated the following strengths of the methodology: the precision of questions, large area of covered issues and ease of use. Usefulness of the method was evaluated as high and results were considered as important for use case model improvement.

FUTURE TRENDS

In our opinion, the following areas need to be integrated into the research on quality of software models:

- Proper configuration of modeling tasks and artefacts in the context of software project;
- Integration of reuse technology with modeling technology;
- Need for methods which enable evaluation of the fit between the problem and modeling technology;
- Usability of the modeling technology;
- Visual modeling language engineering.

There is a need for knowledge which would enable proper configuration of the modeling artefacts and modeling tasks within the software project depending on the type of project. Modeling takes time and often does not result in increase of code (except from the situation where the full code is generated by tools). Thus, the suggestion is that the use of models is efficient and beneficial only to certain limits. Furthermore, unlike suggestions in pure modeling solutions, in realistic projects still a lot of documentation is made in the textual form. It is worth to mention that quality of models and thus the quality of the system which is represented by these models depends on whether analysts and developers have enough time for making them. It depends also on the proper scope of modelling. Understanding of this impact requires more research and should result in guidelines on the scope of application of models in the software project depending on the project characteristics. Once the development process is customised, the proposed methodology of designing methods of model evaluation can be used to create proper methods of evaluation.

Integration of the reuse technology with the modeling technology is the next of the challenges. We have a lot of evidence about benefits of reuse for both efficiency and quality of software. However, there are some risks as well. The most important reuse solution related to modeling are patterns and components. The related questions include: How to use patterns effectively for the purpose of software development? How to study their impact for software quality? How the fit between a problem and the applied patterns matters? How components could be incorporated into models? How to analyse their impact for quality of software? The proposed method can facilitate approach to evaluation, but several new factors must be defined e.g. fit of the reuse method to the solution of the problem.

There is a need for methods which enable evaluation of the fit between a problem and given technical modeling solution. It is not a surprise that a problem which can be difficult in one technology can be easy to solve in another. Expert's knowledge is necessary to evaluate modeling solutions against the problem at hand. Examples of decisions include: whether to use UML or a domain specific language, object-oriented or aspect-oriented solutions; how to choose between several technical spaces? It is worth to remember that this perspective sets the space of achievable effects.

Much more work should be done on the usability side of modeling methods. Most decisions so far are technology-driven and result in difficult to use methods and tools. Focus on users of the modeling technology could surely result in better

fit of the technology to support several roles of software development process and their activities. This area of research can use pragmatics by the means of identifying pragmatics profiles and their tasks and then evaluating how they are supported by modelling technology.

And finally, we would like to support the research in the discipline of visual modelling language engineering. The technology changes, modeling paradigms are passing by, but the activity of modeling is useful anyway. The role of visual modeling language engineering would be to capture the knowledge about modeling which is technology invariant. It could collect universal and consistent knowledge about models in the areas including:

- understanding the role of models;
- quality of models;
- guidelines for creating visual modeling languages;
- quality of visual modeling languages;
- guidelines for use of modeling technology in the software project.

The proposed method could be a part of visual modeling language engineering.

CONCLUSION

The objective of this chapter was to present an approach to dealing with model quality in a way which integrates quality criteria and methods of evaluation. The methodology for designing methods of evaluation customized for selected objectives and objects under evaluation which check out these objects against desired quality criteria fulfills the requirements which were stated before it. It is explicitly configurable to different objectives and objects under evaluation and thus it fits to configurable software development process in which particular models, modeling and model checking activities are results of decisions made by process engineer. With integrating quality criteria and methods of evaluation it is easy to manage scope of inspection and enable better fit to the objects under evaluation. Inspirations from linguistic theories provide a framework for understanding impact of given criteria for the quality of software under development and software development process. The proposed methodology is flexible and universal. The case studies have delivered an evidence of feasibility of the methodology and its coverage for a diversity of cases as well as usefulness of the results achieved with the use of designed methods of evaluation.

REFERENCES

Bach, K. The Semantics-Pragmatics Distinction: What It Is and Why It Matters, Retrieved in June 2005, from http://userwww.sfsu.edu/~kbach/sempprag.html

Blackwell, A.F. & Green, T.R.G. (2000) A Cognitive Dimensions questionnaire optimised for users. In: *Proceedings of the Twelth Annual Meeting of the Psychology of Programming Interest Group* (pp.137-152).

Bobkowska, A.E. (2005) A methodology of Visual Modeling Language Evaluation, In: *Proceedings of SOFSEM 2005*, LNCS 3381 (pp. 72-81).

Bobkowska, A.E. (2001) *Software Quality Prediction with UML*, Unpublished doctoral dissertation, Gdansk University of Technology.

Booch, G. (1994) *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings.

Cox, K. (2000) Cognitive Dimensions of use cases: feedback from a student questionnaire. In: *Proceedings of the Twelth Annual Meeting of the Psychology of Programming Interest Group.*

De Champeaux, D.& Faure, P. (1992) A comparative study of object-oriented analysis methods.

In: Journal of Object-Oriented Programming, 1(5) 21-33.

Firesmith, D., Henderson-Sellers, B., Graham, I.& Page-Jones, M. (1996) OPEN Modeling Language (OML). Reference Manual.

Gu A., Henderson-Sellers B.& Lowe D. (2002) Web Modeling Languages: The Gap Between Requirements And Current Exemplars. In *Proceedings Of The Eighth Australian World Wide Web Conference*.

Grotehen, T. & Dittrich, K.R. *The MeTHOOD Approach: Measures, Transformation Rules, and Heuristics for Object-Oriented Design*, Technical Report., retrieved in October 2003 from http://www.ifi.unizh.ch/groups/dbtg/MeTHOOD/index.html

Hommes, B.J.& van Reijswoud, V. (2000) Assessing the Quality of Business Process Modeling Techniques, In *Proceedings of the 33rd Hawaii International Conference on System Sciences.*

Hong, S. & Goor, G. (1993) A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In *Proceedings of the 26th International Hawaii International Conference on System Sciences.*

Kutar, M., Britton, C.& Barker, T. A. (2002) Comparison of Empirical Study and Cognitive Dimensions Analysis in the Evaluation of UML Diagrams. In *Proceedings of the Fourteenth Annual Meeting of the Psychology of Programming Interest Group.*

Lorenz, M. & Kidd, J. (1994) *Object-oriented Software Metrics. A Practical Guide.*, Prentice Hall.

Lindland, O.I., Sindre, G. & Sølvberg, A. (1994) Understanding Quality in Conceptual Modeling, *IEEE Software.*

Martin, J. (1993) *Principles of object-oriented analysis and design*, Prentice Hall

McGregor, J.D. (1998), The fifty-foot look at the analysis and design models, *Journal of Object-Oriented Programming* 11(4) 10-15.

Opdahl, A.L.& Henderson-Sellers, B. (2002) Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. *Journal of Software and System Modeling,* 1.

Rumbaugh, J. (1999) Notation Notes: Principles for choosing notation, In *Journal of Object-Oriented Programming*, 12, 4,.

Selic, B. (2003) The Pragmatics of Model-Driven Development, *IEEE Software* 9, 19-25.

ADDITIONAL READING

Basili, V., Green, S., Laitenberger O., Shull F., Sorumgaard S., & Zelkowitz M, (1996) The Empirical Investigation of Perspective-Based Reading, *Empirical Software Engineering: An International Journal*, vol. 1, 2 (pp.133-164) Another approach to reviews which is based on perspectives of people involved in the process.

Fenton N.E. & Pfleeger S.L. (1998), *Software Metrics: A Rigorous and Practical Approach, Revised*, Course Technology; 2 edition A book for quick and solid introduction to the role of metrics in software engineering.

Gilb T. & Graham D. (1993), *Software Inspection*. Workingham: Addison-Wesley.

An introduction to inspections and reviews in software engineering.

Unhelkar B. (2005) *Verification and Validation for Quality of UML 2.0 Models*, John Wiley & Sons Inc. A book describing checklists for syntactical correctness, semantics and aesthetics of models.

This work was previously published in Model-Driven Software Development: Integrating Quality Assurance, edited by J. Rech and C. Bunse, pp. 78-94, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 7.7
Software Security Engineering:
Towards Unifying Software Engineering and Security Engineering

Mohammad Zulkernine
Queens University, Canada

Sheikh I. Ahamed
Marquette University, USA

ABSTRACT

The rapid development and expansion of network based applications have changed the computing world in the last decade. However, this overwhelming success has an Achilles' heel: almost every software controlled system faces threats from potential adversaries both from internal and external users of the highly connected computing systems. These software systems must be engineered with reliable protection mechanisms, while still delivering the expected value of the software to their customers within the budgeted time and cost. The principal obstacle in achieving the above two different but interdependent objectives is that current software engineering processes do not provide enough support for the software developers to achieve security goals. In this chapter, we reemphasize the principal objectives of both software engineering and security engineering, and strive to identify the major steps of a software security engineering process that will be useful for building secure software systems. Both software engineering and security engineering are ever evolving disciplines, and software security engineering is still in its infancy. This chapter proposes a unification of the process models of software engineering and security engineering in order to improve the steps of the software life cycle that would better address the underlying objectives of both engineering processes. This unification will facilitate the incorporation of the advancement of the features of one engineering process into the other. The chapter also provides

a brief overview and survey of the current state of the art of software engineering and security engineering with respect to computer systems.

INTRODUCTION

With the proliferation of connectivity of computer systems in the applications where the quality of service depends on data confidentiality, data integrity, and protection against denial of service attack, the need for secure networks is evident. In these applications, the consequences of a security breach may range from extensive financial losses to dangers to human life. Due to heavy dependence of computer network based applications on various software and software controlled systems, software security has become an essential issue. Almost every software controlled system faces potential threats from system users, both insiders and outsiders. It is well accepted that "the root of most security problems is software that fails in unexpected ways when under attack" (McGraw, 2002). Therefore, software systems must be engineered with reliable protection mechanisms against potential attacks while still providing the expected quality of service to their customers within the budgeted time and cost. Software should be designed with the objective not only of implementing the quality functionalities required for their users but also of combating potential and unexpected threats. The principal obstacle in achieving the above two different but interdependent objectives is that current software engineering processes do not provide enough support for the software developers to achieve security goals.

Some of the principal software engineering objectives are usability, performance, timely completion, reliability, and flexibility in software applications (Finkelstein & Kramer, 2000; Pressman, 2001; IEEE, 1999). On the other hand, some of the major objectives of security engineering are customized access control and authentication based on the privilege levels of users, traceability and detection, accountability, non-repudiation, privacy, confidentiality, and integrity (Pfleeger & Pfleeger, 2003; Viega & McGraw, 2001). Having stated that, software security engineering objectives are to design a software system that meets both security objectives and application objectives. However, software security engineering is still considered a difficult task due to inherent difficulties associated with the addressing of the security issues in the core development and maintenance of software systems. Both software engineering and security engineering are ever evolving disciplines and software security engineering is still in its infancy. A precise and well-accepted understanding of software security engineering does not yet exist (ISSEA, 2003; Wolf, 2004).

The principal objectives of software security engineering need to be reinvestigated, and a methodology is required that can be employed for building secure software systems. Software security engineering is a practice to address software security issues in a systematic manner. We believe that the security issues must be addressed in all the stages of software system development and maintenance life cycle such as requirements specifications, design, testing, operation, and maintenance to provide secure software systems. This chapter identifies some possible ways to adapt readily the current practices of security engineering into a software engineering process model. In other words, this chapter suggests a unification of the process models of software engineering and security engineering. The unification is desirable between the two processes by removing the unnecessary differences in their corresponding steps that obscure the fundamental principles of the development and maintenance of secure software systems. This unification will help system designers to employ the techniques and tools of software engineering and security

engineering in a complementary fashion. It will also help to incorporate the advancement of the features of one engineering process into the other. In our attempt, we suggest to incorporate new, more intuitive but well-defined steps in the software security engineering process in order to build secure software systems. It may be noted here that a number of studies have already proved that "end-to-end integration" of security aspects in the software life cycle have significant benefits with respect to the quality of the end software products (Devanbu & Stubblebine, 2000; Viega & McGraw, 2001).

In this chapter, we describe a software security engineering process, which is primarily based on the famous waterfall software development process model (Royce, 1987). The major steps of the proposed software security engineering process model are described in detail. The rationale for choosing the waterfall model as a basis for the description is that it contains most of the fundamental steps present in all other software process models that currently exist in the literature. Therefore, our idea of a software security engineering process which incorporates security issues in all the steps of software life cycle will work fine in other software process models. For example, the incremental model is widely used in industry with a view to reducing the delay in delivering software products. It consists of several increments or builds, where each increment follows the waterfall model. Similarly, in an incremental model based software security engineering process, each increment may adopt the steps described in this chapter.

The rest of the chapter is organized as follows. Section 2 provides a brief overview of software engineering and security engineering, and presents a brief synopsis of the current state of the art of other related efforts towards software security engineering. Section 3 provides the details of the proposed software security engineering process model. The chapter concludes in Section 4 with a summary of current research and future recommendations.

BACKGROUND

A Glimpse of Software Engineering

In the IEEE Standard Glossary of Software Engineering Terminology (IEEE Std. 610.12-1990), software engineering is defined as follows (IEEE, 1999): "(1) Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)". It primarily focuses on applying an engineering approach to the design, implementation and maintenance of software systems. Software engineering is an engineering discipline, which is concerned with all aspects of software production, from the early stages of system specification to maintenance of the system after it has been deployed for use (Finkelstein & Kramer, 2000). Software engineering is a subset of system engineering, which focuses on overall system (both software and hardware), while software engineering only focuses on the software aspects of the system development.

There are different software development life cycle models such as the waterfall, incremental, spiral, and rapid prototyping (Pressman, 2001). Among these models, the waterfall model (Royce, 1987) is the most widely used and discussed. Most models consist of steps for requirement gathering and analysis, software architecture, implementation, and testing. Stakeholders involved in a software engineering process influence software engineers to follow certain process models to address and solve different issues of the software in an effective way. The stakeholders include analysts, designers, programmers, testers, maintenance personnel, project managers, and end users or customers. The project manager

leads the whole software development group of analysts, designers, programmers, testers, and maintenance personnel. An analyst collects and analyses the requirements, and a designer forms the architecture with a view to leading the requirements towards the implementation. A programmer implements the architecture or design provided by the designer, and testers test the software for ensuring high quality end product. Maintenance personnel deploy and maintain the software so that the operators can use the software effectively.

Software engineering refers to techniques intended for making software development more orderly, systematic, and organized. It also helps to reduce time and cost for developing software in a scientific way. In summary, low cost, high quality, and rapid software development are the key goals of software engineering. The rapid development and expansion of the software are possible with effective use of a software engineering process; however, security issues have not yet been incorporated or addressed in the software development processes.

A Glimpse of Security Engineering

The IFIP (International Federation for Information Processing) working group WG10.4 on dependable computing and fault tolerance considers security as one of the attributes of software 'dependability', where dependability is defined as the "trustworthiness of a computer system such that reliance can be justifiably placed in the service it delivers" (Laprie, 1992). The other principal attributes of dependability are reliability, availability, and safety, while security professionals usually consider safety and reliability as attributes of security. The threats to security are identified as interception, interruption, modification, and fabrication of data (Pfleeger & Pfleeger, 2003). The threats are usually defined based on some security policy of the organizations, while the violations of any security policy are combated using different security mechanisms such as encryption, authentication, authorization, and auditing.

Security engineering is a discipline for "building systems to remain dependable in the face of malice, error, or mischance", where the discipline "emphasizes the tools, processes, and methods needed to design, implement, and test complete systems, and to adapt existing systems as their environment evolves" (Anderson, 2001). Security engineering deals with a number of issues such as intrusion protection and detection. It ensures a system to be secured from both software and hardware perspectives. System security is becoming increasingly more important due to the new trend of ubiquitous/pervasive computing that has evolved during the last few years; numerous mobile or embedded devices such as PDAs and cell phones are connected via internet, incurring enormous security vulnerabilities.

Security engineering addresses the complete lifecycle for building a secure system: requirements specification and analysis, design, development, integration, operation, maintenance and re-engineering (ISSEA, 2003). It requires expertise in inter-disciplinary areas such as computer system management and development, software and hardware security, networking and cryptography, applied psychology, organizational management, and law. Without incorporating the security engineering principles derived from the relevant cross-disciplinary areas to every level of a software development process, it is nearly impossible to deliver a secure software system in its first release.

Related Effort to Address Security Issues in Software Engineering Context

We begin this section with the following question that was asked in (Wolf, 2004): "Is security engineering really just good software engineering?" We also believe in their concluding ob-

servation that security engineering cannot be differentiated from software engineering if "the underlying mathematics and higher-level policy and human factor issues" are not considered. Our approach of software security engineering is along the line of the following research challenges outlined in (Finkelstein & Kramer, 2000; Devanbu & Stubblebine, 2000): integration of security requirements with software requirements into early development life cycle; design of adaptive software systems to handle new attacks and changing security policies of any organizations; testing security requirements of software systems; derivation of intelligent and responsive framework for operational stage monitoring and maintenance. A number of other related efforts towards incorporating security engineering principles into software development life cycles are discussed in the following paragraphs.

In (Viega & McGraw, 2001), a software risk management process is proposed for assessing, quantifying, and establishing an acceptable level of risk in an organization. The proposed process is inspired by Boehm's spiral software process model (Boehm, 1988). In this process, security aspects are considered when software requirements are derived. The most severe security risks are evaluated and strategies are planned for resolving those risks. The risks are addressed through prototyping, and validated that the risks of those security violations are eliminated. Then, the solution to the security risks is applied for the next version of the software requirements, design, or source code. Finally, the planning for the next phase is outlined.

In (Flechals, Sasse, & Hailes, 2003), a process for usable and secure software systems called 'Appropriate and Effective Guidance in Information Security (AEGIS)' is proposed. The AEGIS involves the context in which the system is going to be used, and analyzes the potential risks to help developers handle security related issues. Similar to the risk management process proposed in (Viega & McGraw, 2001), the AEGIS is also based on Boehm's spiral software process model. It requires communication among a system's stakeholders in the risk analysis and the selection of appropriate defense mechanisms so that the developers gain a better understanding of the security and usability requirements. The AEIGS employs Unified Modeling Language (UML) (Jacobson, Booch, & Rumbaugh, 1999) to describe a way to perform "risk-usability-design" simultaneously.

Software Security Assessment Instrument (SSAI) project (Gilliam, Wolfe, & Sherif, 2003) identifies five important entities with respect to software security engineering: software security checklist (SSC), vulnerability matrix (Vmatrix), flexible modeling framework (FMF) for verification of requirements, property-based tester (PBT) for testing vulnerabilities, and security assessment tools (SATs). Similar to our objective in this chapter, the SSC "provides instrument to integrate security as a formal approach to the software life cycle".

The System Security Engineering Capability Maturity Model (SSE-CMM) (ISSEA, 2003) proposes a unified process of practicing security engineering in the context of software development. In the systems security engineering process, the security engineering task is divided into three interrelated but separate basic process areas: risk process, engineering process, and assurance process. The risk process identifies and prioritizes the risks inherent to the product under development; the engineering process interfaces with the other engineering disciplines to provide solution to the identified risks; and the assurance process improves the level of confidence of the product developer as well as the customers with respect to the proposed security solutions. The SSE-CMM describes the characteristics of five capability levels, which are inspired by the CMM used to evaluate software engineering practices.

The Unified Modeling Language (UML) (Jacobson, Booch, & Rumbaugh, 1999) has cur-

rently become the de facto standard to specify, visualize, and document models of software systems including their structure and design. More specifically, it is suitable for object oriented analysis and design. The UML defines twelve types of diagrams of three categories. Nevertheless, the UML does not provide a framework that can address the issues with respect to developing secure software system. A number of efforts are taking place in order to enrich the UML based software development processes. Some of them are as follows: UMLSec (Jürjens, 2003), Misuse and Abuse cases (Sindre & Opdahl, 2000; Dermott & Fox, 1999), Secure UML (Lodderstedt, Basin, & Jurgen, 2002; Basin, Doser, & Lodderstedt, 2003), and Security Integrated Rational Unified Process (Oikinomopoulos & Giritzails, 2004).

To address security issues in the context of software development processes, some of the elements of UML such as use case, class, sequence, state chart, package and deployment diagrams are extended (Jürjens, 2003). This UML extension for secure software development is called UMLSec. The extensions included in the UMLSec are based on using stereotypes and tag values, which enrich the existing UML diagrams with security related information.

Standard use case diagrams are often useful for eliciting functional requirements, while they are not that suitable for describing security requirements (Jacobson, Booch, & Rumbaugh, 1999). The security requirements are usually related to prohibited activities. Therefore, an extension to standard UML use case notations is proposed in (Sindre & Opdahl, 2000) in order to include unwanted use case scenarios. A use case describes a sequence of actions that represents a service to the user. On the other hand, a misuse case in the context of security requirements is defined as a sequence of actions that results in security violations for the organization or some specific stakeholder. They also define "mis-actor" as just the opposite of an actor and it initiates the misuse cases. In (Dermott & Fox, 1999), a use case based object-oriented modeling technique is used to represent significant forms of abuses that need to be prevented. However, security issues are not addressed in every phase of the software engineering process.

Secure UML (Lodderstedt, Basin, & Doser, 2002; Basin, Doser, & Lodderstedt, 2003) describes a methodology for modeling access control policies and integrating it into a model-driven software development process. This work is inspired by the idea of using UML to model Role Based Access Control (RBAC) policies. The unique feature of the Secure UML is that it also includes elements to be used in behavioral models.

A number of above extensions of the UML for the purpose of security are compared and contrasted in (Oikinomopoulos & Giritzails, 2004). We agree to their view that most of those extensions are concerned with the presentation (notational) issues of security related information rather than a methodology, which emphasizes the activities to be performed for software security engineering. In most of the above work, there is very little or no guideline about how and in what phase of the development cycle the extended UML notations will be utilized. In (Oikinomopoulos & Giritzails, 2004), they incorporate security engineering to the Rational Unified Process (RUP). The RUP divides the software development effort in phases and major workflows. They extend each workflow with additional security related activities, and identify the artifacts that should be produced. To support these activities, they also introduce the role of a "security engineer" in their methodology.

SOFTWARE SECURITY ENGINEERING

Having discussed both software engineering and security engineering, it is important to note that

"while software engineering is about ensuring that certain things happen, security is about ensuring that they don't" (Anderson, 2001). Nevertheless, following the IEEE Standard 610.12-1990 (IEEE, 1999) definition of software engineering, software security engineering can be defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of" secure software; i.e, "the application of engineering" to obtain secure software (Anderson, 2001). In the software security engineering process, the principles and specialties of other engineering disciplines with respect to software and security are integrated to achieve the "trustworthiness" in software systems. This process will help software development to proceed in a way where it will have less chance to expose itself to security related loopholes. It will endorse low cost, high quality, and a rapid software development process where the security issues will be addressed in every phase of the process. Software security engineering involves the stakeholders of software engineering while actively involving security engineers in most of its phases. Similar to both software engineering and security engineering, software security engineering incorporates expertise from various other cross-disciplinary areas such as computer system management and development, software and hardware security, computer networks, organizational management, psychology, and law.

We describe a software security engineering process, which is primarily based on the famous waterfall software development process model. The waterfall model contains most of the fundamental steps present in other software process models currently used in industries and academia. The major steps of the proposed model are as follows: system engineering, requirements specification, software design, program implementation, and system operation. The steps are presented schematically in Figure 1. While the following paragraphs provide a brief introduction to the steps, all of the steps are further detailed in the following subsections.

System Engineering. The software concerns of a system development are separated from its hardware concerns. Then both functionality issues and security issues are identified based on the users' or stakeholders' needs and in the context of other system elements such as hardware, software, people, database, documentation, and procedure.

Requirements Specification. The security requirements of a software system are specified, analyzed, and integrated along with the other functional and nonfunctional requirements of the software system. It is recommended that both security requirements and the other software requirements should be specified using the same formal language . The requirements are expressed as a set of scenarios.

Software Design. Both functionality and security issues should be considered when deciding where the application components are deployed and how they communicate with each other. A scenario based software architecture is designed based on functional scenarios from users' perspective and attack scenarios from intruders' perspective. The architecture should be compositional so that whenever a new attack is encountered, the system architecture can be easily adapted to defend against the attack.

Program Implementation. The system is implemented to meet the specified requirements, and the correctness and effectiveness of security mechanisms along with functional characteristics are verified with respect to the requirements specification. When the required functionalities are implemented, the source code is refactored (changed) to control any number of vulnerabilities against threats from attackers, while still meeting all of the functional acceptance tests.

System Operation. The software is monitored for security violation or conformance to its functional requirements based on the specification

Figure 1. A waterfall based software security engineering process model

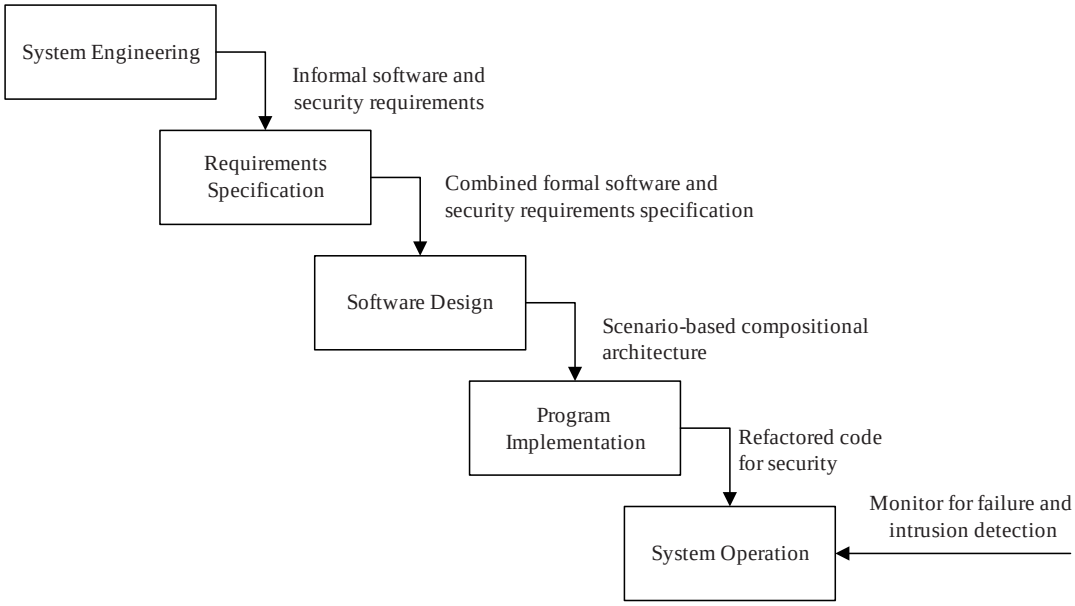
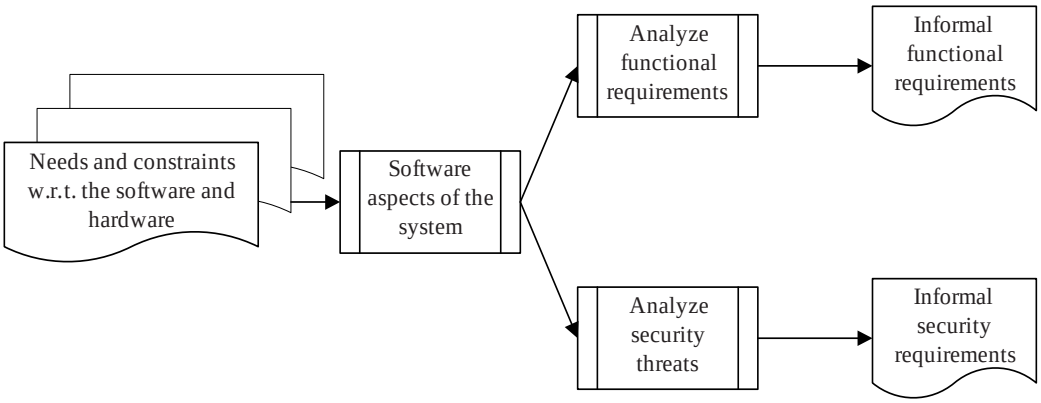


Figure 2. The system engineering phase of the software security engineering process



formalized in the requirements specification phase. In other words, the software monitor is equipped with different intrusion detection capabilities. The monitor also measures whether the operational impacts due to security vulnerabilities or residual software faults in a system are acceptable or not.

System Engineering

Software engineering deals with the software aspects of system engineering while security engineering focuses on the overall system engineering (both software and hardware). Similar to software engineering, the end product of the

software security engineering process is software. The system engineering phase of the software security engineering process identifies the software aspects of the system to be developed. Obviously, this phase equally focuses on both functionalities and security issues of the software system based on the users' or stakeholders' needs and in the context of other system elements such as hardware, software, people, database, documentation, and procedure. Both software engineers and security engineers closely work in this phase.

Similar to IEEE Standard 1220-1998 for system engineering (Thayer, 2002), this phase of the software security engineering process involves defining the system requirements that determine the needs and constraints of the software and hardware. This is accomplished through analyzing the functional requirements and all the aspects of security related threats (see Figure 2). The outcomes of this phase are the informal functional and security requirements. These informal requirements then become the input of the requirements specification phase.

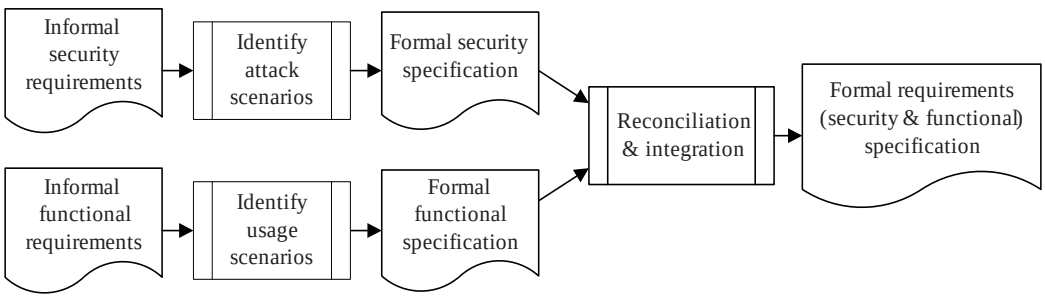
Requirements Specification

Software requirements are usually referred to as functional requirements that specify the behavior of the system, that is, the relationship between inputs and outputs. A nonfunctional software requirement describes not what the software will perform, but how effectively the software will perform. For example, usability, reliability, interoperability, scalability, and security are some of the nonfunctional properties of software systems. Like most other nonfunctional requirements, security is regarded as an important factor to the success of any software project, and it is usually incorporated in the system development in a subjective manner.

There are some inherent difficulties associated with security requirements specification: security requirements are often difficult to be specified in the early phases of the software development life cycle since the software model tends to be very abstract, and is usually presented in an informal manner. In addition, security risks are difficult to be estimated, and as the software development progresses, developers receive new information about the system security (Zhang & Zulkernine, 2004). Given that, it is often difficult to specify security measures and integrate them into the actual system from the very beginning of the software development process. In most cases, security requirements are not addressed at all during the early design phases.

In the software security engineering process, both security and functional requirements are specified in the requirements specification phase (see Figure 3). We believe that the weaknesses of

Figure 3. The requirements specification phase of the software security engineering process



software controlled systems with respect to the software security can be alleviated by specifying functional and security requirements using formal methods, and then by reconciling and integrating the two sets of formal specifications at an abstract level. As the software design evolves, the specifications are refined to reflect the progressive changes towards the end product. In this way, security requirements get closely involved in the later stages of software development.

Scenarios have been widely used as a technique for elicitation, analysis, and documentation of software requirements. A scenario is a sequence of a limited number of related events or actions that represents some meaningful high level operation in a system. The length of a scenario may vary from a single event to all the events of a system. However, all the events of a single scenario must occur during a particular run of the system. In this phase of the software security engineering process, two types of scenarios are identified based on the corresponding requirements: usage scenarios and attack scenarios. The usage scenarios are selected based on the operational profile of the intended software system. The attack scenarios can incorporate expert knowledge on security, and they may be ranked based on their importance in the system being implemented. For example, defending against denial of service attack in a client machine may not be as important as it is in a server machine.

A scenario can be represented in various forms: natural language, pseudo code, data flow diagrams, state machine based notations, communicating event based notations, etc. To avoid inconsistency and redundancy, it is highly recommended that both usage scenarios and attack scenarios should be defined using the same specification language. In the literature, we find a number of formal software specification languages and a relatively fewer number of attack languages to describe attack scenarios (Eckmann, 2001; Vigna, Eckmann, & Kemmerer, 2000). However, currently available software specification languages are not completely suitable for specifying the security related aspects of software systems, nor are attack specification languages suitable in specifying functional requirements. As an initial attempt, software specification languages can be translated to attack languages and vice versa. For example, in (Zhang & Zulkernine, 2004), a formal software requirements specification language of Microsoft called Abstract State Machine Language (AsmL) (Barnett, Grieskamp, Gurevich, Schulte, Tillman, & Veanes, 2003) are automatically transformed to a high level attack specification language called State Transition Analysis Technique Language (STATL) (Vigna, Eckmann, & Kemmerer, 2000), which can be preprogrammed and loaded into an intrusion detection system at runtime. The translation scheme will initially help to close the gap between security requirements and functional requirements with respect to their modeling and implementation.

Software Design

A software architecture or software design defines the software components of a system and the interactions among these components. The examples of application components include user interfaces, databases, and middleware (Kazman, Carriere, & Woods, 2000). Both functionality and security issues are interconnected to where the software components are deployed and how they communicate with each other. For example, the architecture and its protection mechanisms depend on whether the system will operate as a centralized or a distributed one. The software components should be deployed based on the trustworthiness and information flow among them following the principle: “what needs to be protected and from whom those things need to be protected, and for how long protection is needed” (Viega & McGraw, 2001).

In the software security engineering process, a software system should be designed based on the following two principles: compositional software architecture and scenario based software architecture. The following paragraphs discuss these two principles in the context of software security engineering.

New attacks are made to the software systems and security loopholes may be discovered at any point during the life of software. Flexible software architectures are desirable to make the design changes in those situations. A compositional software architecture is one in which the changes required for the whole system can be obtained by applying the changes to its components. As a result, whenever a new attack is encountered, the system architecture can be easily adapted to defend against the attack if a clear and clean notion of component is present. This is only possible if the principle of compositionality and the security concerns were taken into consideration in defining each software component in the software design phase. This adaptive architecture is very useful for rapidly evolving security policies in the post deployment system administration period, and to facilitate the re-engineering of security features into legacy systems. We define the concept of compositionality with respect to software security engineering following the definition of compositional specification and verification (de Roever, 1997). Assume that a system C is composed of the components C1, C2,..., and Cn by employing a set of composition operators. Let the security requirements of C1, C2,..., and Cn be S1, S2,..., and Sn respectively. A compositional design implies that the whole system C satisfies the complete specification S if each component Ci satisfies Si for i= 1,...,n. Therefore, any change of security requirements in any of the components can be addressed and verified without modifying the other components (in a true compositional case).

Scenarios are employed to build the software architecture of a system with a view to incorporating different features described by each scenario (Kazman, Carriere, Woods, 2000). Scenarios are very useful to compare design alternatives, and to understand a legacy system by analyzing the system responses in the context of different environments. Booch et al compared scenario-based design to "making a film" (Jacobson, Booch, & Rumbaugh, 1999). However, scenario-based design of a software system is more than making a single movie as scenarios may be composed in parallel, alternative, or iterative manner in order to build the desired composite system. Similarly, the potential attacks to a system can be specified using attack scenarios, which initially may be abstract, and are refined as the design evolves by considering all security relevant issues.

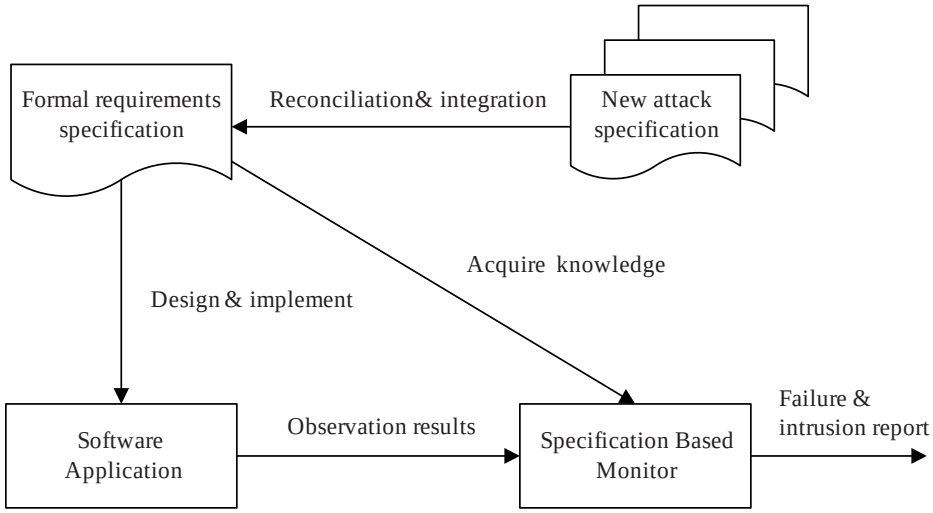
Implementation

Security requirements define the behavior of a system, which should not occur. A programmer working for a particular implementation or an intruder who somehow manages to gain access to the source code can initiate certain actions in the program that may cause target system behavior which should not be allowed. During programming, we should ensure that potential security flaws, identified as security requirements and avoided in the earlier phases of the software security engineering process are not re-introduced in the source code. Any unexpected behavior with respect to program security is called 'security flaw', while 'vulnerability' is considered as a special security flaw (Pfleeger & Pfleeger, 2003). In (Landwehr, Bull, McDermott, & Choi, 1994), the surveyed 'program security flaws' are divided into two categories: intentional and inadvertent. Intentional flaws are categorized as malicious and non-malicious, while inadvertent flaws are divided into the following six types: incomplete or inconsistent validation error, domain error, serialization or aliasing, insufficient identification or authentication, boundary condition violation, and other potential exploitable logic errors.

Given the varying and intricate nature of the above mentioned security flaws, security engineers should participate in coding or understand the code well enough to identify potential security flaws, and convey those to software developers. Security engineers may participate in formal code inspection and review processes to document any security relevant issues present in the source code. More specifically, in the software security engineering process, we propose to follow the principle of code refactoring successfully employed in the context of extreme programming (Beck, 1999). In extreme programming, code refactoring refers to the practice of restructuring the source code without changing the intended functionalities of the system such as removing any duplicate or unreachable code and re-organizing code for simplicity and flexibility. It is important to confirm that refactoring does not introduce any new faults or security flaws into the code. Therefore, individual refactoring steps should be small to address the above mentioned security flaws, and each change should be tested to ensure that the implementation is as functional and self-protected as it was before the refactoring. Note that refactoring is not intended to replace testing in the software security engineering process but rather to complement testing by increasing the confidence in the source code of security critical applications. Once the functionalities required for the system are implemented, refactoring is employed to change the program so that the software can control its vulnerabilities against attacks from intruders while still passing all of the functional requirements tests.

The software should be tested for both functional and security requirements. Based on these requirements along with other non-functional requirements such as performance and quality of service identified in the early phases of the software security engineering process, the test cases for the software system are derived. Vulnerability scenarios of the software that can be exploited for intrusions or security violations are tested. A "read teaming" approach may be used whereby a system is tested using simulated scenarios under which it may be attacked (Viega & McGraw, 2001). We have to ensure that the system is tested in the network environment in which the system will actually run. Keeping that in mind, the testing of

Figure 4. Monitoring an operational software application using a specification based monitor



the software can be divided into two steps: unit testing that tests a host for security; and integration testing that tests the whole system by deploying the system in the target network configuration guided by the applicable security policies.

System Operation

A number of studies have shown that professionally written programs may contain between one and ten faults per thousand lines of codes. Moreover, despite rigorous use of different 'protective shields', there exist security loopholes which elude those protection efforts, and unauthorized intrusions may occur when the network is operational. The difficulty in releasing correct and secure software has resulted in a growing interest in the usefulness of operational stage monitoring of software systems. However, existing intrusion detectors cannot monitor the behavior of software systems, while the current software monitors cannot detect security violations in the target systems.

In software security engineering, a software monitor should also have the capability of detecting intrusions (see Figure 4). The monitor passively observes some of the actual events occurring in a software application at runtime and reports the occurrences of intrusions or behavioral failure. To detect any intrusion or behavioral failure in a target system, the monitor must have the knowledge of what is the ideal or normal behavior of the monitored system. This knowledge can be obtained from the system's formal (behavioral and security) specification identified in the requirements specification phase. Note that this requirements specification was originally used to design and implement the software application as shown in Figure 4. During system operation, the behavior of the monitored system when it is under attack by the intrusions known later on can be specified. Similar to previously considered intrusions, it is also obvious that the newly discovered intrusions may change the behavioral profile of a host or a network. Therefore, new attack specifications are reconciled with the existing formal requirements specification. The monitor interprets the formal specification using the observed events of the monitored host or the network in order to generate a report on intrusive activities as well as unexpected behavior.

CONCLUSIONS AND RECOMMENDATIONS

In this chapter, we have presented a software security engineering process based on the waterfall software process model. The adaptation of the proposed process into other existing software engineering process models is straightforward. It is worth mentioning here that many new engineering disciplines such as software engineering and security engineering have incorporated many concepts and principles from other mature engineering disciplines. Similarly, the individual concepts and techniques suggested in the proposed process are not completely new. What is new in this chapter is their systematic integration and organization into a process that will lead to secure software systems.

It is very difficult to achieve two sets of desired objectives while they "trade off against one another" (Viega & McGraw, 2001). For example: an application software may require root level access privileges in a system to run the application while in some cases it may mean weakening the security of the target system. We believe that the trade off among these goals is very application dependent. The potential conflict between software characteristics and network defenses should be resolved early in the development phase, that is, in the requirements specification phase. In other words, security measures must not hamper the expected quality of service of a system, while at the same time any functionality should not be

achieved at the cost of weakening security. Given that, a software security engineering process should be employed, where functionalities of a system must be specified and defined by considering the security concerns or requirements with respect to the system.

Currently available specification languages are not completely suitable for defining security related aspects of software behavior. The standardized formal software specification languages used in industry should be investigated and extended (if needed) for expressing security related features of target systems. However, it may not be always feasible to formally specify all of the suspicious activities occurring in the target system. Therefore, the formal specification may be supplemented by the statistical components of the operational profiles of the system, users, or intruders. Statistical analysis may be performed to detect any additional intrusions.

History of well-established engineering disciplines shows the key role that automatic development tools played in their advancement. Automatic tools can play a similar role in the area of software security engineering to improve the productivity and quality of the software with respect to security. Existing Computer Aided Software Engineering (CASE) tools cannot be used for software security engineering since their focus is solely on software not security. Both security and software along with productivity and quality, need to be addressed in the CASE tools. The availability of such tools is likely to provide support to further the evolution of software security engineering.

Though we have described the proposed process independent of the current software development paradigms such as procedural, object oriented, or component based, it is worth mentioning here the issue of trustworthiness of software components. With the increase in component based software development, the security of components has become an important issue due to design of software systems composed of untrusted components from independent sources. It is still an open question as to the level of security one can achieve when integrating third party components of a software system (Lindqvist & Jonsson, 1998). In component based software development, components interact to accomplish the functionalities of software systems. In future, a metadata based approach can be used to address the security issues in component interactions since metadata describes the static and dynamic aspects of the components including the accessibility by their users (Orso, Harrold, & Rosenblum, 2001).

REFERENCES

An Early Iteration, *Proc. of the Workshop on Specification and Automated Processing of Security Requirements - SAPS'04*, Linz, Austria.

Anderson, R. (2001). *Security Engineering - A Guide to Building Dependable Distributed System*, Wiley.

Barnett, M., Grieskamp, W., Gurevich, Y., Schulte, W., Tillman, T., & Veanes, M. (2003). Scenario-oriented modeling in AsmL and its instrumentation for Testing, *Proc. of 2nd International Workshop on Scenarios and State Machines: Models. Algorithms, and Tools*, 8-14.

Basin, D., Doser, J., & T. Lodderstedt. (2003). Model Driven Security for Process Oriented Systems, *Proc. of the 8th ACM symposium on Access control models and technologies.*

Beck, K. (1999). *eXtreme Programming Explained: Embrace Change*, Addison Wesley.

Boehm, B. W. (1988). A spiral model of software development and Enhancement, *IEEE Computer*, 21(5), 61-72.

de Roever, W. (1997). The Need for Compositional Proof Systems: A Survey, *Compositionality: The Significant Difference (COMPOS '97), Lecture Notes in Computer Science, 1536*, 1-22, Springer-Verlag.

Dermott, J., Fox, C. (1999). Using Abuse Case Models for Security Requirements Analysis, *Proc. of the 15th Annual Computer Security Applications Conference*, Phoenix, Arizona.

Devanbu, P. T. & Stubblebine, S. (2000). Software Engineering for Security: A Roadmap, *The Future of Software Engineering*, Anthony Finkelstein (Ed.), *International Conference on Software Engineering*, Limerick, Ireland, 227-239.

Eckmann, T. (2001). Translating snort rules to STATL scenarios, *Proc. of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001), Lecture Notes in Computer Science*, 2001, 69-84.

Finkelstein, A. & Kramer, J. (2000). Software Engineering: A Roadmap. *The Future of Software Engineering*, Anthony Finkelstein (Ed.), *International Conference on Software Engineering*, Limerick, Ireland, 5-22.

Flechals, I., Sasse, M. A., & Hailes, S. M. V. (2003). Bringing Security Home: A process for Developing Secure and Usable Systems, *Proc. of the New Security Paradigms Workshop*, Ascona, Switzerland.

Gilliam, D. P., Wolfe, T. W., & Sherif, J. S. (2003). Software Security Checklist for the Software Life Cycle, *Proc. of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, (WETICE '03*).

IEEE. (1999). Software Engineering: Customer and Terminology Standards, *1*, IEEE.

ISSEA (2003). SSE-CMM: Model Description Document, Version 3.0, SSE-CMM – ISO/IEC 21827, International Systems Security Engineering Association, http://www.sse cmm.org/model/model.asp.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*, Addison Wesley.

Jürjens, J. (2003). *Secure Systems Development with UML*, Springer-Verlag, December 2003.

Kazman, R., Carriere, S. J., & Woods, S. G. (2000). Toward a Discipline of Scenario-Based Architectural Engineering, *Annals of Software Engineering, 9*, 5-33.

Landwehr, C. E., Bull, A. R., McDermott, J. P., & Choi, W. S. (1994). A taxonomy of computer program security flaws, *ACM Computing Surveys*, 26(3). 211-254.

Laprie, J. (1992). *Dependability: Basic Concepts and Terminology - In English, French, German, and Japanese*, Vienna, Springer-Verlag.

Lindqvist, U. & Jonsson, E. (1998). A Map of Security Risks Associated with Using COTS, *IEEE Computer*, 60-66, http://downloads.securityfocus.com/library/cots98.pdf

Lodderstedt, T., Basin, D., & Doser, J. (2002). SecureUML: A UMLBased Modeling Language for Model-Driven

McGraw, G. (2002). Managing Software Security Risks, *IEEE Computer,* 99-101.

Oikinomopoulos, S. & Giritzails, S. (2004). Integration of Security Engineering Into the Rational Unified Process –

Orso, A., Harrold, M.J., & Rosenblum, D. (2001). Component Metadata for Software Engineering Tasks, *Proceedings of the EDO 2000, Lecture Notes in Computer Science*, 1999, Springer-Verlag.

Pfleeger, C. P. & Pfleeger, S. L. (2003). *Security in Computing*, Prentice-Hall.

Pressman, R. P. (2001). *Software Engineering – A Practitioner's Approach*, McGraw Hill.

Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques, In WESCON Technical Papers,

1970, (Reprinted) *Proceedings of the Ninth International Conference on Software Engineering*, 1987, 328–338.

Security, *Proceedings of the Fifth International Conference on the Unified Modeling Language - the Language and its applications*, Germany.

Sindre, G. & Opdahl, A. L. (2000). Eliciting security requirements by misuse cases, *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems*, 174-183.

Thayer, R. H. (2002). Software System Engineering: A Tutorial, *Computer*, IEEE Computer Society Press, California, USA.

Viega, J. & McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley Pub Co.

Vigna, G., Eckmann, S. T., & Kemmerer, R. A. (2000). Attack languages, *Proceedings of the IEEE Information Survivability Workshop*, Boston, MA.

Wolf. A. L. (2004). Is Security Engineering Really Just Good Software Engineering?. *Keynote Talk, ACM SIGSOFT '04/FSE-12*, Newport Beach, CA, USA.

Zhang, Q. & Zulkernine, M. (2004). Applying AsmL Specification for Automatic Intrusion Detection, *Proc. of the Workshop on Specification and Automated Processing of Security Requirements – SAPS'04*, Linz, Austria.

ENDNOTES

1 For brevity, functional software requirements are mentioned as software requirements or functional requirements.

2 IEEE definitions (IEEE, 1999) of 'fault' and 'failure' are defined from a different perspective. However, (Wolf, 2004) view "vulnerability" as a "fault" since an attack cannot cause a security failure if no vulnerability exists.

This work was previously published in Enterprise Information Systems Assurance and Systems Security: Managerial and Technical Issues, edited by M. Warkentin, pp. 215-233, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 7.8
Trusting Computers Through Trusting Humans:
Software Verification in a Safety–Critical Information System

Alison Adam
University of Salford, UK

Paul Spedding
University of Salford, UK

ABSTRACT

This article considers the question of how we may trust automatically generated program code. The code walkthroughs and inspections of software engineering mimic the ways that mathematicians go about assuring themselves that a mathematical proof is true. Mathematicians have difficulty accepting a computer generated proof because they cannot go through the social processes of trusting its construction. Similarly, those involved in accepting a proof of a computer system or computer generated code cannot go through their traditional processes of trust. The process of software verification is bound up in software quality assurance procedures, which are themselves subject to commercial pressures. Quality standards, including military standards, have procedures for human trust designed into them. An action research case study of an avionics system within a military aircraft company illustrates these points, where the software quality assurance (SQA) procedures were incommensurable with the use of automatically generated code.

INTRODUCTION

They have computers, and they may have other weapons of mass destruction. Janet Reno, former US Attorney General

In this article our aim is to develop a theoretical framework with which to analyse a case study where one of the authors was involved, acting as an action researcher in the quality as-

surance procedures of a safety-critical system. This involved the production of software for aeroplane flight systems. An interesting tension arose between the automatically generated code of the software system (i.e., 'auto-code'—produced automatically by a computer, using CASE [Computer Aided Software Engineering] tools from a high level design) and the requirement of the quality assurance process which had built into it the requirement for human understanding and trust of the code produced.

The developers of the system in the case study designed it around auto-code—computer generated software, free from 'human' error, although not proved correct in the mathematical sense, and cheaper and quicker to produce than traditional program code. They looked to means of verifying the correctness of their system through standard software quality assurance (SQA) procedures. However, ultimately, they were unable to bring themselves to reconcile their verification procedures with automatically generated code. Some of the reason for this was that trust in human verification was built into (or inscribed into [Akrich, 1992]) the standards and quality assurance procedures which they were obliged to follow in building the system. Despite their formally couched descriptions, the standards and verification procedures were completely reliant on human verification at every step. However these 'human trust' procedures were incompatible with the automated production of software in ways we show below. The end result was not failure in the traditional sense but a failure to resolve incommensurable procedures; one set relying on human trust, one set on computer trust.

Our research question is therefore: How may we understand what happens when software designers are asked to trust the design of a system, based on automatically generated program code, when the SQA procedures and military standards to which they must adhere demand walkthroughs and code inspections which are impossible to achieve with auto-code?

The theoretical framework we use to form our analysis of the case study is drawn from the links we make between the social nature of mathematical proof, the need to achieve trust in system verification, the ways in which we achieve trust in the online world, the methods of software engineering, and within that, the software quality movement and the related highly influential domain of military standards.

In the following section we briefly outline the social nature of mathematical proof. The next section discusses the debate over system verification which encapsulates many of the ideas of mathematical proof and how such proofs can be trusted by other mathematicians. The article proceeds to consider 'computer mediated' trust, briefly detailing how trust has been reified and represented in computer systems to date, mainly in relation to the commercial interests of e-commerce and information security. Trust is particularly pertinent in the world of safety-critical systems, where failure is not just inconvenient and financially damaging, although commercial pressures are still evident here, but where lives can be lost. The model of trust criticised by e-commerce critics is more similar to the type of trust we describe in relation to safety-critical systems, than one might, at first, expect. Understandably, we would like to put faith in a system which has been mathematically proved to be correct. However computer generated proofs, proofs about correctness of computer software, and automatically generated code are not necessarily understandable or amenable to inspection by people, even by experts. The question then arises of whether we can bring ourselves to trust computer generated proofs or code, when even a competent mathematician, logician,or expert programmer cannot readily understand them.

Following this, we describe the evolution of software development standards and the SQA movement. We argue that the development of quality assurance discourse involves processes of designing human ways of trusting mathematical evidence into standardisation and SQA. Military

standards are an important part of the SQA story, having consequences far beyond the military arena. Standards are political devices with particular views of work processes inscribed (Akrich, 1992) in their design. We note the way that military standards, historically, moved towards formal verification procedures only to move back to rely more on 'human' forms of verification such as code walkthroughs and inspections in the later 1990s. The story is shot through with a tension between finding ways to trust the production of information systems and finding ways to control them. Formal methods, based on mathematical proof offer the promise of control, but only if we can bring ourselves to trust a proof generated by a machine rather than a proof constructed by another person. We present the background to the case study in terms of a description of the complex 'post cold war' military and commercial environment. This is followed by a description of the action research methodology employed in the project, an outline of the case study and an analysis of the case study findings in terms of our theoretical framework. In the conclusion we briefly note that mathematicians and others are gradually finding ways of trusting computers.

THE SOCIAL NATURE OF MATHEMATICAL PROOF

At first sight, the concept of mathematical proof appears to be relatively simple. The idea of a logical and rigorous series of steps, leading from one or more starting positions (previous theorems or axioms) to the final conclusion of the theorem seems to be the basis of mathematics. The concept of mathematical proof leading inexorably to true and incontrovertible truths about the world is very compelling. It is not surprising that we would like to apply the apparent certainty and exactness of mathematical approaches to computer programming. However if we consider briefly how agreement on mathematical proof and scientific truth is achieved by communities of mathematicians, then the social and cultural dimension of proof, as an agreement amongst trusted expert witnesses, reveals itself.

With the epistemological and professional success of mathematical proof, many of the cultural processes which go into making a proof true sink from consciousness and are only rendered visible in times of dispute; for example as in claims to the proof of Kepler's conjecture or Fermat's last theorem (Davies, 2006; Kuhn, 1962; Singh, 1997). Only on the margins then do we call into question our ability to trust these people when a mathematical proof cannot be agreed to be true by an expert community of mathematicians, as sometimes happens.

The apparently pure and abstract nature of mathematical proof fairly quickly breaks down when we inspect it more closely. In particular, when there is disagreement about a proof, the nature of proof is revealed as a social and cultural phenomenon; the matter of persuading and convincing colleagues. DeMillo, Lipton, and Perlis (1977, p. 208) wrote

Mathematicians talk to each other. They give symposium and colloquium talks which attempt to convince doubting (sometimes hostile) audiences of their arguments, they burst into each others' offices with news of insights for current research, and they scribble on napkins in university cafeterias and expensive restaurants. All for the sake of convincing other mathematicians. The key is that other mathematicians are inclined to listen!

This traditional approach towards mathematical proof, which could be described as one of *persuasive rigorous argument between mathematicians leading to trust*, is not the only way to address the idea of proof. A quite different approach appeared in the 1950s and was based on the work on logic developed by Bertrand Russell and others in the 1930s and used the newly invented electronic computer. This new

logic-based approach was not dependent on the computer, but the computer's speed and accuracy had a major impact on its application to the proof of theorems in replacing the persuasive rational argument of competent mathematicians with a *formal* approach which sees any mathematical proof as a number of steps from initial axioms (using predicate logic), to the final proof statement (based purely on logical inference) without the requirement of a human being.

Many proofs can be completed by either method. For instance, many persuasive rigorous argument proofs can be converted to formal proofs (MacKenzie, 2004). It should be emphasised, however, that there is a real difference between the two types of proof. We are not simply talking about a machine taking on the role of a competent mathematician. Some proofs which are readily accepted by mathematicians rely on arguments of symmetry and equivalence, analogies, and leaps of imagination, which humans are very good at understanding but which a formal logic approach cannot replicate. Symmetry and analogy arguments of this type cannot be established by formal methods based on logical progression because symmetry relies on understanding semantics and cannot be gleaned from the syntax of a proof.

Whereas the persuasive rigorous argument, the 'human' approach, has been used for thousands of years, the formal or 'computer generated' approach has been in use for only about half a century. Clearly, the two methods are not treated in the same way by the expert community of mathematicians. With a rigorous argument type of proof, although one may expend much energy convincing one's colleagues of the validity of the proof, the *potential* for coming to agreement or trust of the proof is there. Essentially, in trusting that a mathematical proof is correct, mathematicians are demonstrating their trust in other competent mathematicians. However, expert mathematicians clearly have trouble bringing themselves to trust computer proofs, for good reason, as a computer cannot explain the steps in its reasoning (Chang, 2004).

COMPUTER SYSTEM VERIFICATION: TRUST AND THE SOCIAL

The preceding section contrasted the *use* of computer technology in a claimed proof: the formal method and the human 'rigorous argument' approach to proof. Although this is not the same thing as the proof or verification of a computer system *itself*, in other words the formal, computer generated proof that the computer system matches the specification, the question of whether we can trust the computer is exactly the same.

The idea of *proof* or *verification* of a program is quite different from simply testing the program. Typically, a large suite of programs might have thousands or millions of possible inputs, and so could be in many millions or even billions of states. Exhaustive testing cannot be possible. If a computer system is to be used in the well-funded and high-profile military field to control a space craft, aeroplane, or a nuclear power station, it is highly desirable if the system can be actually *proved* to be correct, secure, and reliable. Since testing, although vital, can never prove the system's correctness, more mathematical methods involving the notion of proof became of great interest in the late 1960s and have remained so ever since.

In fact the history of the verification of computer systems echoes that of mathematical proof, with basically the same two approaches: those who support the rigour of formal methods and those who believe that the purely formal, mechanised proof lacks the crucial element of human understanding (Tierney, 1993). In a paper to an ACM Symposium, DeMillo et al. (1977) argued that the two types of proof were completely different in nature, and that only the persuasive rigorous argument proof with its strong social aspect will

ultimately be believable and capable of earning *trust*

COMPUTER-MEDIATED TRUST

In ethical terms, trust is a complex phenomenon and is essentially a human relationship (Nissenbaum, 1999; Stahl, 2006). We think of trust in terms of a trustor who does the trusting and a trustee who is trusted. The trustee does not of course have to be human, but Nissenbaum (1999) suggests that the trustee should be a being to whom we ascribe human qualities such as intentions and reasons, what might be termed an 'agent.' Trust allows meaningful relationships and a vast range of intuitions to work. Nissenbaum (1999) argues that when we are guaranteed safety trust is not needed: 'What we have is certainty, security, safety – not trust. The evidence, the signs, the cues and clues that ground the formation of trust must always fall short of certainty; trust is an attitude without guarantees, without a complete warrant.' Intrusive regulation and surveillance are attempts at control and bad for building trust.

This generalised definition of trust clearly maps onto our description of mathematicians trusting proofs. They may not have complete certainty over the correctness of a mathematical proof, but they have good reason to trust a competent member of the community of expert mathematicians. Therefore they can trust the proof supplied by such a person.

Understandably, there has been much interest in trust in the online world, both in terms of online security and trust in e-commerce transactions. Nissenbaum (1999) suggests that excessive safety controls, say in e-commerce, may encourage participation but they limit experience: 'Through security we may create a safer world, inhospitable to trust not because there is distrust, but because trust cannot be nourished in environments where risk and vulnerability are, for practical purposes, eradicated.'

Stahl's (2006) take on trust in e-commerce shows another example of the intangible human nature of trust, which has become reified and commodified, so that it can be measured and exchanged in machine transactions. Like Nissenbaum (1999), Stahl points to the way that a trustor does not have complete control over a trustee; vulnerability and uncertainty must be accepted in a trusting relationship. This of course includes business transactions, and is especially important in e-commerce as many of the traditional ways of developing trust are absent from online transactions. Trust becomes a way of generating profit; small wonder that trust, including technological ways of creating trust and maintaining it, has been of so much interest in e-commerce. In the world of e-commerce research, trusts lose its relational aspects and becomes a form of social control. 'If trust is limited to calculations of utility maximisation in commercial exchange, then most of the moral underpinnings of the mechanisms of trust become redundant. Trust changes its nature and loses the binding moral quality that it has in face-to-face interaction.' (Stahl, 2006, p. 31)

Although, on the face of it, Nissenbaum's and Stahl's arguments on the problems of online trust in e-commerce are not the same as the issue of trust described in the body of this article, there are important congruencies which are very directly applicable to our characterisation of trust. Whether it is a human trusting another human or an expert mathematician trusting another expert mathematician to supply an accurate proof, the same relationship between trustor and trustee obtains.

For Nissenbaum and Stahl, the issue is what happens to trust when it is commodified within an online relationship. In other words, what happens when the human-trusting-human relationship is mediated by technology? In this article we also consider what happens when the human-trusting-human relationship—in terms of a human trusting another human's mathematical proof, or computer program—is replaced by a human

having to trust a machine. Of course, in this trustor-trustee relationship, the trustee, that is, the machine, cannot be understood in the way that another person can be.

The pressure to create computer-mediated trust is completely bound up with commercial pressures. The maximisation of profit drives the reification of trust in e-commerce. Similarly in the world of military avionics we describe, it is the commercial pressure of building systems more cheaply and faster which provides the impetus to turn over proofs, testing of programs, and automatic generation of code to a machine. A third aspect of similarity between Stahl's and Nissenbaum's view of computer-mediated trust and ours relates to the tension between trust and control. This is clearly present in the debate over trust in e-commerce. But it is also present in software quality discourse as we discuss below.

In the following section we briefly discuss some of the ways in which human trust has traditionally been built into procedures designed to verify program correctness, and how this can be seen to mirror an ideal group of mathematicians agreeing upon a mathematical proof.

BUILDING TRUST INTO A COMPUTER SYSTEM

We argue that, historically, much of the development of the software engineering discipline can be understood in terms of the development of procedures, through which we can convince ourselves to trust, and control, the development of information systems and the production of software. For instance, Myers' (1979) classic book on software testing explores the topic of human testing in detail, justifying methods such as formal *code inspections* and *code walkthroughs*. The differences between the two methods depend on different usages of the terms 'inspection' and 'walkthrough,' but the important point is that both involve a small group of professionals carefully reading through code together. We argue that this can be viewed as an imitation of the social (persuasive rigorous argument) form of proof described earlier where 'mathematicians talk to each other' in symposia and colloquia and so on (DeMillo et al., 1977). The original programmer should be in the group, analogous to the mathematician demonstrating a proof or principle to expert colleagues. The aim (as originally suggested by Weinberg [1971]—an 'egoless' approach) is to discover as many errors as possible rather than to try to demonstrate that there are none. So the team is to act as an idealised group of 'Popperian' scientists looking for 'refutations' (Popper, 1963). Under such an approach, one can never be entirely sure that the code is correct. But, as the walkthrough proceeds, the original programmer and the code inspection team can gradually come to trust the code as bugs are weeded out and fixed.

Myers claims positive advantages of code inspections and walkthroughs, including the value of the original programmer talking through the design (and thus spotting the errors). He also notes the ability of human testers to see the causes and likely importance of errors (where a machine might simply identify symptoms) and also the likelihood that a batch of errors will be identified simultaneously. Also the team is able to empathise with and understand the thought processes of the original programmer in a way which a machine arguably cannot. Importantly, the team can be *creative* in its approach. In working together they also, inevitably, form something of a sharing and trusting community (even if it is disbanded after a day or two).

The lesson gleaned from human verification techniques, such as walkthroughs and code inspections, is that these have been regarded, for some time, as reliable, if not exhaustive, ways of ensuring reliability of software.

SOFTWARE QUALITY ASSURANCE AND MILITARY STANDARDS FOR SOFTWARE

The software verification techniques of code walkthroughs and inspections are important parts of the armoury SQA. Effectively, we argue that SQA is a branch of software engineering which formalises and standardises the very human methods of trust, and ultimately control outlined above, which we need to build into software engineering procedures. The SQA movement is an important part of the story of the growth of software engineering because of its quest for rigour and control of potentially unruly programs and programmers.

First of all, SQA offers a promise of rational control over software, the software development process, and those who produce software. Software quality criteria include features for directing, controlling, and importantly, measuring the quality of software (Gillies, 1997). 'Qualification' is achieved when a piece of software can be demonstrated to meet the criteria specified in these quality procedures. An important aspect of SQA involves demonstrating that software meets certain defined independent standards.

The development and adherence to software standards is a very important part of the story of SQA. Generic industry standards are available, but also of much interest—particularly for the case study set out later in the article—are military standards. Indeed, the defence industry is so influential that Tierney (1993) argues that military standards influence software engineering far beyond applications in defence. Hence military standards are a very important part of SQA, and ultimately are important in formalising ways in which designers of computer systems can come to trust the systems and the production of correct software.

A number of military standards have been developed to regulate and control the use of software in defence applications. For instance, US standards DOD-STD-2167A (1988), MIL-STD-498 (1994), and ISO/IEC 12207 (1995) respectively established the requirements for software development and documentation in all equipment to be used by the US military (and effectively that of all Western armed forces), introduced object oriented development (OOD) and rapid application development (RAD), then broadened the scope of international standards to include acquisition and maintenance. (DSDM Consortium, 2006).

The relevant UK standard 00-55, (MoD, 1997) *Requirements for Safety Related Software in Defence Equipment*, was published in 1997 and echoes much of MIL-STD-498, but moves the discussion on provably correct software in a particular direction. At first sight, this seems highly significant to the current argument, because it clearly expressed a preference for *formal* methods, in other words mathematical procedures whereby the software is proved to be correct by a machine (MacKenzie, 2001).

Tierney (1993) argues that the release of UK Defence Standard 00-55 in draft in 1989 had the effect of intensifying the debate over formal methods in the UK software engineering community. It devoted as much space to regulating and managing software development labour processes as the techniques and practices to be used for formal designs. This reinforces our argument that SQA is concerned with control of work processes and those who perform them, the software developers. On the one hand, many argued that mathematical techniques for software development and verification could only ever be used sparingly, as there simply was not enough suitable mathematical expertise in most organisations and it increased software quality at the expense of programmer productivity. On the other side, those from a more mathematical camp argued that there was commercial advantage in proving software correctness as errors could be trapped earlier in the software development cycle (Tierney, 1993, p. 116).

Designed into the MoD (UK Ministry of Defence) standard was a view of safety-critical software as an important area of regulation and control. Some of the reason for this was a change in its own organisation from the 1980s. The UK government sought to open up work traditionally done in-house by the MoD in its own research establishments to private contractors (Tierney, 1993, p. 118). Given that it had to offer its software development to the private sector, it built in ways of controlling it within its defence standards (Tierney, 1993, p. 118). Further political impetus was offered by the introduction of consumer protection legislation in the UK in the late 1980s which required software developers to demonstrate that their software had not contributed, in the event of an accident enquiry, and that they had demonstrably attended to safety. Thus we can see that in Def Stan 00-55, politics, in the shape of the MoD's need to open up software development to the private sector and also to avoid being held responsible for inadequate software in the event of an accident, played an important role.

However, more significantly, this document has itself been superseded in 2004 by (draft) standard 00-56 (MoD, 2004). Def Stan 00-55 has now become obsolete. The changes involved in Def Stan 00-56 are of great interest, in that the preference for formal method is lessened. In the new standard, it is accepted that provably correct software is not possible in most cases and that we are inevitably involved in a human operation when we attempt to show that code is reliable in a safety-critical environment. Without a more detailed consideration of the history of formal methods in the UK over the last decade, which is beyond the scope of the present article, a strong claim that the move back to more human methods of verification might be difficult to sustain. Nevertheless it is interesting to note the way that Def Stan 00-5, with its emphasis on formal approaches and attendant onerous work practices, has been consigned to the history books with a clear move back to human verification.

CASE STUDY CONTEXT

The case study relates to a large European military aircraft company (MAC) with which one of the authors was engaged as a researcher in a joint research project, lasting around three years, during the mid to late 1990s. A high proportion of the senior management were men and its culture was masculine in style, particularly emphasising an interest in engineering and technical mastery (Faulkner, 2000). Indeed there was much interest, pleasure, and admiration for elegant products of engineering (Hacker, 1991). When one of their fighter planes flew over (an event difficult to ignore on account of the engine noise), offices would clear as employees went outside to admire the display of a beautiful machine. A certain amount of military terminology was used, sometimes ironically, in day-to-day work. A number of employees had links with the armed forces. MAC was exclusively involved in the defence industry, with the UK's MoD being its largest customer and other approved governments buying its products.

As a manufacturing company in an economy where manufacturing was in steep decline and with its ties to the defence industry, if a major defence contract went elsewhere, jobs would be on the line. Despite the 'hi-tech' nature of its work, MAC had a traditional feel to it. The company had existed, under one name or another, right from the beginning of the avionics industry. The defence industry, and within that the defence aerospace industry, faced uncertain times as the UK government was redefining its expectations of the defence industry in post-Cold War times. It quickly came to expect much clearer demonstrations of value for money (Trim, 2001). Therefore, the 'peace dividend' brought about by the end of the Cold War meant uncertain times for the defence aerospace industry as military spending was reduced significantly (Sillers & Kleiner, 1997). Yet, as an industry contributing huge amounts to the UK economy (around £5

billion per annum in export earnings Trim (2001, p. 227)), the defence industry is hugely important in terms of revenue and employment. Defence industries have civil wings (which was the case with MAC) and it was seen as important that the defence side of the business did not interfere with civil businesses. For instance, BAE Systems is a partner in a European consortium and was pledged £530 million as a government loan to develop the A3XXX aircraft to rival the USA's Boeing 747 (Trim, 2001, p. 228).

Although not strictly a public sector organisation itself, its location in the defence industry put MAC's business in the public sector. However, in the UK, views of public sector management were undergoing rapid change in the mid 1990s and it was seen as no longer acceptable that the taxpayer should underwrite investment (Trim, 2001). Such firms were required to be more competitive and to be held more accountable financially. Hence, quality management and value for money were becoming key concepts in the management repertoire of the UK defence industry from the mid 1990s onwards. As we discuss in the preceding section, this was at the height of the UK MoD's interest in formal approaches to the production of software. In a climate where post-Cold War defence projects were likely to demand a shorter lead time, there was considerable interest in speeding up the software development process.

Computer technology and related activity clearly played a central role in MAC. One division of MAC, the Technical Directorate (TD), developed most of the airborne software (much of it real-time). This software clearly has a central role in ensuring aircraft performance and safety. Around 100 people were involved in developing systems computing software. It was in this division that Software Development System (SDS), a safety-critical airborne software system for flying military aircraft, was developed.

Research Methodology

The methodological approach of the research was based on action research (Myers & Avison, 2002). As several successful participant observation studies in technology based organisations have been reported in the literature (Forsythe, 2001; Low & Woolgar, 1993; Latour & Woolgar, 1979), an ethnographic approach holds much appeal. However, a strict ethnographic approach was neither feasible nor desirable in this study. As someone with technical expertise, the researcher could not claim to be the sociologist or anthropologist, more typical of reported ethnographic studies of technological systems (Low & Woolgar, 1993; Forsythe, 2001). This also meant that he was not 'fobbed off' by being directed into areas that the participants thought he wanted to look at or where they thought he should be interested in as happened in the Low and Woolgar (1993) case study. Based in the Quality Assurance Division (QAD) in the SQA team, early in his research, the researcher proved his technical credentials by helping run a workshop on software metrics and this helped to gain him full inclusion in the technical work. Although as a technical researcher, rather than a social researcher, it was arguably difficult for him to maintain the 'anthropological strangeness' which ethnographers look for in explaining the common sense and every day logistics of working life. In any case, he had been invited, through this research, to make a contribution to the improvement of SQA procedures. Therefore the research can be characterised as a form of action research (Baskerville & Wood-Harper, 1996), where potential improvements to SQA were to be seen as the learning part of the action research cycle.

Although action research receives a mixed press from the IS research community (Baskerville & Wood-Harper, 1996; Lau, 1999), it is nevertheless seen as a way of coming to grips with complex social settings where interactions with information technologies must be understood

within the context of the whole organisation. Baskerville (1999) notes the growing interest in action research methods in information systems research. Two key assumptions are that complex social settings cannot be reduced for meaningful study and that action brings understanding (Baskerville, 1999). The culture of MAC was extremely complex, as we characterise above and discuss again in what follows. Arguably, key elements would be lost were the researcher to have adopted a more distant role, relying on interviews and questionnaires rather than becoming fully immersed and contributing to the detail of the project. The researcher adopted an interpretivist approach, looking to the interpretations of the other participants of the research. But by allowing for social intervention he became part of the study, producing shared subjective meanings between researcher and subjects as coparticipants in the research (Baskerville, 1999).

For a period of over one year out of the three that the whole project lasted, the researcher spent, on average, one day per week working with MAC staff with access to a variety of staff across the organisation, and was therefore able to participate in a range of meetings and workshops and to gain a familiarity with the individuals concerned. This could not easily have been gained from interviews or surveys. These events included meetings where software quality staff considered quality policy, such as the implication of international standards, to broader meetings where technical staff were considering development methods in detail. Free access was allowed to relevant policy and development documents. This permitted an overview of the detailed practices and culture of this large and complex organisation.

Analysis of Case Study Findings

The initial remit of the researcher was to work with staff to optimise the use of software quality assurance within the organisation. The use of cost benefit analysis was originally suggested by senior management. Given our characterisation of the UK defence industry's particular focus on management of quality and value for money, as described above, it is entirely in keeping with the industry's changing needs that the researcher was initially directed into these areas. The researcher viewed it as problematic to assign monetary cost to SQA activities, and even harder to assign monetary benefits. However, these concerns were never addressed directly in the project as it soon emerged that there was greater interest in a new approach to software development being pioneered by MAC.

Ince (1994, p. 2-3) tells the story of a junior programmer's first day in a new job. A senior programmer shows him around, advising him where to buy the best sandwiches at lunchtime, where to find the best beer after work, and other similarly important matters. Then the senior colleague points to a door. 'Whatever you do don't go through that door, the people there have been given the job of stifling our creativity.' The door, of course, led to the quality assurance department.

The staff of MAC's Quality Assurance Division expressed some similar feelings, albeit less dramatically. They wanted to act as consultants, offering a measure of creativity to the technical development process, although safely wrapped in appropriate quality assurance processes, but all too often they felt like the police. The strong awareness of the safety-critical nature of software development, and the related fairly advanced organisation of quality assurance in MAC, thanks in no small measure to the necessity to adhere to MoD standards, meant that SQA was never going to get quite the negative press that it attracted in Ince's (1994) anecdote. Nevertheless, there was still some feeling that the Quality Assurance Division could be brought on board in a project some time after the Technical Division had time to do the creative part.

Hence, TD had been prototyping the new SDS system for about a year when they decided to bring in Quality Assurance Division. As we explain below, the newness of the style of development in SDS made it unclear how it was to be quality assured. Unsure of how to proceed, the SQA manager turned to the researcher for suggestions. The researcher now became involved in investigating the use of the new software development approach, which would involve the inclusion of computer generated program code ('auto-code') in safety-critical airborne software systems, leading to the approval of the new approach and its incorporation into MAC's software quality assurance systems.

Although there has been a long tradition of using computers to aid the process of software engineering itself, such CASE tools (Pressman, 2005) have not generally been used to generate safety-critical code (this was always written by human programmers). The new MAC SDS was an ambitious system whose targets were principally to reduce avionics systems development time by 40% and the cost by 30%, whilst maintaining the very high quality standards necessary for computer-based system which fly—and therefore can crash—military aircraft.

A key aspect of SDS was process integration using an integrated modeling environment. There was consequentially a heavy reliance on automated methods. A specification was developed in a formal modeling language and this generated programming code automatically. In particular, automatic code generation was eventually to lead to aircraft flying 'auto-code' in safety-critical systems. Two aspects of SDS stand out in the climate of defence spending of the mid 1990s. First, there was pressure to reduce costs and show value for money. Second, the use of formal methods in computer programming received a huge boost in the mid-1990s through the Defence standard DEF Stan 00-55 which mandated the use of formal methods base approaches in safety-critical software. It is not surprising that there was considerable interest in a system which offered the promise of considerably reduced software production times.

MAC invested a great deal of money and time in SDS in the hope that the improved time-scales which SDS promised, together with reduced costs, could keep major current aircraft developments on course. This was particularly important in an environment of political intervention and considerable public interest and concern over escalating costs and delivery times in the public sector, including the defence industry. These benefits could only accrue to MAC if the quality, that is, correctness of the software, could be assured.

SDS was heavily dependent on software (CASE) tools. MAC had used these for many years, and had procedures in place for their qualification (i.e., acceptance) in certain circumstances. However, these applied to mission-critical rather than safety-critical systems. Furthermore, the movement towards auto-generated code led to a different environment than one where tools improved and speeded up the design process, but where failure would show up and be merely time-wasting. There was seen to be a need for a major improvement/update of these procedures, a quantum change, before they would be acceptable for safety-critical applications.

Some tools being used had major world-wide user communities, associated academic conferences, and came from supposedly secure and reliable suppliers. Others might not be so well supported, both intellectually and commercially. (For instance, it might be no use having an ideal tool if the supplier was small and unlikely to survive for many years.) Methods already existed for supplier qualification. These methods were undertaken by software quality staff. However, the qualification of these suppliers could be a crucial issue in the qualification of the tool and ultimately the integrity of the avionics system. The issue was not merely one of qualification, it was also one

of *demonstration* of qualification to customers. Ultimately, the need in some sense to *prove* the new methods became paramount. Hence we can see that quality procedures did not just involve procedures, such as code walkthroughs through which software teams could persuade themselves to trust program code, they also applied to the question of choosing and trusting suppliers.

A number of meetings took place with members of the SDS team. This discussion was very useful for an understanding of SDS and gave the researcher a richer understanding of the SQA needs. It soon became apparent that the necessary fundamental problems with SQA in SDS were going to be difficult to answer.

The difficulties were centred around two conflicting ideas. The first of these was that for the *persuasive rational argument* approach to be successful there would be a need for a group of professionals to participate in code walkthroughs, with consequent discussion and persuasion. On the face of it, this was simply not possible, since the computer which wrote the auto-code could not take part in such a discussion. Alternative approaches were considered. Clearly there would be a stage before the auto-code (at the requirements specification level) where human agents were involved, but this was found to be too high level to meet the relevant military standards (the US MIL-STD-498 [1994] and the UK standard 00-55 [MoD, 1997]). Both standards are very specific about the exact conduct of the necessary walkthrough. It had to be a *code* walkthrough.

On the other hand, for the *formal proof* approach method to work, there would first need to be such a formal proof. This did not seem within the capability of the QAD itself, despite the division being quite well resourced. MAC referred back to the auto-code tools suppliers, but once again there was no such proof and no realistic possibility of achieving such a proof. Although MAC was an important customer for the auto-code tool suppliers, they were not prepared to expend the necessary resources. Furthermore, a 'weakest link' argument demonstrates a fundamental flaw with the formal approach in computer systems. If the auto-code tool itself could be formally verified, it would then become necessary also to consider the operating system on which the tool would run and the hardware systems involved. Potentially this could involve a seemingly infinite regression of hardware and software systems having to be proved correct, where the system is only as good as its weakest link. Frustration grew as no solution was forthcoming and ultimately SDS was shelved indefinitely.

We have argued that mathematical proof is essentially a human achievement between members of the expert mathematical community who are persuaded of the correctness of mathematical proofs because they trust each other. These processes of trust are replicated in the procedures that have been developed in software engineering, and within that, software quality assurance. As part of the defence industry, developing safety-critical systems, MAC had highly developed SQA procedures which were obliged to follow international military standards. Their code walkthroughs, which are analogous to the ways mathematicians achieve trust in a proof, were an important part of such quality procedures. Formal methods offer the promise of an attractive certainty and control over software production and hence control over the work processes of human programmers. They also offer the promise of automatic verification of software systems which, potentially, could be much cheaper than traditional human based approaches to the verification of software through traditional SQA procedures.

SDS achieved very little despite the huge efforts put into it by the many people working for MAC. Although it was not, at the time, formulated in such stark terms, success was elusive because an attempt was being made to achieve the impossible: namely using auto-code whilst being held to quality assurance procedures which demanded

code walkthroughs which could not possibly be achieved in an auto-code system. Attempts were made to consider formally proving the correctness of the auto-code. In addition to supplier reluctance, this raised the spectre of the infinite regress. If one looks to proving the auto-code correct, then the operating system must be proved correct, the hardware platform and so on.

This was at the height of interest in formal methods for safety-critical systems for defence, a view embodied in Def Stan 00-55. The rise of formal methods is crucially linked to the defence industry. The interest in formal methods and automated approaches arrived as pressure mounted on Western governments to prove cost effectiveness due to the changing nature of defence developments after the end of the Cold War and the need to avoid litigation for software that might be implicated in an accident. Yet the difficulties of applying formal methods in systems of any level of complexity and the need to trust the program code acted as a spur to maintain complex human centred software quality assurance procedures.

CONCLUSION: TRUSTING COMPUTERS

There is much evidence that we already *do* trust computers in many walks of life without formal proof or other formal demonstration, even to the extent of trusting safety-critical systems such as the 'fly by wire' software in the Boeing 777 airliner, two million lines of code which have not been fully proved (Lytz, 1995). Expert mathematicians have begun to accept computer generated proofs, albeit in qualified ways (Chang, 2004). As MacKenzie (2001, p. 301) argues, 'moral entrepreneurs' of computerised risk ensure that warnings about computerised risk are heeded so that safety-critical software is avoided and, where it is unavoidable, much care is taken over its development. Military standards, so detailed about the use of formal methods in software design and attendant work processes in the 1990s, have moved a decade later to be much less prescriptive about the work methods of ensuring software quality, thereby allowing for the crucial element of human inspection in order that the software may be trusted. As Collins (1990) notes, we are remarkably accommodating to computers, making sense of them and involving them in our social networks, and will continue to find imaginative ways of doing so. This echoes Nissenbaum's (1999) view that we may trust computers if we can treat them as 'agents.' We may meaningfully ascribe intentions and reasons to them.

In this article we have sought to tell a story of trust, in particular how software may be trusted when it is not produced by a human programmer. This involves consideration of a complex set of discourses including the question of mathematical proof and how proof is achieved within mathematical communities. We see a similar need to replicate such human processes of trust in trusting computer systems. We have argued that the making of standards to be applied within software quality assurance procedures shows ways in which mechanisms of trust are inscribed in software standards. Our case study, an action research project in a military aircraft company, demonstrates the difficulties which occur when quality assurance procedures involving code walkthroughs—procedures with built-in human trust mechanisms—are incommensurable with a system which relies on auto-code. The climate of defence research and spending was a major influence, both on our case study and the wider development of standards. There is a continued tension between needing to trust and trying to control: trusting the software and controlling its production. The story which we tell here is one of continuing human ingenuity in finding ways of trusting computer software.

REFERENCES

Akrich, M. (1992). The de-scription of technical objects. In W. E. Bijker & J. Law (Eds.), *Shaping technology/building society: Studies in sociotechnical change* (pp. 205-224). Cambridge, MA/London: MIT Press.

Baskerville, R. Investigating information systems with action research. *Communications of the Association for Information Systems, 19*(2). Retrieved October 5, 2006, from http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.htm

Baskerville, R., & Wood-Harper, A.T. (1999). A critical perspective on action research as a method for information systems research. *Journal of Information Technology, 11*, 235-246.

Chang, K. (2004, April 6). In math, computers don't lie. Or do they? *New York Times*. Retrieved October 5, 2006, from http://www.math.binghamton.edu/zaslav/Nytimes/+Science/+Math/sphere-packing.20040406.html

Collins, H.M. (1990). *Artificial experts: Social knowledge and intelligent machines*. Cambridge, MA: MIT Press.

Davies, B. (2006, October 3). Full proof? Let's trust it to the black box. *Times higher education supplement.*

De Millo, R.A., Lipton, R.J., & Perlis, A.J. (1977). Social processes and proofs of theorems and programs. In *Proceedings of the 4th ACM Symposium on Principles of Programming Language* (pp. 206-214).

DSDM Consortium. (2006). White papers. Retrieved October 5, 2006, from *http://www.dsdm.org/products/white_papers.asp*

Faulkner, W. (2000). The power and the pleasure? A research agenda for 'making gender stick.' *Science, Technology & Human Values, 25*(1), 87-119.

Forsythe, D.E. (2001). *Studying those who study as: An anthropologist in the world of artificial intelligence*. Stanford University Press.

Gillies, A.C. (1997). *Software quality: Theory and management* (2nd ed.). London/Boston: International Thomson Computer Press.

Hacker, S. (1989). *Pleasure, power and technology: Some tales of gender, engineering, and the cooperative workplace*. Boston: Unwin Hyman.

Ince, D. (1994). *An introduction to software quality assurance and its implementation*. London: McGraw-Hill.

Kuhn, T.S. (1962). *The structure of scientific revolutions***.** University of Chicago Press.

Latour, B., & Woolgar, S. (1979*). Laboratory life: The social construction of scientific facts*. Princeton University Press.

Lau, F. (1999). Toward a framework for action research in information systems studies. *Information Technology & People*, *12*(2), 148-175.

Low, J., & Woolgar, S. (1993). Managing the sociotechnical divide: Some aspects of the discursive structure of information systems development. In P. Quintas (Ed.), *Social dimensions of systems engineering: People, processes and software development* (pp. 34-59). New York/London: Ellis Horwood.

Lytz, R. (1995). Software metrics for the Boeing 777: A case study. *Software Quality Journal***,** *4*(1), 1-13.

MacKenzie, D.A. (2001). *Mechanizing proof: Computing, risk, and trust***.** Cambridge, MA/London: MIT Press.

MacKenzie, D.A. (2004). *Computers and the cultures of proving.* Paper presented at the Royal Society Discussion Meeting, London.

Ministry of Defence (MoD). (1997). Requirements for safety related software in defence equipment

Retrieved October 5, 2006, from http://www.dstan.mod.uk/data/00/056/01000300.pdf

Ministry of Defence (MoD). (2004). Interim defence standard 00-56. Retrieved October 5, 2006, from http://www.dstan.mod.uk/data/00/056/01000300.pdf

Myers, G.J. (1979). *The art of software testing.* New York: Wiley.

Myers, M.D., & Avison, D.E. (Eds). (2002). *Qualitative research in information systems: A reader.* London: Sage Publications.

Nissenbaum, H. (1999). Can trust be secured online? A theoretical perspective. *Etica e Politica*, *2*. Retrieved October 5, 2006, from http://www.units.it/~etica/1999_2/nissenbaum.html

Popper, K.R. (1963). *Conjectures and refutations.* New York: Harper.

Pressman, R. (2005*). Software engineering: A practitioner's approach* (6th ed.). London/New York: McGraw Hill.

Sillers, T.S., & Kleiner, B.H. (1997). Defence conversion: Surviving (and prospering) in the 1990s. *Work Study*, *46*(2), 45-48.

Singh, S. (1997). *Fermat's last theorem.* London: Fourth Estate.

Stahl, B.C. (2006). *Trust as fetish: A Critical theory perspective on research on trust in e-commerce.* Paper presented at the Information Communications and Society Symposium, University of York, UK.

Tierney, M. (1993). The evolution of Def Stan 00-55: A socio-history of a design standard for safety-critical software. In P. Quintas (Ed.), *Social dimensions of systems engineering: People, processes and software development* (pp. 111-143). New York/London: Ellis Horwood.

Trim, P. (2001). Public-private partnerships and the defence industry. *European Business Review*, *13*(4), 227-234.

Weinberg, G. (1971). *The psychology of computer programming.* New York: Van Nostrand Reinhold.

This work was previously published in International Journal of Technology and Human Interaction, Vol. 3, Issue 4, edited by B. Stahl, pp. 1-14, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.9
Access Control Specification in UML

Manuel Koch
Free University of Berlin, Germany

Francesco Parisi-Presicce
George Mason University, USA and University of Rome "La Sapienza", Italy

Karl Pauls
Free University of Berlin, Germany

ABSTRACT

Security requirements have become an integral part of most modern soft¬ware systems. In order to produce secure systems, it is necessary to provide soft¬ware engineers with the appropriate systematic support. This chapter discusses a methodology to integrate the specification of access control policies into UML. The methodology, along with the graph-based formal semantics for the UML access control specification, allows to reason about the coherence of the access control spec¬ification. The chapter also presents a procedure to modify policy rules to guarantee the satisfaction of constraints, and shows how to generate access control requirements from UML diagrams. The main concepts in the UML access control specification are illustrated with an example access control model for distributed object systems.

INTRODUCTION

Security requirements are an important aspect in the development of software systems that are not used in completely trusted environments. In order to increase the overall system security and to better satisfy security constraints, security policies should be specified in terms of security models that are integrated with the general software engineering models. Modelling techniques that are well known to software engineers should be used, to prevent them from specifying mistakes in security

specifications due to inadequate expertise of the particular specification technique. The reasoning about and the verification of security properties require a formal semantics for the specification technique.

Since the UML is nowadays the de-facto standard modelling language, the usage of UML for the specification of security aspects is particularly attractive, since software engineers are used to the UML notation and the accompanying tools (Brose, Koch, & Lohr, 2002; Devanbu & Stubblebine, 2000; Epstein & Sandhu, 1999; Jurjens, 2001; Lodderstedt, Basin, & Doser, 2002). We identify in this chapter the parts necessary to specify an access control policy and propose a UML specification for these parts. In our framework, we use only existing UML model elements and extension mechanisms to ensure compatibility with UML tools that can then be directly used for the access control specification. The UML specification of an access control model makes use of UML class diagrams, object diagrams, some additional stereotypes and OCL constraints (OMG, 2003).

Access control constraints restrict an access control model and prevent the system from reaching unwanted protection states. The expression and specification of access control constraints is a difficult task and existing languages are often too complex for administrators to determine whether a set of constraints really satisfies the requirement. Therefore, access control languages have been proposed which have a manageable complexity and are understandable by administrators but still expressive enough to capture most practical access control constraints (Ahn & Sandhu, 2001; Jaeger & Tidswell, 2001; Koch, Mancini, & Parisi-Presicce, 2002c). We present in this chapter an approach to specify access control constraints in UML. We consider OCL constraints as a textual presentation of access control constraints, but present also visual constraint specification by UML object diagrams (Koch et al., 2002c; Ray, Li, France, & Kim, 2004).

In our approach, access control models are specified on several levels, depending on the added application domain information similar to the UML meta-modelling architecture (without the metametamodel). On the metamodel level, there is no application domain information and the access control specification shows only the policy rules and constraints. This is the level at which the generic access control model, such as Role-Based, Mandatory, Discretionary, are described. The model level is an instance of the meta access control model and refines it using application domain information. At this level, the specific roles used in RBAC or the clearance, and categories used to construct the labels in MAC are defined depending on the application. The last level is an instance of the access control model in a specific information domain. This is the level at which principals are assigned to roles or labels, or specific permissions are assigned to roles.

To verify that an access control policy satisfies all the access control constraints (i.e., the policy permits only system states that satisfy all the constraints) an appropriate formal semantics for the UML access control model is needed. We give a graph-based formal semantics (Rozenberg, 1997) to the UML access control specification by a transformation of UML class and object diagrams, respectively, into graphs and graph rules. The graph-based semantics enables us to use several verification concepts to verify the constraints with respect to the access control model (Koch, Mancini, & Parisi-Presicce, 2001; Koch, Mancini, & Parisi-Presicce, 2002b).

To know how to model access control requirements into UML is only part of the problem. Another essential part is to determine the access control requirements themselves. We present an approach to generate automatically access control requirements from UML use case, class, and sequence diagrams. The extracted access control information is the basis for the UML access control model.

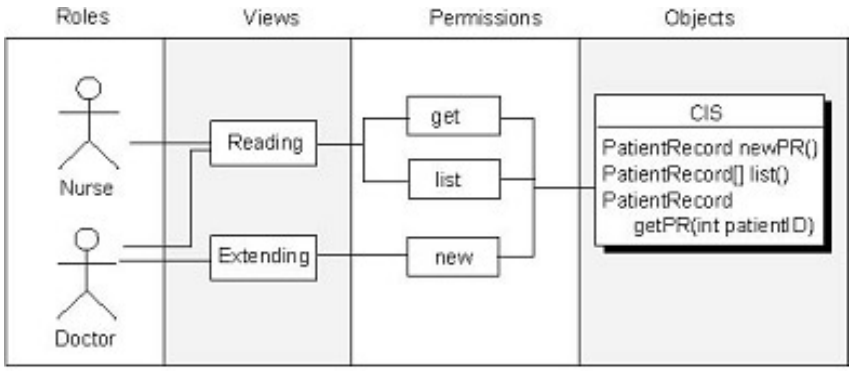
The remainder of the chapter is organized as follows: the next section briefly discusses access control policies and introduces View Based Access Control as the access control model used in this chapter as the running example. The following section gives an overview of our approach, introducing the three specification levels of access control models. The two subsequent sections present the UML specification of the access control metamodel and its refinement to an application domain specific access control model. Our example application domain is a small hospital information system. The model is then further refined with the specification of access control constraints in UML. The section on Verification briefly mentions graph transformations as an effective way to give a formal semantics for an UML access control specification. The next section illustrates the generation of access control requirements from UML diagrams, whereas the penultimate section provides a brief discussion of other approaches and of related work. The chapter closes with a summary of its contents and pointers and suggestions for open problems.

ACCESS CONTROL IN OBJECT-ORIENTED SYSTEMS

Access control is the typical approach to limit access to resources in general, and to objects in particular. The restrictions include preventing principals from reading the contents of a file to protect the confidentiality of the data, or from modifying entries in a database to maintain the integrity of the data and the trustworthiness of the information contained. Access control policies describe which states of the system are deemed acceptable (safe), in which authorized principals can access entities in the appropriate manner (e.g., in read mode for confidential data, in write mode for trusted data), and security mechanisms are designed to implement the security policies.

View-based access control (VBAC) is an access control model specifically defined to support the design and management of access control policies in object-oriented systems (Brose 2000; Brose 2001a). VBAC relies on roles as abstractions of callers and can be regarded as an extension of *role-based access control* (RBAC) (Sandhu, Coyne, & Feinstein, & Youman, 1996) to distributed object systems. Rolebased access control reduces the complexity and cost of security administration in large systems because roles serve as a link between access modes for

Figure 1. VBAC example



objects (e.g., read or write access if the object is a file, the print command for a printer, a method of a distributed object etc.) and subjects (users or processes that run on behalf of them). A subject can access an object if it has been authorized for, and has activated, a role that has been assigned the required permissions for the access.

The principal new feature of VBAC is that of a view for the description of fine-grained access rights, which are permissions for calling operations of objects. Views on objects are assigned to roles, and a subject can call an operation of an object if it has a view on the object with a permission for the operation. The subject has no access to the operation if the operation is not in a view assigned to one of the subject's roles.

Figure 1 illustrates a small access control example for CIS objects providing the operations newPR(), list() and getPR() (the meaning of these objects will be explained in more detail in a later section). The CIS represents a hospital central information system used to manage the records of patients.

The permissions get, list and new give the access right to call the operations *getPR()* (returns the patient record of a patient for a given ID), *list()* (lists all patient records) and *newPR()* (creates a new patient record). There are two views in this example: the view Reading contains the permissions get and list, and the view Extending has the permission new. The view Reading is assigned to both the role *Nurse* and the role *Doctor*, while the view Extending only to the role *Doctor*. Therefore, any user in the role *Doctor* can call any operation of CIS objects, while a user in the role *Nurse* has access to read the patient records, but cannot create a new one.

VBAC access policies are deployed (via descriptor files) together with applications in the target environments, as in approaches like EJB (Sun Microsystems, 2000) or the CORBA Component Model (OMG, 1999). These policies are initially designed by developers and subsequently used and possibly adapted by deployers and security managers. The deployment and management infrastructure designed for this approach is called Raccoon (Brose 2001a; Brose, 2001b). A deployment tool processes policy descriptors, and stores static view and role definitions in repositories that can be managed using graphical management tools. At runtime, role membership is represented by digital certificates, issued by a role server. Access decisions are made locally in the server processes that host application objects, based on policy information supplied by the policy servers, which rely on the deployed policy information. Policies can be managed in terms of roles and views using graphical management tools. A more detailed presentation of the policy enforcement infrastructure can be found in (Brose 2001a; Brose 2001b; Brose 2002).

OUTLINE OF OUR APPROACH

An access control policy consists of a set of policy rules that define the choices in the individual and collective behaviour of the entities in the system. To specify an access control policy, it is necessary to model the entities in the system, the policy rules for the behaviour of these entities and (possibly) additional constraints. We model all these parts of an access control policy using UML in order to integrate them easily into the UML software process. The entities on which the policy operates are specified in a UML class diagram, the policy rules are specified by specially annotated object diagrams, and the access control constraints are specified either textually by Object Constraint Language (OCL) constraints or visually by object diagrams. Therefore, a *UML access control specification* ACS = (*T, PRules, Constr*) consists of the following UML diagrams:

- T , called *type diagram*, is a class diagram that specifies the available entities,

- *PRules* is a set of object diagrams for the specification of the policy rules,
- *Constr* is a set of object diagrams or OCL constraints to model Access Control constraints.

In general, an access control model is independent of an application. For example, the RBAC model describes a general concept of how to manage access on resources that can be applied to many application domains. Therefore, we use a threelayer model structure consisting of an access control metamodel (ACMM), an access control model (ACM) and an access control instance model (ACIM). Each of these models (ACMM, ACM, and ACIM) consists of a type diagram and object diagrams for policy rules and constraints, but describes the access control policy on a different abstraction level. The access control metamodel defines the language for specifying access control models such as VBAC, RBAC, Discretionary or Mandatory Access Control models. The ACMM is independent of any information or application domain. The access control model is an instance of the access control metamodel and introduces the information and application domains. If the metamodel specifies an RBAC model, doctor and nurse would be roles specific to a medical information domain. The instance access control model is an instance of the access control model in a specific information or application domain. Examples would be orthopaedist, surgeon, internist, etc., as doctor role instance objects. The following table summarizes the three layers.

Figure 2 shows the relations between the access control diagrams on the different layers.

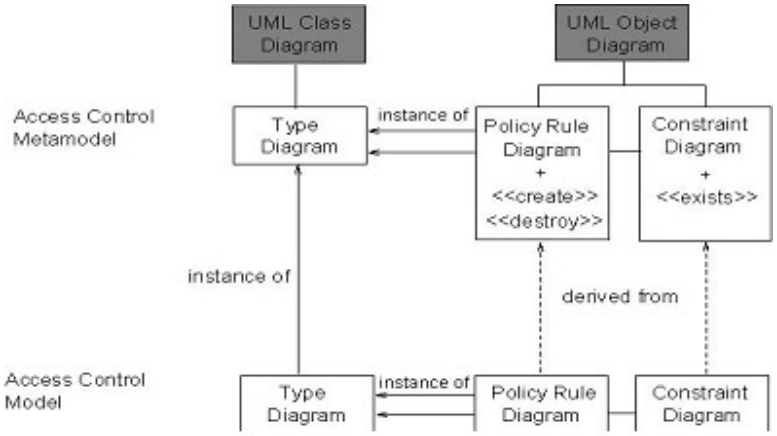
We focus here on the access control metamodel and the access control model and introduce, in the next sections, their specification for the example of the VBAC model described in the previous section. The application domain for the access control model is a medical information system.

The three-layer model structure gives a method to specify access control policies in UML, but it leaves open the question of what a system designer has to specify in a concrete access control policy while developing a concrete application since, so far, there is no support for access control requirement analysis. In the last section, we briefly present a model-driven approach to access control specification which supports a system designer in developing an access control model for a given application domain.

Table 1.

Layer	Description	Example
Access control metamodel (ACMM)	Access control model independent of an information/application domain	VBAC, RBAC, etc.
Access control model (ACM)	Access control model regarding an information/application domain	Nurses, Doctor, etc. for roles in RBAC
Access control instance model (ACIM)	Instance of the access control model in a specific information/application domain	surgeon, internist, etc. for role Doctor

Figure 2. Diagram overview



MODELLING ACCESS CONTROL ENTITIES

This section concerns the modelling of access control entities in UML on both the metamodel and the model layer. The access control entities are, for example, the objects on which access must be controlled, the subjects, which try to access the objects and the permissions that allow or deny access. In the Access Control List implementation, known from Unix operating systems, the entities are mainly users, groups, processes, and files/directories with their read, write and execute permissions. In a Role based access control model, the main entities are the roles and their assigned permissions, the users, the sessions and the objects. In the sequel, the VBAC model serves as the example for the access control model, a hospital application as the application domain for the application model.

Access Control Metamodel

The access control entities are modeled in a class diagram, which we call a type diagram. The associations between the class diagram elements specify the relations between the access control entities. Therefore, the class diagram constrains already all the access control relations, since a relation that does not occur in the type diagram is forbidden. Figure 3 shows the specification of the entities of the VBAC model by a type diagram. The intended meaning of the access control entities is given next.

In a concrete application, the classes in the type diagram are specialized to classes taken from the application class diagram. For example, the role class in the VBAC type diagram must be specialized to the concrete roles of the given application (e.g., to the roles Doctor or Nurse of Figure 1).

Application-Specific Access Control Model

The access control metamodel defines a language for specifying access control models and is independent of an information domain. The access control metamodel for VBAC defined above has, as metaobjects, Subject, Role, View, Permission, and Object. In an information domain, these metaobjects are mapped to application dependent objects.

The access control entities in the metamodel are specified in a class diagram. The access control

Table 2.

Object:	Used to represent the distributed objects in the system to which access must be controlled by a policy. Objects can be related (e.g., an object can be a sub-object of another object).
Subject:	Used to represent the system users and the processes that run on behalf of the users. Subjects may have references to objects to access them.
Permission:	A permission specifies a right to access an object, and is uniquely assigned to that object. An object, however, can be associated to several permissions.
View:	A view groups a number of permissions belonging to the same object (no view can contain permissions defined for different objects).
Role:	Used to represent the access control roles, which are assigned to subjects. A role can be played by several subjects, and a subject can play several roles. A view can be assigned to different roles and a role may have several views. Roles can be related by an inheritance relation, where the extended role inherits all the views of the base role.

Figure 3. The VBAC model

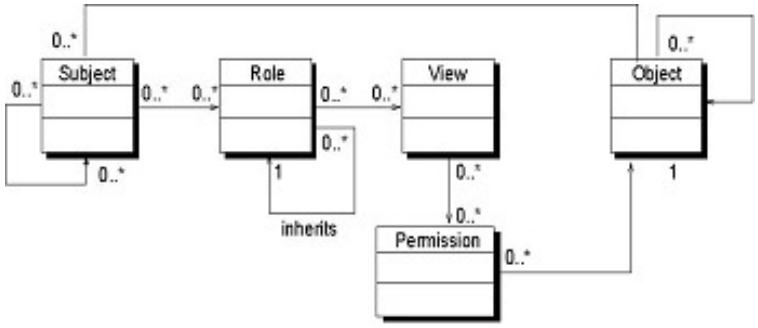


Figure 4. Class diagram for the application

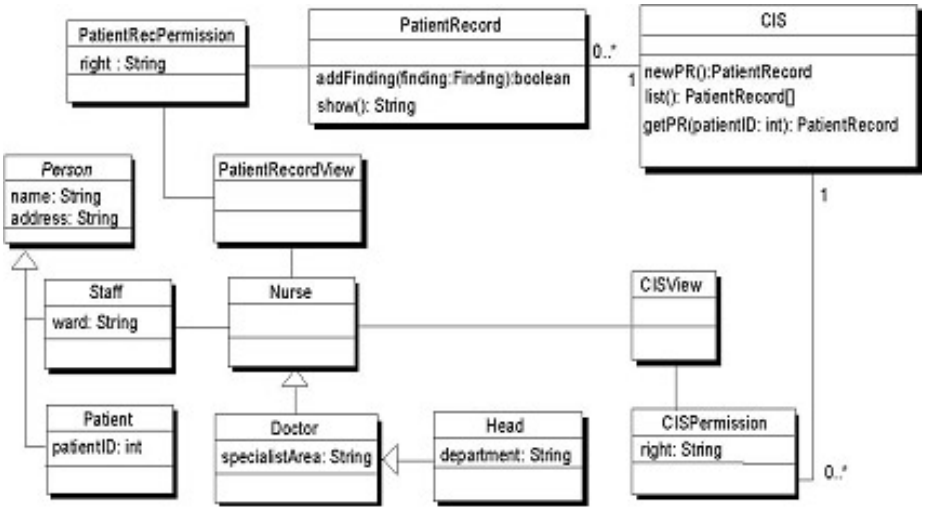


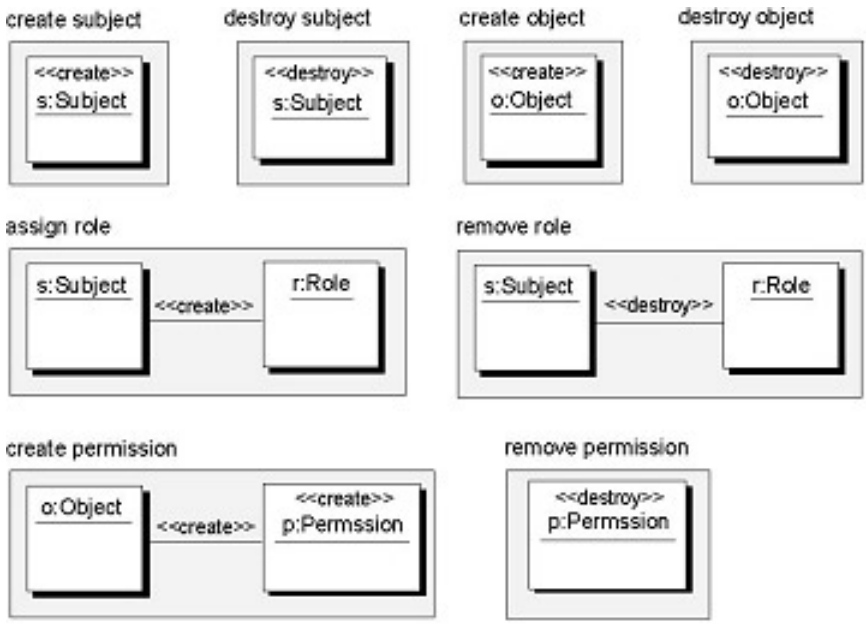
Table 3.

Metamodel	Model	Description
Subject	Person	A person has a name and an address. The class is abstract.
	Staff	Specialization of Person. Specifies the staff in the hospital. The attribute ward specifies on which ward the staff member is currently working.
	Patient	Specialization of Person. Patients are identified by a patient identifier patientID.
Role	Nurse	Role for nurses.
	Doctor	A specialization of the role nurse from which it inherits the permissions. The attribute specialistArea, e.g. surgery or orthopaedy, describes the doctor's specialization.
	Head	Specialization of the role Doctor, assigned to the department head of a ward.
Metamodel	**Model**	**Description**
View	CISView PatientRecord View	View for CIS permissions. Views for PatientRecord permissions.
Objects	CIS Patient-Record	The central information system (CIS) contains the electronic patient records. Patient records can be created by the operation newPR() if it is the patient's first visit in the hospital. The list of all the patient records can be requested by listPR(), a specific patient record is returned by getPR(). The (textual) contents of a patient record can be requested with the operation show(). Whenever findings for a patient are created due to new examinations, the findings are added to the patient record with the operation addFinding().
Permission	CIS-Permission Patient-Record Permission	Permissions for object CIS. The attribute right contains the name of the operation, to which the permissions grant access. Permissions for the object PatientRecord.

entities in the access control model are also specified in a class diagram, which is an instance of the metamodel class diagram and which contains information and application domain specific access control entities.

Figure 4 shows an instance of the VBAC metamodel applied to a medical information domain. We consider a small part of a hospital in which patients are medicated by doctors and nurses. Each ward has a department head. The following table shows the mapping of the metaobjects to the objects in the access control model with a brief description of their meaning. We keep the hospital application small, but detailed enough to demonstrate the specification of the VBAC model. A more detailed discussion to the hospital application can be found in Brose, Koch, and Lohr (2003).

Figure 5. VBAC policy rules



MODELLING ACCESS CONTROL POLICY RULES

An access control protection state is generally not fixed, but changes during the system lifetime. Possible causes for changes in the protection state are the addition or deletion of users or roles, the modification of permissions associated with roles, changes in access requirements of objects, etc. Access control policy rules define the permitted state changes and therefore Control the behavior of the system.

Access Control Metamodel

Policy rules are specified in object diagrams using the stereotypes <<create>> and <<destroy>>. The intended meaning of an object or link with a stereotype <<create>> is that the object or link is created by the system. The intended meaning of an object or link with stereotype <<destroy>> is that the object or link is removed from the system. Our approach in representing the actions of a policy rule is similar to the representation of post-conditions for actions in the Catalysis approach (D'Souza & Wills 1998). To visualize an action in Catalysis, one diagram is used to present the state before and after the action. Catalysis, however, uses a presentation of the "after" state by bold elements or different colors instead of stereotypes.

Figure 5 (role management) and Figure 6 (view management) show the policy rules for the VBAC model. Their intended meaning is described next.

Access Control Model Rules

The policy rules of the access control model are derived from the object diagrams for the policy rules of the metamodel. Metaobjects are instantiated with information domain specific objects specified in the type diagram of the access control model. The object diagram obtained by instantiating a policy rule must be an instance of the type diagram of the access control model.

Table 4.

create subject:	The object diagram for the policy rule create subject consists of one subject instance with the stereotype <<create>> which specifies that the subject instance is created by the rule. The creation of new subjects is unconditional and does not depend on the current system state (no context conditions)
destroy subject:	The object diagram for the rule destroy subject is similar to the create-subject but with stereotype <<destroy>>.
create object:	Object instances can be created with a rule similar to the one for subjects.
destroy object:	Object instances can be removed from the system with a rule similar to the one for subjects.
assign role:	The rule assign role specifies the assignment of a subject to a role, modeled by a link between the subject instance and the role instance. The link is created (carries stereotype <<create>>) while the subject and the role must already exist (no <<create>> stereotype).
remove role:	The rule remove role specifies the removal of the assignment of a subject to a role by deleting the link between the role and the subject. Both the subject and the role instance remain in the system, since they do not carry a <<destroy>> stereotype.
create permission:	The rule create permission creates a new permission instance which is immediately connected to an object, since a permission must always belong to a unique object.
remove permission	The rule remove permission specifies the removal of a permission
assign view:	The assignment of a view instance to a role instance is specified by a link created by the rule assign view between an existing role and an existing view
remove view:	The rule remove view specifies the removal of a view from a role by destroying the link between the role and the view
create view	View instances can be created analogously to subjects.
destroy view:	View instances can be removed analogously to subjects.
extend view:	The assignment of a permission to a view instance is specified by a link inserted by the rule extend view between the view and the permission instance
restrict view:	The rule restrict view removes a permission from a view by removing the connecting link

Figure 6. VBAC policy rules for view management

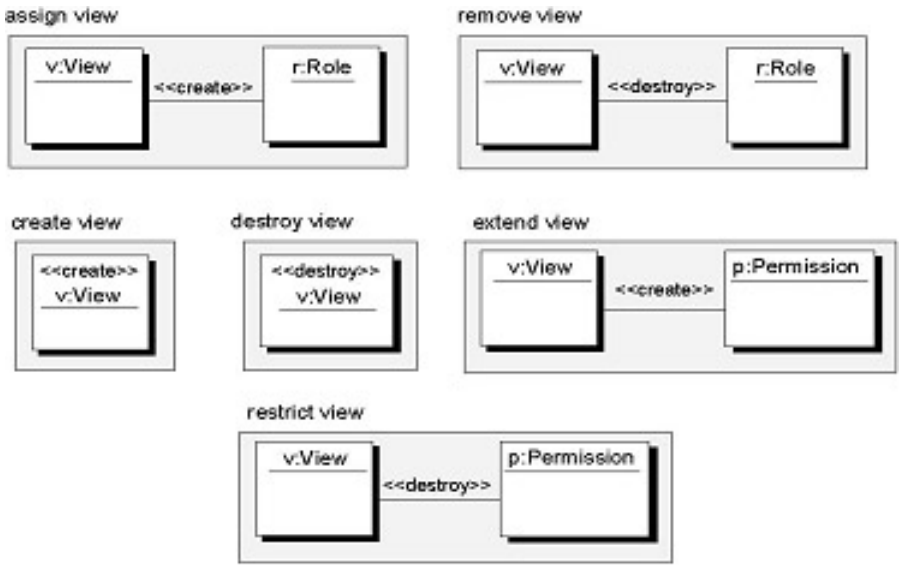
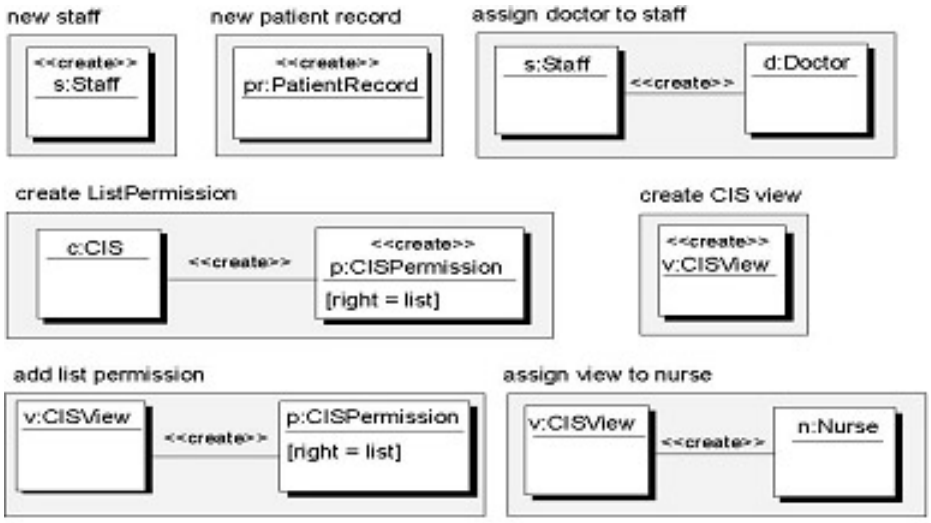


Figure 7. Policy rules in the access control model



For example, the rule assign role in Figure 5 can be specialized to a rule in which the metaobject s:Subject is instantiated by the object s:Staff and the metaobject r:Role by d:Doctor. Another example is the metarule create permission. The metaobject o:Object can be instantiated by a CIS object, the permission object p:Permission by a CISPermission with a value list for the attribute right. Figure 7 shows this policy rule and other examples.

Not all instantiations are valid. The following are examples of invalid instantiations:

- The instantiation of the metaobject s:Subject to p:Person is not valid since the class Person is abstract (i.e., it has no instances).

- The instantiation of the metaobject s:Subject to p:Patient and the metaobject r:Role to d: Doctor in rule assign role is not acceptable since the type diagram of the access control model does not permit links between patient and doctor objects.

The policy rules of the access control model are given by all the object diagrams that can be derived from the policy rules of the metamodel, where the objects are taken from the access control model type diagram, which are instances of the type diagram.

MODELLING ACCESS CONTROL CONSTRAINTS

The access control policy rules determine the acceptable system states. In some situations, however, it is necessary to reduce the system states generated by the policy rules. For example, the policy rule assign role of the VBAC model can be used to assign any number of subjects to the same role (by repeated application to the same rule). In many applications, however, there are role cardinality constraints that limit the number of subjects in a role. Some cardinality constraints are already included in the access control type diagram of a UML AC specification (e.g., Figure 3 requires a unique object for each permission). But a requirement for at least one permission for a view assigned to a role is a prerequisite constraint not expressible by cardinalities. Therefore, it must be specified separately.

Possible representatives of constraints for our running example are:

1. In VBAC, if the view is assigned to a role, then this view must have at least one permission; otherwise permissionless views are allowed.
2. In the hospital application, there can be at most one department head for each ward.

The first constraint concerns the access control metamodel since it is a constraint that must be satisfied by all VBAC policies in any application domain. The second constraint concerns a special application domain (here a hospital domain) and is specified in the access control model. Therefore, access control constraints are specified on both the metamodel and the model levels. Before we consider the difference between these two levels, however, we discuss some ways to specify access control constraints in UML present in the literature, namely the Object Constraint Language (Ahn & Shin, 2001) as a representative of a textual constraint language and two different approaches based on object diagrams (Koch et al., 2002c; Ray et al., 2004) as representatives of visual constraint languages.

Textual Specification of Access Control Constraints by OCL

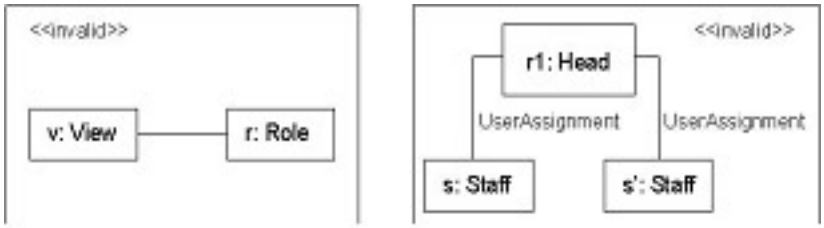
The object constraint language (OCL) (OMG 2003), a part of the UML specification, is a powerful specification language, based on first-order Logic, in which complex constraints can be textually expressed and attached to the graphical UML diagrams.

By using the OCL, the first of the two example constraints previously can be described with the following OCL expressions:

```
context View inv
(self.role->notEmpty) implies (self.permission-
>notEmpty)
```

This OCL constraint specifies that if a view is related to a role, then there is at least one permission related to this view. Note that the constraint does not place restrictions on views unrelated to roles. The second constraint is represented in OCL as:

Figure 8. Object diagram patterns




```
context Head inv self.staff->size() < 2
```

This OCL constraint forbids more than one person from the staff assigned to the application domain specific role Head.

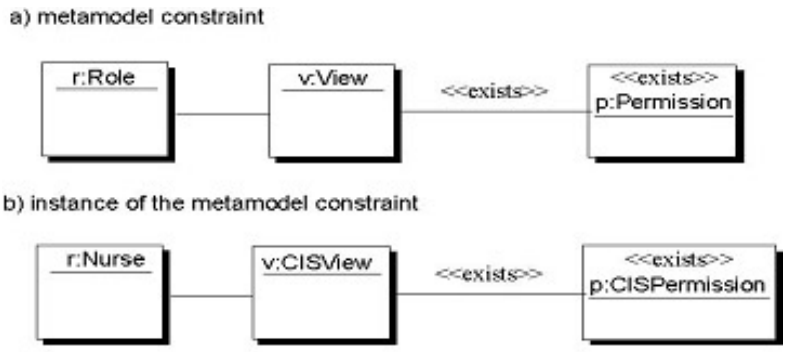
Visual Specification of Access Control Constraints

Visual specification languages to specify access control constraints in UML start from the observation that OCL is difficult to understand and does not integrate well with the remaining visual UML diagrams. Therefore, Ray et al. (2004) present a visualization of access control constraints using UML object diagrams. In their approach, object diagram templates describe object structure patterns that are violations of access control constraints. These patterns are used to check for violations of a constraint, as such an object diagram pattern represents a system structure which must not occur in the system state.

Figure 8 shows the object diagram for the two example constraints. On the left-hand side, the object diagram for the prerequisite constraint is specified. It shows the invalid state in which a view is assigned to a role, but there is no permission assigned to the view. On the right-hand side, the cardinality constraint is specified by the invalid state of two staff objects assigned to the role Head object.

The visualization of access control constraints presented in Koch et al. (2002c) also uses object diagrams. Similar to Ray et al. (2004) object diagrams are used to specify invalid system states which must not occur in a system state for the

Figure 9. A metamodel constraint and an instance of this metamodel constraint in the hospital access control model



constraint to be satisfied. The difference between the approach of Koch et al. (2002c) and that of Ray et al. (2004) are additional object diagrams for required states and a formal semantics based on graph transformations (Rozenberg, 1997) which can be used to apply verification concepts on access control constraints. The semantics will be explained in more detail in the next section.

An object diagram for a required state specifies an object structure that must occur in a system state. A system state satisfies such a constraint if the required object structure occurs as a substate. The existence of this object structure may be conditional, i.e., the object structure is required only in certain substates. The object diagrams for required states are called positive constraints and use the stereotype <<exists>> in the following way: all objects and links labeled with this stereotype must exist in a system state when the remaining (i.e., the non <<exists>>labeled) objects and links occur, as well.

Figure 9 shows the visual specification of the viewpermission requirement by a positive constraint. It shows the assignment of a permission p to a view v, which in turn is assigned to a role r. The permission object p and its link to the view v carry the <<exists>> stereotype. Both must exist whenever the view v is assigned to a role r. If no role is assigned to the view, the assigned permission need not exist. The object diagram for the invalid system state of two users in role Head is the same as the one shown in Figure 8.

Access Control Metamodel and Access Control Model

As previously mentioned, constraints may refer to, and be specified for, the metamodel level if they must be satisfied by all access control models independently of a specific application, or they may concern only a specific application and therefore must be valid only in a specific access control model.

Since metamodel constraints must be valid in any access control model, they must be part of a given access control model for a certain application, as well. Therefore, access control model constraints are derived from the access control constraints of the metamodel, as is the case for the policy rules: The metaobjects in the metamodel constraint are instantiated with objects given in the access control model type diagram, so that the resulting object diagram is a valid instance of the ACM type diagram. An example is the specialisation of the VBAC metamodel constraint in Figure 9 in which the metaobject role is specialized to a Nurse object, the metaobject view is specialized to a CISView object and the permission is a CIS-Permissiom. An invalid instantiation, according to the ACM type diagram, would be a CISView object for the view and a PatientRecPermission object for the permission, since it would be in conflict with the type diagram.

The access control constraints of an access control model, therefore, consist of a) all the object diagrams obtained by instantiating the metamodel constraint diagrams using objects specified in the access control model (ACM) type diagram, so that the resulting object diagrams are instances of the ACM type diagram and b) a set of object diagrams for information domain specific constraints.

VERIFICATION OF ACCESS CONTROL CONSTRAINTS

The access control constraints restrict the possible system states that can be produced by the policy rules. Whether a set of object diagrams for the policy rules of a UML access control specification satisfies all the access control constraints should be checked automatically as much as possible. To have an automatic verification, it is necessary to have a formal semantics for both the UML access control model and the UML constraints, on which an automatic verification support can be built.

Consider, as an example, the policy rules for the access control metamodel shown in Figure 5 and Figure 6. These object diagrams are specialized to application specific objects given in the access control model type diagram in Figure 4. One possible specialization is the instantiation of the policy rule assign role in Figure 5 with a Staff instance as subject and Head as role. Applying this rule several times to the same Head role but with different staff member instances, would produce several staffers who play the role Head for the same department. The negative constraint in Figure 8 on the right-hand side, however, forbids more than one subject in the role Head. Therefore, the UML access control specification is not correct since the policy rules can construct a system state that does not satisfy all the access control constraints.

We have developed (Koch et al., 2001) a graph-based, formal semantics for a UML access control specification which allows us to check statically if the set of object diagrams for the policy rules satisfy the constraints (Koch et al., 2002a; Koch et al., 2002c). To use these verification concepts we need to transform UML diagrams into graphs, UML object diagrams for policy rules into graph transformation rules and (visual) UML constraints into graphical constraints (for transformation details see (Koch & Parisi-Presicce 2002d). Due to the graphical notation of UML diagrams, the transformation of these diagrams into graphs is natural. Access control constraints are transformed into graphical constraints. The transformation of visually specified UML constraints can be done in a natural way. The transformation of OCL constraints is more difficult, but possible for a part of the OCL constraints. These are mainly those OCL constraints used to express properties on the object structure of the system state. In Bottoni, Koch, Parisi-Presicce, & Taentzer (2001), the interested reader can find a more detailed presentation on the visualization of (most) OCL constraints

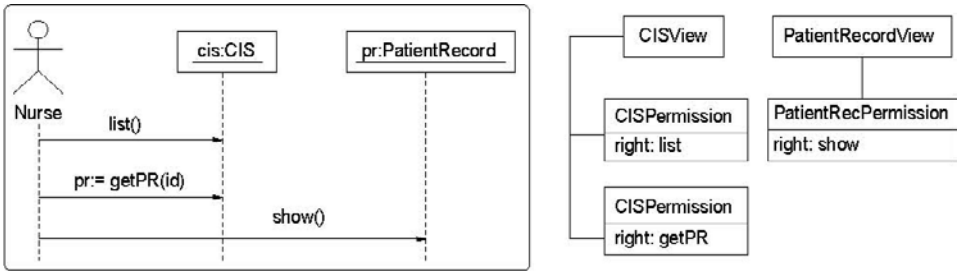
The checking algorithm (Koch et al., 2002c) determines, for each policy rule and access control constraint, whether the policy rule can construct a system state that violates the constraint. In this case, the algorithm modifies the rule by adding an additional condition to the rule, to ensures that the policy rule, whenever it is applied to a valid system state, will always construct a new system state which again satisfies the constraint. The policy rule condition forbids the application of the rule to system states that would lead to an invalid new system state. If we consider the example, mentioned above, of the policy rule assign role and the negative constraint which forbids two or more staffer in the role Head, then the checking algorithm adds a condition to the policy rule assign role to prevent its application to the Head role if there is already a staffer assigned to this role.

We have built a prototype of an RBAC administration tool, which implements the checking algorithm as well. The tool has been implemented according to the proposed NIST standard for RBAC (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001) and is based on the general graph transformation engine AGG (Taentzer, Ermel, & Rudolf, 1999). It allows a security administrator to define graph-based policy rules and constraints, including several kinds of authorization constraints identified in the literature (Ahn & Sandhu, 2001) such as various types of separation of duty (SoD) properties (simple static SoD, simple dynamic SoD, sessionbased dynamic SoD, and objectbased static SoD), cardinality constraints, and prerequisite roles and permissions.

ACCESS CONTROL REQUIREMENT ANALYSIS

So far, we have described how to model an access control policy in UML. We use a metamodel for the access control model and refine this metamodel to a model that considers the application specific access control requirements. To determine the ac-

Figure 10. A sequence diagram



cess control requirements (i.e., what to model) is a difficult task, especially since system designer are usually not security experts. Therefore, the designer should be supported in the software development process to obtain the necessary access control requirements.

In Koch and Pauls (2005a) and Koch and Pauls (2005b), we have presented a model-driven approach to derive access control requirements from the UML diagrams for the functional system analysis and design. For example, access control roles are based on the actors in the use case diagrams, access rights are based on the class diagrams and the access control permissions are based on the operation calls in the sequence diagrams. The access control information can be automatically extracted from these UML diagrams and is presented in UML diagrams for access control as shown in the previous sections.

We present next the generation of views from UML sequence diagrams as an example of this generation process. For a more detailed presentation of the model-driven approach we refer to Koch et al. (2005a) and Koch and Pauls (2005b). Sequence diagrams contain inherent access information by specifying the access required of actors to call the operations. The sequence diagram in Figure 10 shows the accesses required of a nurse to read a patient record. The nurse role must be able to call the operations list() and getPR() on the class CIS to retrieve the list of all available patient records and a single patient record, respectively. To read the patient records, the nurse role needs the permission to call the operation show() on the class PatientRecord.

The algorithm for the generation of access control views from a sequence diagram considers all the objects in the sequence diagram on which operations are called: For each of these objects, a view on the class of the object is generated which contains the permissions to call the operations on the objects of this class. In our example sequence diagram in Figure 10 we generate two views: one on the class CIS with CISPermissions to call the operations list() and getPR() and one view on the class PatientRecord with PatientRecPermission to call the operation show(). The views for all the remaining sequence diagrams are generated in a similar way.

The generated views are usually incomplete in the sense that they do not give a complete access control specification. This is due to the fact that sequence diagrams show only scenarios the designer is interested in. On the other hand, the views may also be redundant in the sense that the same view may be generated from different sequence diagrams. Therefore, the designer uses the generated views as a starting point to be refined into the final access control specification.

RELATED WORK

A few authors have addressed the problem of integrating security requirements into UML.

Jurjens presents the integration of security into UML in Jurjens (2005), where he shows how to model several security aspects by UML model elements as, for example, stereotypes or tagged values. His approach is more general than ours since it is not restricted to access control but considers, for example, also security protocols. Furthermore, Jurjens considers a wider variety of UML diagrams (e.g., also deployment diagrams) since his concern is with the integration of security into the UML in general. Our approach is focused on access control, and the generation and enforcement of access control policies in distributed systems, and, therefore, we only use the UML diagrams necessary to meet our objective. In contrast to our approach and the one by Basin, Doser, and Lodderstedt (2004), Jurjens does not consider a model-driven approach and does not provide tool support or an infrastructure to generate security policies from UML models or to enforce security policies.

Moreover, Jurjens (2002) has extended UML for specifying aspects of multilevel secure systems and security protocols, where he proposes a tailored formal semantics to evaluate UML diagrams and to indicate possible weaknesses. Unlike our approach, there is no model-driven integration of security and the verification does not provide an automatic consistency construction to resolve unsatisfied security constraints.

Fernandez-Medina, Martinez, Medina, and Piattini (2002) extend UML to support the design of secure databases. They consider a multilevel database model in which all model elements have a security level and a security role, both modeled by tagged values. They extend the OCL to be able to represent multilevel system constraints. Unlike the approach in Fernandez-Medina et al. (2002) which is developed for multilevel databases, our approach is more general, dealing with any access control model and is not restricted to databases. In contrast to Fernandez-Medina et al. (2002), which extends the OCL by nonstandard features, we use standard OCL for the specification of constraints. The OCL extension has the advantage of a clearer and more legible specification in the particular context of multilevel databases, but does not ensure compatibility.

Epstein and Sandhu (1999) introduce a UML-based notion for RBAC, but this notion is not suitable for the verification of security properties and cannot easily be used for the generation of access control specifications.

An approach closely related to ours is SecureUML (Lodderstedt et al., 2002). SecureUML is a UML-based modelling language for the model-driven development of secure systems. It provides support for specifying constraints, as well. The security information integrated in the UML models is used to generate access control infrastructures. In contrast to our approach, SecureUML focuses on static design models, which are closely related to implementations. Therefore, there is no support for detecting security requirements (e.g., which roles are needed and which permissions they need). Unlike our approach, which is suitable for arbitrary access control models, SecureUML is based on RBAC, and does not have a formal semantics to verify security properties.

Work related to our model-driven development approach to security engineering is presented by Basin et al. (2004), to describe a model driven approach, called SecureUML, to develop role based access control policies for J2EE applications. A formal basis allows the designer to reason about access control properties, and tool support is given by an integration of SecureUML into the ArcStyler tool (Interactive-Objects, 2005). In contrast to our approach, however, the analysis stage of the software process is not considered and the process starts with the design models. The role based model in (Basin et al., 2004) is more coarse-grained compared with the view based access control model here, in which access rights are fine-grained on the level of operations and

operation parameters. Basin et al. use the J2EE infrastructure to enforce the generated policies, but their approach makes it difficult to modify a policy dynamically: the modified policy must be changed manually in the XML-based EJB descriptor files. Dynamic changes of the access control policy, however, are an integral part of our approach. On the one hand, the concept of schemas considers the changing of rights already in the model; on the other hand, our proposed infrastructure supports policy changes at runtime, immediately enforced without interrupting the application and the system.

CONCLUSION

This chapter has presented an approach to the specification of Access Control policies in UML by means of UML class and object diagrams that can be modelled with existing UML tools. It has been shown how the framework models the access control entities, the rules of the access control policy, and the access control constraints. By using a layered model structure, we have been able to model the different components at an abstract (metamodel) level that is application independent, and at a concrete (model) level that includes application specific information. A translation of the UML Access Control specification into a graph-based security framework permits the application of verification concepts from graph transformations to reason about the coherence of a UML Access Control specification. In particular, it allows the modification of the context of application of existing rules to ensure that the new state satisfies all the constraints (if the old one did). The chapter closes with an indication of how to extract access control requirements from UML diagrams.

The UML Access Control specification can be modelled using existing UML CASE tools. One possible direction of future work is the transformation of the XMI export (obtained from the CASE tool) into a XML format for graphs (Taentzer, 2001). Then, graph transformation tools (Ehrig, Engels, Kreowski, & Rozenberg, 1999) can be used for the automatic verification of Access Control requirements.

REFERENCES

Ahn, G. J., & Sandhu, R. (2001). RoleBased authorization constraints specification. *ACM Transactions on Information and System Security, 3*(4), 207-226.

Ahn, G. J., & Shin, M. (2001). Rolebased authorization constraints specification using object constraint language. *Proceedings of the WET-ICE'01* (pp. 157-162).

Basin, D., Doser, J., & Lodderstedt, T. (2004). Model driven security. In M. Broy, J. Grunbauer, D. Harel, & T. Hoare (Eds.) *Engineering Theories of Software Intensive Systems.* Springer.

Bottoni, P., Koch, M., Parisi-Presicce, F., & Taentzer, G. (2001). A visualization of OCL using collaborations. *Proceedings of the UML 2001* (pp. 257-271). The Unified Modeling Language (LNCS 2185). Springer.

Brose, G. (2000). A typed access control model for CORBA. *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS)* (LNCS 1895, pp. 88-105). Springer

Brose, G. (2001a). *Access control management in distributed object systems.* PhD thesis, Freie Universität Berlin.

Brose, G. (2001b). Raccoon—An infrastructure for managing access control in CORBA. *Proceedings of the International Conference on Distributed Applications and Interoperable Systems (DAIS).* Kluwer.

Brose, G. (2002). Manageable access control for CORBA. *Journal of Computer Security, 10*(4), 301-337.

Brose, G., Koch, M., & Lohr, K. P. (2002). Integrating access control design into the software development process. *Proceedings of the 6th International Conference on Integrated Design and Process Technology (IDPT)*

Brose, G., Koch, M., & Lohr, K. P. (2003, January 3). Entwicklung und Verwaltung von Zugriffsschutz in verteilten Objektsystemen–eine Krankenhausfallstudie. In Praxis der Informationsverarbeitung und Kommunikation (PIK), 26, 1. KG Saur Publishing.

D'Souza, D., & Wills, A. (1998). *Components and frameworks: The catalysis approach*. Boston: Addison Wesley.

Devanbu, P. T., & Stubblebine, S. (2000). Software engineering for security: A roadmap. In A. Finkelstein (Ed.), *The future of software engineering.* New York: ACM Press.

Ehrig, H., Engels, G., Kreowski, H. J., & Rozenberg, G. (1999). *Handbook of graph grammars and computing by graph transformations. Vol. II: Applications, Languages, and Tools.* Singapore: World Scientific.

Epstein, P., & Sandhu, R. (1999). Towards a UML based approach to role engineering. *Proceedings of the ACM RBAC*. New York: ACM Press.

Fernandez-Medina, E., Martinez, A., Medina, C., & Piattini, M. (2002). UML for the design of secure databases: Integrating security levels, user roles, and constraints in the database design process. In J. Jurjens, M. V. Cengarle, E. B. Fernandez, B. Rumpe, & R. Sandner (Eds.), *Proceedings of the CSDUML02* (pp. 93-106). Number TUMI0208 in Technical Report TU Munchen.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R., (2001). Proposed NIST standard for rolebased access control. *ACM Transactions on Information and System Security, 4*(3), 224-274.

Harrison, M., Ruzzo, M., & Ullman, J. (1976). Protection in operating systems. *Communications of the ACM, 19*(8), 461-471.

Interactive-Objects. (2005). *Arcstyler.* Retrieved from www.interactive-objects.com

Jaeger, T., & Tidswell, J. (2001). Practical safety in flexible access control models. *ACM Transactions on Information and System Security, 4*(2), 158-190.

Jurjens, J. (2001). Towards development of secure systems using UMLsec. *Proceedings of the FASE'01* (pp. 187-200). (LNCS 2029). Springer.

Jurjens, J. (2002). UMLsec: Extending UML for secure systems development. *Proceedings of the UML 2002* (pp. 412-425). Number 2460 in Lect. Notes in Comp. Sci., Springer.

Jurjens, J. (2005). Secure systems development with UML. Heidelberg: Springer-Verlag.

Koch, M., Mancini, L.V., & Parisi-Presicce, F. (2001). Foundations for a graphbased approach to the specification of access control policies. In F. Honsell & M. Miculan (Eds.), *Proceedings of the Foundations of Software Science and Computation Structures (FoSSaCS 2001),* (LNCS 2030, pp. 287-302). Springer.

Koch, M., Mancini, L.V., & Parisi-Presicce, F. (2002a). Conflict detection and resolution in access control specifications. In M. Nielsen & U. Engberg (Eds.), *Proceedings of the Foundations of Software Science and Computation Structures (FoSSaCS 2002)* (LNCS 2303, pp. 223-237). Springer

Koch, M., Mancini, L.V., & Parisi-Presicce, F. (2002b). Decidability of safety in graphbased models for access control. *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)* (LNCS 2502, pp. 229-243).

Koch, M., Mancini, L. V., & Parisi-Presicce, F. (2002c, August). A graph based formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC), 5*(3), 332-365.

Koch, M., & Parisi-Presicce, F. (2002d). Access control policy specification in UML. In Jurjens, Cengarle, Fernandez, Rumpe, & Sandner (Eds.), *Proceedings of CSDUML02* (pp. 63-78). Technical Report TUMI0208, TU Munchen.

Koch, M., & Pauls, K. (2005). An access control language for dynamic systems modeldriven development and verification. *Proceedings of the 12th SDL Forum.*

Koch, M., & Pauls, K. (2005). Modeldriven development of access control aspects. *Proceedings of the Sicherheit 2005* (pp. 273-284). Lecture Notes in Informatics.

Lodderstedt, T., Basin, D., & Doser, J. (2002). SecureUML: A UML based modeling language for modeldriven security. *Proceedings of the 5th International Conference on the Unified Modeling Language*,(LNCS 2460). Springer.

OMG. (1999, October). CORBA 3.0 New Components Chapters, TC Document ptc/991004. OMG.

OMG. UML 2.0 Specification. OMG, 2003.

Ray, I., Li, N., France, R., & Kim, D. K. (2004). Using UML to visualize rolebased access control constraints. *Proceedings of the SACMAT'04* (pp. 115-124). ACM.

Rozenberg, G. (1997). *Handbook of graph grammars and computing by graph transformation* (Vol. 1). Foundations. Singapore: World Scientific.

Sandhu, R., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *IEEE Computer, 29*(2), 38-47.

Sun Microsystems. (2000, October). *Enterprise JavaBeans Specification, Version 2.0*, Final Draft. Retrieved from http://java.sun.com/products/ejb/docs.html

Taentzer, G (2001). Towards common exchange formats for graphs and graph transformation systems. *Proceedings of Uniform Approaches to Graphical Process Specification Techniques UNIGRA'01*, (ENTCS 47). Elsevier.

Taentzer, G., Ermel, C., & Rudolf, M. (1999). The AGG approach: Language and tool environment. In H. Ehrig, G. Engels, H.-J. Kreowski, & G. Rozenberg (Eds.), *Handbook of graph grammars and computing by graph transformation* (Vol. 2). Singapore: World Scientific.

This work was previously published in Integrating Security and Software Engineering: Advances and Future Visions, edited by H. Mouratidis & P. Giorgini, pp. 220-243, copyright 2007 by Information Science Publishing (an imprint of IGI Global).

Chapter 7.10
Ethics in Software Engineering

Pankaj Kamthan
Concordia University, Canada

INTRODUCTION

As software becomes pervasive in our daily lives, its values from a purely human perspective are brought to light. Ethical conduct is one such human value.

There are various reasons for discussing the issue of ethics within a software engineering context. By participating in a software development process, software engineers can influence the final product, namely the software itself, in different ways including those that may be contrary to public interest. In other words, they could engage in an unethical behavior, inadvertently or deliberately. This could lead to personal harm, and potentially result in loss of confidence in software and loss of trust in organizations that own them. This can adversely affect the acceptance of software as a useful product, question the credibility of software engineering as a profession, lead to legal implications, and impact the bottom line of the software industry at-large.

This article is organized as follows. We first outline the background necessary for later discussion. This is followed by a proposal for a quality-based framework for addressing ethics, and software quality treatment of a software engineering code of ethics. Next, avenues and directions for future research are outlined, and finally, concluding remarks are given.

BACKGROUND

By viewing software engineering as a profession, we define ethics as a code of professional standards, containing aspects of fairness and duty to the profession and the general public.

Since a software can either be a benefit or a hazard to its potential users, the issue of ethics in

its engineering arises. Software failures (Sipior & Ward, 1998) that have led to loss of human life, rendered computer systems unusable, led to financial collapse, or caused major inconveniences are grim reminders of that.

In this article, we discuss the issue of ethics from the viewpoint of software product quality considerations in practice. There is an apparent symbiosis between ethics and quality. For example, the causes of the aforementioned failures were attributed to violations of one or more quality attributes such as reliability, safety, and so forth, and/or to lack of proper validation/verification of these.

Indeed, in the Software Engineering Body of Knowledge (SWEBOK) (Abran, Moore, Bourque, & Dupuis, 2001), ethics has been placed within the software quality "knowledge area." The issue of information technology in general, and the role of quality in software development in particular, have been addressed in (Reynolds, 2003; Tavani, 2004). Moreover, software quality is viewed as an ethical issue from a philosophical perspective (Peslak, 2004). However, these efforts are limited by one or more of the following issues: quality and ethics are often viewed as a tautology, treatment of software quality is at a very high level and often as a single entity, and there is lack of specific guidance for improvement of software quality within the domain of software ethics.

One way to enforce ethical standards in a software project is by explicitly documenting the ethical expectations from stakeholders such as via a *code of ethics*. The Software Engineering Code of Ethics and Professional Practice (SECEPP) is a recommendation of the ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices. SECEPP puts forth eight categories of principles decomposed further into clauses that software engineers should adhere to in teaching and practicing software engineering. However, these principles and associated clauses suffer from several issues (expounded in the next section): lack of separation (of concerns), recency, precision, completeness, reachability (to certain audience), and specificity, which makes their realization difficult. The relevance of SECEPP for practical purposes has been questioned (Qureshi, 2001), however the view is largely managerial rather than oriented towards the software product.

ETHICS IN SOFTWARE ENGINEERING AND SOFTWARE PRODUCT QUALITY

For the purpose of this article, our understanding of the discussion on ethics in software engineering is based on the following interrelated hypothesis:

Hypothesis 1. Ethical behavior is dynamic, rather than static. Specifically, by appropriate means (such as code of ethics), ethical actions of software engineers could be regulated and with education even be instilled.

Hypothesis 2. Ethics is a "meta-concern" (Qureshi, 2001) leading us to adoption of steps for software quality assurance and evaluation. Specifically, ethics and software quality are related by direct proportionality, and so overall improvement in the quality of a software product leads to an improvement in ethical considerations related to that product.

A Theoretical Framework for Addressing Ethics from a Software Product Quality Perspective

In order to address the practicality of introducing the ethical dimension in software engineering, we first need a theoretical foundation. To do that, we separate the concerns involved as follows:

Table 1. A framework for ethics in a semiotic approach to software product quality

Ethical Concern	Software Product		
Semiotic Level	**Levels of Quality Attribute**	**Example(s) of Quality Attributes**	**Decision Support**
Social	External: Tier 1	Credibility, Trust	Feasibility
	External: Tier 2	Legality, Safety	
	External: Tier 3	Privacy, Security	
Pragmatic	External: Tier 1	Accessibility, Maintainability, Usability	
	External: Tier 2	Interoperability, Portability, Reliability	
Semantic	Internal	Completeness, Validity	
Syntactic	Internal	Correctness	
Empirical	Internal	Characters, Character Set	
Physical	Internal	Hardware Characteristics	

1. View ethics as a qualitative aspect and attempt to address it via quantitative means so as to minimize the potential for heuristics and to make the evaluation repeatable.
2. Select a theoretical basis for communication of information, and place ethics within its setting.
3. Address software product quality in a systematic and practical manner by means of adopting a quality model. In particular, select the quality model that separates internal and external quality attributes.

Using this as a basis, we propose a framework for ethics from the perspective of software product quality (see Table 1).

We now describe each of the components of the framework in detail.

Semiotic Levels

The first column of Table 1 states the semiotic levels. Semiotics (Stamper, 1992) is concerned with the use of symbols to convey knowledge. From a semiotics perspective, a representation can be viewed on six interrelated levels: physical, empirical, syntactic, semantic, pragmatic, and social, each depending on the previous one in that order.

The physical level is concerned with the physical representation of signs in hardware and is not of direct concern here. The empirical level is responsible for the communication properties of signs. The syntactic level is responsible for the formal or structural relations between signs. The semantic level is responsible for the relationship of signs to what they stand for. The pragmatic level is responsible for the relation of signs to interpreters. The social level is responsible for the manifestation of social interaction with respect to signs.

Software Product Quality Attributes and Examples

The second column of Table 1 draws the relationship between semiotic level and software product quality attributes.

The first level of quality attributes gives the *external attributes,* which on decomposition leads to second level of quality characteristics called *internal attributes* that are directly measurable via quality metrics. External attributes are extrinsic to the software product and are directly the user's

concern, while internal attributes are intrinsic to the software product and are directly the software engineer's concern.

Now, external attributes in the collection are *not* all at the same level. For example, security is a necessary condition for safety, which in turn is a necessary condition for credibility of a software product. Therefore, credibility of a software product is at a higher echelon than safety, which in turn is higher than security. So we extend the traditional software product quality models by introducing a further decomposition of external attributes into different tiers.

The third column of Table 1 includes examples of quality attributes at different semiotic levels based upon our experience from past software projects in different domains, observations from several software product quality models (Lindland, Sindre, & Sølvberg, 1994; Fenton & Pfleeger, 1997), and most importantly, those that have a direct impact on stakeholders. Their purpose is only to make the argument concrete, not to mandate a required list (that could be shortened or lengthened based on the application domain anyway).

Decision Support

The practice of software engineering must take into account organizational constraints (personnel, infrastructure, schedule, budget, and so on). These, along with external forces (market value or competitors), force us to make software-related decisions that could cross over on either side of the ethical boundaries. Therefore, the practice of ethics (say, enforcement via code of ethics) must also be feasible and is acknowledged in the last column of Table 1. There are well-known techniques for carrying out feasibility analysis, but further discussion of this aspect is beyond the scope of this article.

Ethics and Software Product Quality: Making SECEPP Practical

The following issues inherent to principles and associated clauses SECEPP make their realization difficult:

- **Separation:** The concerns are not separated clearly, and there are apparent overlaps across principles. For example, cost estimation is addressed in both Principle 3: PRODUCT and Principle 5: MANAGEMENT.
- **Recency:** The last version of SECEPP was announced in 1999 and is seemingly not constantly maintained.
- **Precision:** The clauses often include terminology that is either vague or open to broad interpretation, and many of the terms therein lack links/references to standard definitions. For example, Principle 3: PRODUCT states that "software engineers shall ensure that their products and related modifications meet the highest professional standards possible." However, it is unclear what "highest" means in this context and by whose measure, the reason for the distinction between "products and related modifications," and which "professional standards" are being implied.
- **Completeness:** SECEPP does not address certain classical practices such as (ethics in the light of) software reuse, reengineering, or reverse engineering, or current practices such as software modeling, and therefore needs to be extended. In retrospective, SECEPP does point out that the list of principles and clauses in it are not exhaustive.
- **Reachability:** The principles and/or clauses are often at a high level of abstraction that can make them prohibitive for novices such as beginning software engineering students

that need to start on the right foot. In this sense, SECEPP resembles the problems inherent in the use of guidelines per se.

- **Specificity:** In order to be applicable to the field of software engineering, SECEPP had to be both abstract and general. It does not include concrete examples for or against ethical conduct for development scenarios for specific software such as Web applications.

One of the main purposes of the article is to tackle some of these issues, particularly as they relate to the software product.

The commentary on SECEPP is structured as follows. Each principle and associated clause (and any sub-statements) in SECEPP that is seen within the scope of non-trivial quality considerations is listed verbatim in *italics*. This is followed by corresponding [Issues] and [Resolutions] for every case. The resolutions vary in their coverage where some address the issues better and in a more complete manner than others. We do not claim that the resolutions as being absolute, and their limitations are pointed out where necessary.

Principle 1: PUBLIC. Software engineers shall act consistently with the public interest. In particular, software engineers shall, as appropriate

1.07. Consider issues of physical disabilities…that can diminish access to the benefits of software.

[Issues] It is unclear why only *physical disabilities* are being addressed when software is also used by people with other forms of disabilities (such as visual, auditory, and cognitive/neurological) as well. Also, there are no details on means for addressing accessibility concerns.

[Resolutions] For the Web applications domain, the Web Content Accessibility Guidelines (WCAG) and its ancillary specifications or the Section 508 mandate of the U.S. Government Federal Access Board provide detailed guidelines and examples/non-examples of assuring accessibility. There are tools available that can automatically test for conformance to these guidelines.

3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.

[Issues] In absence of further details, it is unclear what are considered as *achievable goals*, how to define and achieve them, and when they are considered having been achieved.

[Resolutions] A goal-oriented software project is seen as a set {goals, means} to be satisfied. Indeed, one view of goal-oriented software engineering in general and measurement in particular is given by the Goal-Question-Metrics (GQM) approach (Van Solingen & Berghout, 1999). In GQM, a set of measurement goals are stated, corresponding to each of one or more questions asked, followed by one or more metrics to tackle each question. One of the limitations of GQM, however, is that it does not explicitly address the issue of feasibility.

3.07. Strive to fully understand the specifications for software on which they work.

[Issues] The term *fully understand* is not defined. If it means each software engineer must comprehend all aspects of a given specification at all times, then this goal is unrealistic.

[Resolutions] For our purpose, a working definition of comprehension is the acquisition of the explicit knowledge inherent in a specification. For a non-trivial specification, it is not realistic that each stakeholder will be able to comprehend each statement made by the specification in its entirety at all times. A comprehension is *feasible* (Lindland et al., 1994) if the cost of reducing the incomprehensible statements in a specification does not exceed the drawbacks of retaining them.

3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

[Issues] It is unclear what *adequate* or *significant* means here. There is also no suggestion as to what kind of documentation is being addressed, and how and where such documentation should appear.

[Resolutions] All documentation should be minimal in the sense that if there is some documentation, then that must correspond to a problem/solution, and vice versa, and it should not be redundant (never appear in more than one place). This can be addressed via the "single source approach." All documentation should be modular in the sense that semantically similar aspects should be cohesive, and clearly separated from others. For example, all design decisions, including the ones not included, corresponding to each software requirement must be uniquely identified and rationalized, and appear in the software design description document. All documentation should be "globally" (anybody, anywhere, any device, any time) reachable. This could, for example, be done by representing software process documentation in the Extensible Markup Language (XML). The guidance for structuring and presenting technical documents has been given (Houp, Pearsall, Tebeaux, and Dragga, 2005).

3.12. Work to develop software and related documents that respect the privacy of those who will be affected by that software.

[Issues] In absence of details, it is non-trivial to address privacy concerns.

[Resolutions] One of the main privacy concerns that the users have is not knowing how or by whom the personal information that they have voluntarily or involuntarily released will be used. The Platform for Privacy Preferences Project (P3P) enables providers to express their privacy practices in a format that can be retrieved automatically and interpreted easily by user agents. This ensures that users are informed about privacy policies before they release personal information. A P3P Preference Exchange Language (APPEL) complements P3P and allows a user to express his or her preferences in a set of preference-rules that are placed in a profile document. This can then be used by the user agent to make decisions regarding the acceptability of machine-readable privacy policies from P3P-enabled applications.

Challenges to a Software Quality Approach to Addressing Ethics in Software Engineering

Open Source Software (OSS) suffers from various quality issues (Michlmayr, Hunt, & Probert, 2005), including that of usability and maintainability. Since there are modest opportunities, if any, for enforcing ethical regulations in the OSS realm, this brings into question the ways in which OSS can be used.

Due to constraints not imposed by choice but by technical considerations in practice of software engineering, quality attributes in technical and social tiers compete. For example, the pairs of usability–security or maintainability–efficiency are often in competition.

Finally, we note that there are ethical predicaments in software development that are irreconcilable from a software quality perspective. For example, a software engineer working for a company producing defense software could, when asked to make a decision on including a certain feature in software that will lead to striking improvement in a certain weapon capability, find his or her ethical commitments to the organization in conflict with his or her ethical obligations to society at-large. Therefore, alternative means of tackling such scenarios are desirable.

FUTURE TRENDS

Since project and process quality is not mutually exclusive from the product quality, it would be worth examining them in general and within the SECEPP framework. For example, according to the Capability Maturity Model (CMM), the organizations below Level 3 do not have formal quality evaluation practices, such as inspections, in place.

In a similar spirit of organizational quality management, it would of interest to study the interplay between ethics and the Total Quality Management (TQM) that focuses on the optimization of industrial processes under economic considerations in such a way as to achieve customer satisfaction.

Finally, it would be of interest to closely investigate the ethical concerns in specific software domains such as mobile or Web applications (Johnson, 1997). The issue of ethics is also important to these applications as they are increasingly being used as technological means for persuading consumers (Fogg, 2003). To that regard, a recast of SECEPP applicable to Web engineering is also highly desirable.

CONCLUSION

Software engineers are expected to share a commitment to software quality as part of their culture (Wiegers, 1996). There needs to be a regulatory and/or educational shift to accommodate making ethics a first-class member of this culture as well.

The improvement of software quality is technical as well as an ethical imperative. By addressing software product quality in a systematic manner, ethical concerns can be largely attended to.

Organizations that are engaged in software production from an engineering standpoint need to adhere to a code of ethics and monitor its practice. Institutions that have software engineering as part of their curriculum must include the role of ethics on the forefront. A code of ethics such as SECEPP can be helpful in that regard, but it needs to evolve to be more effective in such practical situations.

REFERENCES

Abran, A., Moore, J.W., Bourque, P., & Dupuis, R. (2001). *Guide to the software engineering body of knowledge—SWEBOK*. IEEE Computer Society.

Fenton, N.E., & Pfleeger, S.L. (1997). *Software metrics: A rigorous & practical approach.* International Thomson Computer Press.

Fogg, B.J. (2003). *Persuasive technology: Using computers to change what we think and do.* San Francisco: Morgan Kaufmann.

Houp, K.W., Pearsall, T.E., Tebeaux, E., & Dragga, S. (2005). *Reporting technical information* (11th ed.). Oxford: Oxford University Press.

Johnson, D.G. (1997). Ethics online. *Communications of the ACM, 40*(1), 60-65.

Lindland, O.I., Sindre, G., & Sølvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software, 11*(2), 42-49.

Michlmayr, M., Hunt, F., & Probert, D.R. (2005, July 11-15). Quality practices and problems in free software projects. *Proceedings of the 1st International Conference on Open Source Systems* (OSS 2005), Genova, Italy.

Peslak, A.R. (2004, April 22-24). Improving software quality: An ethics based approach. *Proceedings of the 2004 SIGMIS Conference on Computer Personnel Research: Careers, Culture, and Ethics in a Networked Environment* (SIGMIS 2004), Tucson, AZ.

Qureshi, S. (2001). How practical is a code of ethics for software engineers interested in quality? *Software Quality Journal, 9*(3), 153-159.

Reynolds, G. (2003). *Ethics in information technology.* Thompson Publishing Group.

Sipior, J.C., & Ward, B.T. (1998). Ethical responsibility for software development. *Information Systems Management, 15*(2), 68-72.

Stamper, R. (1992, October 5-8). Signs, organizations, norms and information systems. *Proceedings of the 3rd Australian Conference on Information Systems,* Wollongong, Australia.

Tavani, H.T. (2004). *Ethics and technology: Ethical issues in an age of information and communication technology.* New York: John Wiley & Sons.

Van Solingen, R., & Berghout, E. (1999). *The goal/question/metric method: A practical method for quality improvement of software development.* New York: McGraw-Hill.

Wiegers, K. (1996). *Creating a software engineering culture.* Dorset House.

KEY TERMS

Code of Ethics: A resource that describes the ethical and professional obligations with respect to which the public, colleagues, and legal entities can measure the behavior of and decisions made by a professional.

Quality: The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs.

Quality Model: A set of characteristics and the relationships between them that provide the basis for specifying quality requirements and evaluating quality of an entity.

Semiotics: The field of study of signs and their representations.

Software Engineering: A discipline that advocates a systematic approach of developing high-quality software on a large scale while taking into account the factors of sustainability and longevity, as well as organizational constraints of time and resources.

Software Failure: A departure from the software system's required behavior from a user's perspective.

Software Fault: A human error in performing some software activity from a software engineer's perspective. A fault is not a sufficient condition for a software failure.

This work was previously published in Encyclopedia of Information Ethics and Security, edited by M. Quigley, pp. 266-272, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.11
Free Access to Law and Open Source Software

Daniel Poulin
Université de Montréal, Canada

Andrew Mowbray
University of Technology, Sydney, Australia

Pierre-Paul Lemyre
Université de Montréal, Canada

ABSTRACT

Law consists of legislation, judicial decisions, and interpretative material. Public legal information means legal information produced by public bodies that have a duty to produce law and make it public. Such information includes the law itself (so-called primary materials) as well as various secondary (interpretative) public sources such as reports on preparatory work and law reform and resulting from boards of inquiry and available scholarly writing. The free access to law movement is a set of international projects that share a common vision to promote and facilitate open access to public legal information. The objectives of this chapter are to outline the free access to law movement, to set out the philosophies and principles behind this, and to discuss the role that open source software has played both in terms of its use and development.

INTRODUCTION

The *free access to law movement* is a set of international projects that share a common vision to promote and facilitate open access to public legal information. There are direct synergies between the notion of "freeing the law" by providing an alternative to commercial systems and the ideals that underpin *open source software*. In addition, open source software has been an essential foundation for the work that has been done and new open source code has been developed.

The objectives of this chapter are to outline the *free access to law movement*, to set out the philosophies and principles behind this, and to discuss the role that open source software has played both in terms of its use and development. It concludes with an assessment of what has been achieved and of the similarities between the free access to law and open source software movements.

BACKGROUND

Law consists of legislation, judicial decisions and interpretative material. *Public legal information* means legal information produced by public bodies that have a duty to produce law and make it public. This includes the law itself (so-called *primary materials*) as well as various *secondary* (interpretative) public sources such as reports on preparatory work and law reform and resulting from boards of inquiry and available scholarly writing. It also includes legal documents created as a result of public funding.

Lawyers have been interested in the electronic publication of legal materials and associated information retrieval systems for a very long time. The earliest reported experiment is generally said to have been done by John Horty at the University of Pittsburgh in the late 1950s (Bing, 2004). The first major commercial system appeared in 1973 with the launch of *Lexis* (now *LexisNexis*). This was based on an earlier system developed by the Ohio Bar (OBAR) which had been established in 1969. OBAR was acquired by Mead Data Central and redesigned to become Lexis. LexisNexis is now one of the largest commercial text databases in the world. It is currently owned by the Reid publishing group. Lexis was followed by *Westlaw* in 1976. Westlaw is now owned by Thomson Publishing and is the major business competitor to Lexis. Several other commercial and government based systems also appeared about this time, but were largely ultimately unsuccessful such as the now defunct European system *EUROLEX* and the Australian system *SCALE* (Greenleaf, Mowbray, & Lewis, 1988).

In the 1980s and 1990s, Lexis and WestLaw expanded the scope of their services to include international collections and in their original jurisdiction (the United States) established a near duopoly (McKnight, 1997; Arewa, 2006). Attempts were made in various other places such as Australia and Canada to create either government or government sanctioned commercial monopolies (Greenleaf et al., 1988).

The resulting environment was, and to some extent still is one that is characterised by limited access to basic legal materials. Whilst the commercial systems provide a very sophisticated set of services they are for the most part targeted at the legal profession, they require significant training in order to use them. The services are very expensive and generally are not available for casual use. Non-lawyers seldom access the commercial systems and even lawyers can often not afford to use them.

Why is Free Access to Legal Information Important?

At the most fundamental level, access to public legal information supports the rule of law. Citizens are governed by laws and so have a need and right for effective access to these laws. Businesses also generally operate in a regulated environment and have similar needs. Effective access to basic legal information is essential both from a social perspective and also to facilitate the proper operation of business and commerce.

Apart from being able to access domestic laws, there is also increasingly a need to access law from other jurisdictions. Business operates on an international basis. Corporations need to be aware of international regulatory requirements and countries need to make their legal systems

transparent to encourage international investment and trade. Particularly in the case of developing countries, there is a major need for access to international laws to assist with law reform and development (Poulin, 2004).

The Free Access to Law Movement

The *free access to law movement* has grown out of a set of projects that have attempted to address these issues and to provide alternatives to the commercial legal publishers' systems. Most of these projects are called *legal information institutes* (or *LII*s for short).

The earliest initiatives were in the United States and Canada. In 1992, Tom Bruce and Peter Martin established the Cornell *Legal Information Institute* (Bruce, 2000). This service was initially based on *Gopher* and provided free access to decisions of the United States Supreme Court and the United States code. It moved to the Web in 1994. In Canada, Daniel Poulin and his team at *LexUM* started publishing the full text of decisions of the Canadian Supreme Court in 1993 (Poulin, 1995).

Both systems helped to identify a strong demand for free public access to primary legal materials. In Australia, Graham Greenleaf and Andrew Mowbray founded *AustLII* (the *Australasian Legal Information Institute*) in 1995 (Greenleaf, Mowbray, King, & van Dijk, 1995). By the end of the year, AustLII was publishing some 16 databases including the decisions of most of the major Australian federal courts as well as federal and state legislation and by 1998 became the first LII to achieve national coverage. It now includes over 200 databases covering virtually all courts and tribunals in the country.

Other systems adopting a similar approach followed. These included the *British and Irish Legal Information Institute* (*BAILII*) in 1999, the *Pacific Islands Legal Information Institute* (*PACLII*) and the *Canadian Legal Information Institute* in 2000, and the *Hong Kong Legal Information Institute* (*HKLII*) in 2003. Various *meta-systems* were also built that drew upon the information contained in the other LIIs (*WorldLII, Droit francophone* and *CommonLII*).

The *free access to law movement* was proclaimed at the annual *Law via the Internet* conference in Montreal in 2002. The current terms of the *Montreal Declaration* (as amended in Sydney, November 29, 2005 and Paris November 5, 2004) are (in part):

Legal information institutes of the world, meeting in Montreal, declare that:

- *Public legal information from all countries and international institutions is part of the common heritage of humanity maximising access to this information promotes justice and the rule of law;*
- *Public legal information is digital common property and should be accessible to all on a non-profit basis and free of charge;*
- *Independent non-profit organisations have the right to publish public legal information and the government bodies that create or control that information should provide access to it so that it can be published.*

...

Legal information institutes:

- *Publish via the internet public legal information originating from more than one public body;*
- *Provide free, full and anonymous public access to that information;*
- *Do not impede others from publishing public legal information; and*
- *Support the objectives set out in this Declaration.*

Each LII is responsible for publishing legal materials for a particular country or geographical region. AustLII, for example, publishes materials for Australasia (i.e., Australia and New Zealand).

Apart from providing access to the full-text decisions of all major courts (such as the High Court, Federal Court, and State Supreme Courts), as has been said, AustLII also publishes decisions of nearly all Australian tribunals. Access to consolidated (and in some cases, *point in time*) legislation and regulations from all nine jurisdictions is also available. Other content includes: Law Reform Commission reports from most States; access to most Australian law journals; and a database of all bilateral and multi-party treaties.

Like most of the other LIIs, AustLII uses automated processes to add rich hypertext markup to its materials. In all, the system currently includes around 40 million internal hypertext links. Free text searching is available over the entire system or selected databases. AustLII is the major source of legal information in Australia and accounts for 25-30% of all legally related traffic in the country.

At the time of writing, the various LIIs together publish around 663 databases containing legal materials from 86 countries as well as 21 international collections. The total number of individual documents exceeds 3 million. Total usage is estimated to be in the vicinity of 3.5 million direct hits (or page accesses) per day.

The content of these databases consists mainly of *primary materials*—that is, court decisions, legislation and treaties, but increasingly secondary materials such as law journals, law reform commission reports, and the like are being added.

The LIIs have changed the way that law is made available to the public. Whereas in the past, there was exclusive reliance upon commercial publishers as conduits for the dissemination of this information, primary legal information now flows directly from courts and governments to consumers. The LIIs freely offer a level of value adding that establishes a new baseline for commercial publishers. Examples of this value adding include hypertext markup and search capabilities. The citator created by LexUM for CanLII (Reflex) provides a further example (Poulin, Paré, & Mokanov, 2005).

Each LII concentrates on making available domestic laws, but beyond these local endeavors all LIIs collaborate to expand the freely accessible law space internationally. This collaboration takes many forms. First of all, they all participate in promoting and supporting free access to the law by lobbying data providers such as courts, governments, and other bodies. They also provide, within their means, technical assistance and advice and training to other organizations. They hold annual conferences in order to exchange information and share knowledge. These conferences are public and all those interested can register to take part. Since many LIIs are based in universities, a significant part of those conferences is set aside for academic exchange of research results.

This cooperative spirit can be easily illustrated by the collaboration between the University of the South Pacific and AustLII to establish PacLII. Robynne Blake had worked for a number of years to build a substantial collection of South Pacific legal materials. AustLII assisted by provision of technical know-how and their software. In 2006, after many years of progress PacLII obtained a large grant from New Zealand Aid to expand its reach towards making the laws of the various states of the area freely accessible on the Internet. PacLII is now (in terms of the number of staff) one of the largest of the LIIs.

Similarly, LexUM collaborated with many interested parties in Burkina Faso to establish Juriburkina. Today, Juriburkina is operated from Ouagadougou by the local bar association and with the support of the higher courts, government general secretariat and a local Internet startup called ZCP informatique. A similar approach is being followed in Senegal and the project has reached implementation stage.

MAIN FOCUS OF THE CHAPTER

Development of Interest in Open Source Software

The LII promoters and developers were not always early adopters of open source software. Although most of the LIIs were Unix based, the significance of open source software only started to become more evident towards the end of the 1990s. Today, not only is most of the software used by LIIs open source, but the LIIs have themselves started to offer elements of their own production software under open source licences.

Many reasons may be put forward to explain the initial caution. First of all, in the early 1990s, open source was not as developed and mature as it is today. At the time, the LIIs rightly set "making the law accessible for free" as their principal agenda item. To achieve this, the most effective software, proprietary or otherwise was deployed. The reluctance towards more generally embracing open source by the LIIs, was partly based on the lack of maturity of the available open source software and partly attributable to the dominant prevailing prejudice towards conventional corporate approaches.

There was a major reappraisal of the initial attitude towards open source software from around 1998. At the time, for example, the operating system of choice for the LexUM's servers was Solaris from Sun Microsystems (this was also in use at AustLII and the Cornell LII). However, LexUM's programmers were mostly undergraduates and some of them had Linux installed on their home computers. These programmers were aware of the value of open source and argued strongly for the adoption of GNU/Linux. In the course of this campaign, they had even installed for demonstration purposes another open source flagship of the time, the already well respected—Apache Web server to replace the Netscape Enterprise server that was then in use. But despite the apparent functioning of Apache, LexUM, was reluctant to abandon the safety of using a proprietary solution for what appeared to be a more risky free alternative.

Then as today, LexUM was working with the Supreme Court of Canada (SCC) to make its decisions available for free in a timely manner. A long-awaited SCC judgment was expected on August 20, 1998 when the court's decision on the legality of a unilateral secession of Quebec from Canada was to be published. The morning the decision became available, the LexUM Netscape Enterprise based server went down at the moment the decision became available. The server was unable to cope with the rise in demand. After over an hour of rebooting the server, LexUM's student programmers brought up the Apache based sever. The move saved the day, and Apache kept running without failure for many weeks. From then on, LexUM used Apache as its Web server. In the following years, LexUM switched all of its servers to Linux and Apache.

The other LIIs had similar experiences. Most either had already or were soon to adopt Apache. Many moved to Linux and to generally adopt open source software as the basis of their production systems.

Current Use of Open Source Software

Although the commitment to open source has never been a religious one, most of the LIIs are nevertheless strongly reliant upon open source software. Although this is partly a matter of simple economics, this is not of itself sufficient to drive the adoption of open source as even free bad software is still obviously a poor choice. The open source orientation leads to a twofold benefit: savings in licence costs, but more importantly it led to the provision of reliable tools and powerful products to achieve the vision of freely accessible law. The current approaches used by the LIIs

closely match open source trends. Open source developers develop many tools targeted for the Web that closely meet the needs of LIIs.

As has been said above, most of the LIIs use GNU/Linux and Apache. In addition some commonly used open source programs include database and indexing programs such as PostgreSQL, Open LDAP, Apache Lucene, and Nutch; programming languages and tools that include: perl, python, gcc, Eclipse, mod_perl, and Mason; and various other tools such as FastCGI and Mason.

Proprietary software is still used but only where a suitable open source solution cannot be identified. For example, most LIIs still rely upon proprietary software for a significant part of basic document preparation and conversion (such as Microsoft Word) and for some aspects of network security (for example, AustLII uses Check Point and Tripwire).

FUTURE TRENDS

Development of Open Source Software by LIIs

Prior to the World Wide Web, the publishing of databases of legal information was essentially the work of commercial publishers who used specialised software that had often been developed in-house. The Web brought with it a number of generic publishing tools such as conversion tools, search tools and Web servers. However, tools to support more specialised legal publishing needs remained rare. This led a number of the LIIs to develop the tools they needed.

One of the first of these was *Sino* (short for "size is no object"). Sino is a high performance free text search engine. It was originally written in 1995 and has been mainly used to provide production level search facilities for most of the Legal Information Institutes that form part of the free access to law movement. Sino went to a major rewrite in 2006 that makes it even faster and adds new functionality. Sino from its initial release has always been a very fast search engine and its indexing and searching time have been kept at the level of the fastest proprietary products.

Sino is designed to be easy to interface with via a simple C/Perl API as well as a ready written interactive interface for testing or for actual use on Unix sockets. The tool is relatively small and easy to understand at about 12K lines of ANSI/POSIX.1 compliant C code. Sino concordances (indexes) are portable across platforms with different architectures. Sino has been in use on a number of major Web sites answering many millions of requests for the past 10 years and so is robust and reliable.

Sino is a tool aimed at improving the access to the law. It was at the heart of AUSTLII from the very beginning and has been subsequently adopted by BAILII, PacLII and HKLII. LexUM used it for CanLII for many years. From 1995 until 2006, Sino and its source code were made available for free to anybody wanting to publish the law openly and for free. With its last rewrite, Sino became open source and it is now licensed under the GNU General Public Licence (GPL).

LexUM has also developed a number of pieces of open source software. *LexEDO* is a legal publishing platform aimed at providing a ready-made and easy to use solution for small-scale publication projects particularly in the developing world. LexEDO provides a means to manage legislation, caselaw, and legal periodicals as simple databases, to automatically convert documents to PDF and HTML and to generate a Website accordingly. All of these tasks can be accomplished by lawyers or law students acting as editors through Web-based management interfaces.

LexEDO has been distributed to such organisations as the Bar of Burkina Faso, the government general secretariat of Burkina Faso and the Bar of Senegal. In the context of these projects, the

availability of the source code was critical for capacity building purposes. In Burkina Faso for instance, LexEDO has been maintained locally for a period of over two years by a private host called ZCP Informatique. To some extent, the fact that LexEDO source code is available allows ZCP to develop local solutions to local problems without requiring LexUM's assistance. It also provides them with the means to control the evolution of their project, or even to replicate it elsewhere thus spreading free access to law. As is the case for Sino, LexEDO is distributed under the GPL.

LexUM has also developed a program called NOME to assist with the anonymisation of judicial decisions. In many jurisdictions some or all judgments must not contain the names of parties or accused. For instance, anonymisation of judicial decisions involving young offenders is mandatory in Canada. To efficiently achieve this result, LexUM worked with the Computer Science Department at the University of Montreal (Plamondon, Lapalme, & Pelletier, 2004). The result was a small program which is capable of guessing and initialising proper names in Word documents. NOME is now distributed for free with its source code.

In respect to software developed in LIIs, Sino is certainly the most mature. Sino, LexEDO, and NOME are distributed under the GPL. Various other software tools have been developed and are distributed by the LIIs to various partner organisations. As other tools become of more general application, they will become candidates to become new open source offerings.

CONCLUSION

The use of open source software by the LIIs reflects the fact that both movements are well aligned and in many senses similar. The most evident of these similarities can be listed as follows:

Avoiding Monopolistic Control over the Information

Legal information, similarly to source code, *wants to be free* (Williams, 2002). Both the free access to law and the open source software movements were conceived in reaction to the seizure of information by entities (state or commercial) not willing to share it freely with others.

Promote the Reuse of Information by Third Parties

As is the case for source code, legal information is useful only if it can be reused for various purposes. Users need the possibility to save legal documents in different formats, to send them to colleagues and to present them in courts. Some users might even need the right to reuse documents in a commercial context (for the publication of a paper based law report, e.g.).

Promote the Development of Standards

As for software development, the dissemination of legal information is improved by the adoption of standards by the players involved. These standards can take the form of uniform citation mechanisms, drafting practices or workflow models. Historically, LIIs are at the center of such initiatives.

Need to Share Tools

Organizations involved in free access to law all face the same difficulties. They constitute a community tied together by the need to edit and convert large volume of legal documents, to publish them on the Web and to provide information retrieval tools to their users. Similarly to every open source software community, LIIs have in-

centives to share their efforts in the achievement of common goals.

Proponents do not Derive Revenue from Selling Information as a Product

The source of revenue of LIIs and open source software developers is the same. It flows not from the information they publish but from the expertise they developed doing so.

Considering all these similarities, the use of open source software can easily be seen as a complementary strategy to strengthen free access to law. It allows the LIIs to achieve near complete transparency by opening-up not only the legal information, but also their publication process. By doing so, the LIIs achieve several goals at once: they guarantee (to a certain degree) the integrity of their data; they facilitate interactions with the other players in the field; and finally, they help foster the emergence of additional free access to law projects.

For people or organisations that would like to pursue free access to law projects in their own country or region, the required software is now available. There are many high quality resources available from the open source community that can be used to establish Web services. The major distributions of Linux (and other open source operating systems) and the Apache Web server are of world-class quality. There are a number of suitable search engines available. The Web and the availability of open source software means that it is now relatively straight forward to disseminate information.

For the more specialized requirements involved in publishing the law such as the conversion of data, hypertext markup, metadata extraction, and the like, the LIIs are able to make a contribution. As a result, it is increasingly the case that for those who wish to make the law more accessible, there are available tools.

REFERENCES

Arewa, O. (2006). *Open access in a closed universe, Lexis, Westlaw and the law school.* Case Legal Studies Research Paper No. 06-03. Retrieved from http://ssrn.com/abstract=888321

Bing, J. (Ed.). (1994). *Handbook of legal information retrieval.* Amsterdam: North Holland.

Bruce, T. (2000). Public legal information: Focus and future. *Journal of Information, Law and Technology,* (1).

Greenleaf, G., Mowbray, A., King, G., & van Dijk, P. (1995). Public access to law via Internet: The Australasian Legal Information Institute. *Journal of Law & Information Science, 6*(1).

Greenleaf, G., Mowbray, A., & Lewis, D. (1988). *Australasian computerised legal information handbook*. Sydney: Butterworths.

McKnight, J. (1997). Wexis versus the Net. *Illinois Bar Journal, 85*(4).

Plamondon, L., Lapalme, G., & Pelletier, F. (2004). Anonymisation de décisions de justice. *TALN Conference Proceedings,* Fès. Retrieved from http://www.lpl.univ-aix.fr/jep-taln04/proceed/actes/taln2004-Fez/Plamondon-etal.pdf

Poulin, D. (1995). Legal resources for Canadian lawyers on the Internet. *CSALT Review—Canadian Society for the Advancement of Legal Technology, 9*(1).

Poulin, D. (2004). CanLII: How law societies and academia can make free access to the law a reality. *Journal of Information, Law and Technology,* (1).

Poulin, D, Paré, E., & Mokanov, I. (2005, November 17-19). Reflex: Bridging open access with a legacy legal information system. In *Proceedings of the 7th Law via the Internet International Conference,* Port Vila, Vanuatu.

Williams, S. (2002). *Free as in freedom: Richard Stallman's crusade for free software*. Sebastopol, CA: O'Reilly & Associates.

KEY TERMS

Jurisdiction: The geopolitical region in which the laws of a certain governing body are recognized as legitimate and can be enforced.

Law: A body of knowledge consisting of legislation, judicial decisions, and interpretative material.

License: Permission needed to use or modify materials in a way that is recognized as legitimate by the owner of such materials and by an overall community familiar that recognized a similar understanding of legitimate use.

Primary Materials: Court decisions, legislation, and treaties.

Public Access: Making materials available for all members of the general public to read and review.

Public Legal Information: Legal information produced by public bodies that have a duty to produce law and make it public.

Secondary Materials: Public sources, such as reports on preparatory work, that report on and often interpret legal developments.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant & B. Still, pp. 373-381, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.12
Ethical, Cultural and Socio-Economic Factors of Software Piracy Determinants in a Developing Country:
Comparative Analysis of Pakistani and Canadian University Students

Arsalan Butt
Simon Fraser University, Canada

Adeel I. Butt
Simon Fraser University, Canada

ABSTRACT

Consumer software piracy is widespread in many parts of the world. P2P based websites have made it easier to access pirated software, which has resulted in an increased emphasis on the issue of software piracy in both the software industry and research community. Some factors that determine piracy include poverty, cultural values, ethical attitudes, and education. Earlier empirical studies have looked at software piracy as an intentional behaviour. This study explores the demographic, ethical and socio-economical factors that can represent software piracy as a social norm among a developing country's university students. The authors have conducted a comparative analysis of university students from Pakistan and Canada, two countries that differ economically, socially, and culturally. The results of the study indicate that software piracy behaviour is different in both groups of students, but that there are also some similarities. Future research directions and implications are also presented.

If nature has made any one thing less susceptible than all others of exclusive property, it is the action of the thinking power called an idea, which an individual may exclusively posses as long as he keeps it to himself; but the moment it is divulged, it forces itself into the possession of everyone, and the receiver cannot dispossess himself of it. Its peculiar character, too, is that no one possesses the less, because every other possesses the whole of it. He who receives an idea from me, receives instruction himself without lessening mine; as he who lights his taper at mine, receives light without darkening me. That ideas should freely spread from one to another over the globe, for the moral and mutual instruction of man, and improvement of his condition, seems to have been peculiarly and benevolently designed by nature, when she made them, like fire, expansible over all space, without lessening their density at any point, and like the air in which we breathe, move, and have our physical being, incapable of confinement or exclusive appropriation. Invention then cannot, in nature, be a subject of property.

-Thomas Jefferson

Congress shall have power … to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries.

-The Constitution of the United States, Article 1, Section 8, 1788

INTRODUCTION

Ethicist Richard Mason (1986) identified four main ethical issues of the information age: privacy, accuracy, property and accessibility. It has been suggested that Mason's work was very significant in the field of Management Information Systems ethics (Freeman & Peace, 2005). Mason (1986) considered intellectual property (IP) "as one of the most complex issues we face as a society" (p. 9). Mason identified *bandwidth* as the real threat in the digital world and viewed it as a scarce and fixed commodity at the time. However, with the rapid progress of hardware and software technology, bandwidth has increased immensely and has therefore made peer-to-peer (P2P) technology possible making e-file sharing a matter of few mouse clicks.

According to Husted (2000), knowledge and information are now more important factors in a national economy than the traditional physical assets that used to indicate economic well-being. Therefore, the protection of intellectual property (IP) has received increased attention in the recent past. Intellectual property refers to "the results of intellectual activity in the industrial, scientific, literary or artistic fields" (Forester & Morrison, 1990, p. 31). A government plays its role to protect the rights of owners by preventing unauthorised use of this intellectual property for a limited period of time (Seyoum, 1996) by using different measures such as copyrights, trade agreements and patents. Legality aside, there are ethical and moral issues that have risen from the use of software and its unauthorised copying both at the consumer and commercial level. The concept of ***Techno-ethics***[1] deals with such aspects of technology. "It designates that portion of ethics which deals with questions arising from technological development and activities. More precisely, technoethics deals with moral questions governing or resulting from the conception, production, distribution and the use of artifacts or technological systems" (Findeli, 1994, p. 50). In our chapter's context technoethics or ethics refers to the moralities and ethical values presumed or perceived with the use and copying of commercial software.

Software is a form of intellectual property and its unauthorized duplication is a crime. However, the practice of making illegal copies of software amounts to high rates in various parts of the world and in environments such as universities,

businesses, and government, the behaviour has become socially acceptable (Sims, Cheng, and Teegen 1996; Cheng, Sims, and Teegen 1997; Hinduja 2001; Christensen and Eining 1991).

Cheng et al (1996) and Husted (2000) suggested that economics play a vital role in determining software piracy. Other studies have however shown that low national income and low personal incomes are not the only reasons for which software is pirated. Swinyard, Rinne, and Kau (1990) observed that attitudes towards software piracy are affected by cultural standards and customs. Hence, "the neglect of culture as an explanation of software piracy seems odd given the fact that cultural values have such a significant impact on a wide array of business practices in different countries" (Husted, 2000, p. 200). Thus, this chapter will not only look into the relationship between economic factors and software piracy, but will also reflect on the cultural and ethical values and social norms that affect the trends of software piracy amongst students.

RESEARCH QUESTION

This study focuses on software piracy amongst university students – specifically with regard to its occurrence over the Internet, sharing (copying or borrowing) software on physical media such as floppy disks and CD-ROMs and buying pirated software from retail outlets[2]. This research explores the question whether software piracy behaviour among university students of a developing country can be conceptualized in terms of social and cultural norms rather than in terms of intentions as has been described (for piracy amongst university students) in most of the literature (e.g. Kwong & Lee, 2002; Rahim, Seyal & Rahman, 2001; Limayem, Khalifa & Chin, 1999; Tang & Farn, 2005; Gopal, Sanders & Bhattacharjee, 2004).

RESEARCH JUSTIFICATION

Empirical studies have been done on the subject of software piracy in different countries such as Saudi Arabia (Al-Jabri & Abdul-Gader, 1997), Thailand (Kini, Ramakrishna & VijayaRama, 2003; Leurkittikul, 1994), People's Republic of China (Wang, Zhang, Zang & Ouyang, 2005) and India (Gopal & Sanders, 1998). Other empirical studies include those done by Gan & Koh (2006); Ki, Chang & Khang (2006); and Mishra, Akman & Yazici (2006). Although Husted (2000) and Proserpio, Salvemini & Ghiringhelli (2004) included Pakistan as one of the countries in their respective analytical studies, empirical studies on software piracy issues of Pakistan do not exist in the literature. A comparative study can provide a means of highlighting differences and possible similarities of software piracy determinants between a developed and a developing country. Therefore Canada was chosen for this purpose as it is culturally and economically in contrast with Pakistan. Moreover, there hasn't been any recent Canadian scholarly literature[3] in this context. This research can therefore help fill a part of that void and the results can provide a better understanding of a developing country's software piracy issues that can help the policy makers to address the problem more effectively.

LITERATURE REVIEW

Intellectual Property and its Protection

According to Merriam-Webster's Dictionary of Law, intellectual property (IP) is a "property that derives from the work of the mind or intellect". Basically IP "refers to a legal entitlement which sometimes attaches to the expressed form of an idea, or to some other intangible subject matter"

(Wikipedia, 2006, para. 1). Moreover, IP law "regulates the ownership and use of creative works, including patent, copyright and trademark law" (Nolo, 2006). Simply put, intellectual property is a realization of someone's idea or thought. Composed music, lyrics, paintings, published written work, and software are the intellectual property of the artists or the professionals that produced or developed them.

Several authors, however, still debate the justifiability of the intellectual property laws (Siponen, 2004; Hettinger, 1989; Ladd, 1997; Stallman, 1995; Weckert, 1997). Although detailed discussion on their arguments is out of the scope of this research, it is important to have a broad view of some of the important concepts.

There are several IP protection organizations across the world, such as Business Software Alliance (BSA), Canadian Alliance Against Software Theft (CAAST), International Intellectual Property Alliance (IIPA), Software & Information Industry Association (SIIA), and World Intellectual Property Organization (WIPO). Many countries have their own intellectual property laws protecting the rights of individuals and organizations alike. The Copyright Act of Canada (Department of Justice Canada, 2006) provides protection to intellectual property in Canada. It considers computer programmes (or software) to be literary work and therefore has several laws controlling their unauthorized copying and distribution. Similarly, computer programmes are considered literary work under the Copyright Ordinance, 1962 of Pakistan. A significant amendment was made to this ordinance in 1992 called the Copyright (Amendment) Act, 1992. This amendment addressed the copyright issues of computer software in more detail than the original Copyright Ordinance. However, some laws are also considered debatable. For example, the Digital Millennium Copyright Act (DMCA) is usually seen as a controversial law approved by the U.S. Congress in 1998[4] (SearchCIO.com, 2006).

With society's transition to a digital world, copyright protection has become an important area of IP law (Blanke, 2004). It is evident from the discussion above that intellectual property rights hold immense importance in today's world but justification of IP rights and laws continues to be a debate among the subject experts. "On one hand are those who believe that anything they conjure up, anything that transforms an idea into form, is intellectual property. On the other are the individuals who believe just as passionately that the entire notion of intellectual property is at best a farce, at worst just another way to suck profits out of the ether" (Gantz & Rochester, 2004, p. xxiii). For example, Hettinger (1989), Ladd (1997), Stallman (1995) and Weckert (1997) view software as an intangible commodity and therefore favour its copying. Weckert reflected on intellectual property rights concerning software as unjustifiable. Hettinger holds similar views on IP rights and their protection. Hettinger suggested that patents and trade secrets are more difficult to justify than copyrights which "restrict only copying an expression of an idea" (Hettinger, 1989, p. 52). Siponen however argues that "it is fair and just for people to claim financial rewards for their creations" (Siponen, 2004, "Concluding remarks" section) and that respecting IP laws and rights is necessary for the society to live in harmony.

Software Piracy

Sims et al (1996) define software piracy as "the illegal copying of computer software" (p. 839). Copying software is easy and can be carried out in many forms. Moores (2003) identified common forms of software piracy as counterfeiting, Internet piracy, and softlifting. He further noted, that "counterfeiting and Internet piracy both involve creating bootlegged copies of licensed software for sale or distribution. Internet piracy makes use of the Internet to distribute the software, and has become a particular concern for vendor organizations" (Moores, 2003, p. 208). Softlifting

is also a very common type of software piracy among businesses that install single-user licensed software on multiple machines (Rahim, Seyal, & Abdul-Rahman, 2001; Simpson, Banerjee, & Simpson, 1994). Another kind of software piracy involves software installation by retailers onto the hard disk drives of customers' personal computers (PCs) in order to encourage the sale of hardware.

Can Software Piracy be Justified?

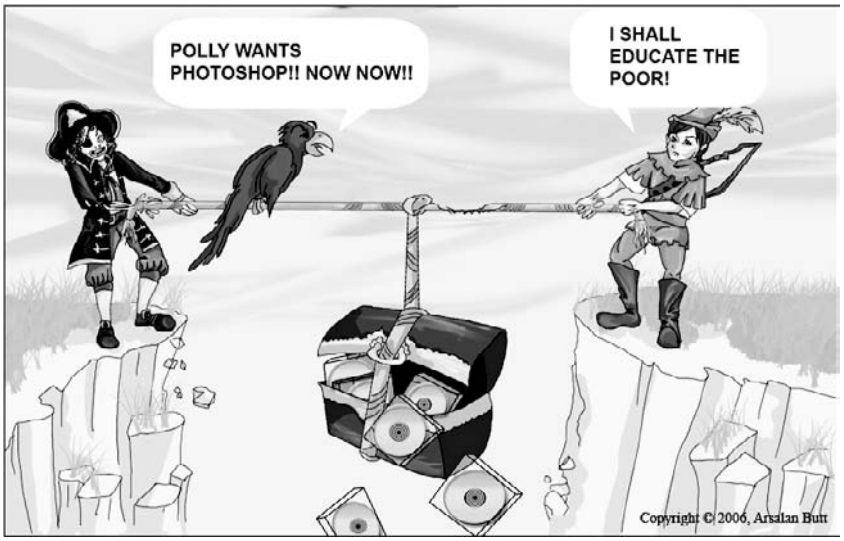
As is the case with intellectual property, issues surrounding software piracy are debated as well. For example, worldwide software piracy figures reported by BSA are cited by almost every published article on software piracy.[5] However, many authors consider BSA's methodology for calculating the levels of software piracy and the amount of reported monetary losses incurred as highly controversial (Locklear, 2004; The Economist, 2005). Even IDC (www.idc.com), an organization which has worked for BSA on the latter's several world software piracy reports, has commented that the conclusions presented in the 2004 BSA study were exaggerated (Locklear, 2004). Some authors, however, have argued that software piracy increases the popularity of the product itself as suggested by Slive and Bernhardt (1998) that "a software manufacturer may permit limited piracy of its software. Piracy can be viewed as a form of price discrimination in which the manufacturer sells some of the software at a price of zero" (p. 886).

A similar opinion was voiced by Microsoft Founder, Chairman, and Chief Software Architect Bill Gates in 1998. Gates reportedly said, "Although about three million computers get sold every year in China, people don't pay for the software. Someday they will, though. And as long as they're going to steal it, we want them to steal ours. They'll get sort of addicted, and then we'll somehow figure out how to collect sometime in the next decade" (CNN.com, 2001). This may also suggest that software manufacturers can allow initial piracy of their product as a strategy to enter or monopolize the market by making consumers attached to a particular software only (as could be the case for Microsoft's operating system Windows), since the purchasing power of the average consumer does not allow him/her to purchase the legal product at full price. Bill Gates faced a lot of criticism for his comments as Microsoft itself is probably the strongest advocate of the anti-software piracy campaign, the company's products being widely pirated all around the world.

Givon, Mahajan and Muller (1995) stated that "software piracy permits the shadow diffusion of a software parallel to its legal diffusion in the marketplace, increasing its user base over time. Because of this software shadow diffusion, a software firm loses potential profits, access to a significant proportion of the software user base, opportunities for cross-selling, and marketing its other products and new generations of the software. However, shadow diffusion may influence the legal diffusion of the software. Software pirates may influence potential software users to adopt the software, and some of these adopters may become buyers" (p. 1). LaRue (1985) also suggested that software publishers could eventually benefit by adopting a shareware marketing strategy of their software. This strategy is currently adopted by many software manufacturers and as Karon (1986) noted, the idea of such marketing strategies has also been supported by the president of an education software firm who believes that some pirates may eventually buy their products due to value-added benefits. Slive and Bernhardt (1998) also stated that piracy of software by home users can be viewed as a price discrimination strategy by the manufacturer (selling software for free) which will eventually increase the demand for the software by business users.

According to The Linux Information Project [LINFO] (2006), the critics of the concept of software piracy argue that the terminology associated

Figure 1. Can software piracy be justified?



with this concept is deliberately manipulated by the major commercial software developers. "That is, use of the term piracy itself is also highly controversial in a software context" (The Linux Information Project [LINFO], 2006, "Inappropriate Terminology" section, para. 1). And "this is also because it implies that people or organizations that create or use copies of programs in violation of their [end user licensing agreement] EULAs are similar to *pirates.* Pirates are violent gangs that raid ships at sea in order to steal their cargoes and rob their crews; they also frequently injure or kill the crews and sink their ships. Critics of this terminology claim that it was chosen for its dramatic public relations value rather than because of any relationship to the traditional use of the word" (LINFO, 2006, "Inappropriate Terminology" section, para. 2).

Another factor that has been shown to associate directly with computer abuse is called the Robin Hood Syndrome (Forester & Morrison, 1990; Perrolle, 1987; U.S. Department of Justice, National Institute of Justice, 1989a, 1989b). Harrington (2002) describes the Robin Hood syndrome as "the belief that harming a large organization to the benefit of an individual is the right behavior" (p. 180). In her study of software piracy, Harrington found that people high in Robin Hood Syndrome are more likely to pirate software as this syndrome allows an "individual to neutralize ethical judgments about software piracy and copy software offered for sale by large organizations" (p.181). The Robin Hood Syndrome could be applied in the context of developing countries as well, where software piracy is justified on the grounds "that it is unfair to charge prices in low income countries that are comparable to those in the higher income countries, and thus virtually unaffordable by most citizens and many businesses in such countries" (LINFO, 2006, "Reasons and Justifications For" section, point 4).

The purpose of this research is not to justify software piracy in a developing country such as Pakistan or any other part of the world. The authors believe that software piracy in any form is illegal and an unethical behaviour. However, it is equally important to stress the fact that depending upon the circumstances, individuals either inevitably *have to* indulge in this behaviour, reasons for which will be discussed later; or they have the option of pirating software, that is to say they do it because they *can*. Another important clarification

that needs to be established at this point is that this research mainly focuses on individual piracy rather than commercial piracy (organizations producing pirated software on a large scale for selling purposes) which is purely done for profit. Commercial piracy however is a crucial element that creates a piracy facilitating environment in a society and will therefore be discussed where relevant. However as stated earlier, the focal point of this research is individual piracy by university students.

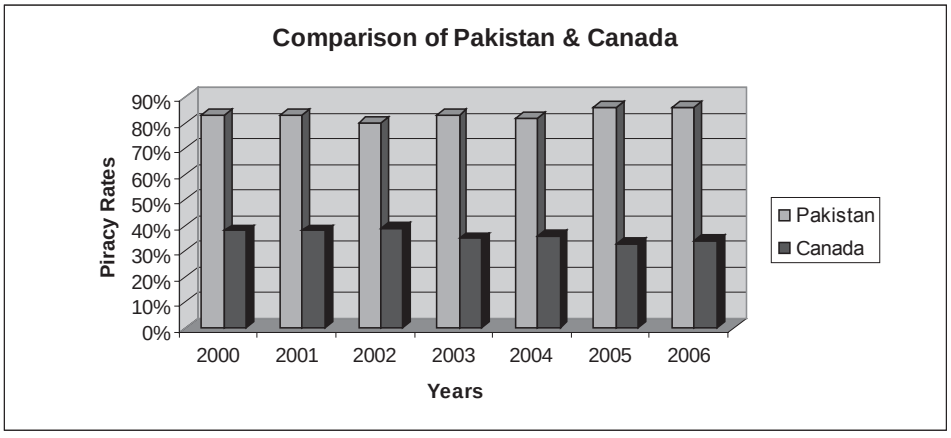
Software Piracy in Pakistan and Canada

Developed nations such as U.S. or Canada have anti-piracy policies and organizations to control unauthorized publishing or copying of software. However, "developing countries are passive in addressing computer ethics in general and intellectual property rights in particular" (Al-Jabri et al., 1997, p. 335). Al-Jabri et al. also suggested that developing countries lack interest groups that combat software piracy. However, it has also been observed that even the presence of these organizations and the existing copyright laws of the country cannot make a significant difference in developing countries (IIPA, 2004). According to IIPA A (2006) Canada also falls short in meeting the objectives laid down in the WIPO Copyright Treaty (WCT) and the WIPO Performances and Phonograms Treaty (WPPT). IIPA notes that the Canadian government introduced Bill C-60 in order to comply with these treaties but the bill eventually died as a result of a call for federal elections in November 2005. IIPA further points out that "Canada remains far behind virtually all of its peers in the industrialized world with respect to its efforts to bring its copyright laws up to date with the realities of the global digital networked environment. Indeed, most of the major developing countries have progressed further and faster than Canada in meeting this challenge" (IIPA, 2006, p. 2). As per the IIPA recommendation, Canada remains on the Watch List of countries. As indicated in Figure 2, the software piracy rates have remained nearly constant over the past few years in Pakistan and Canada.

RESEARCH MODEL

Theory of reasoned action (TRA) suggested by Fishbein & Ajzen (1975) and theory of planned behaviour (TPB) later developed by Ajzen (1991) have been used extensively in the literature to

Figure 2. Software piracy levels in Pakistan & Canada (Source: BSA 2005, 2007)



explain software piracy behaviour and intentions. Both of these theories look at behaviour as an intentional act. While it is true that the literature on software piracy (significant portion of which is based on TRA and TPB) has helped in understanding various aspects of the matter, there have been no empirical studies to prove that software piracy can be conceptualized as an *unintentional behaviour* or *as a behaviour that is the product of the social and cultural environment within which the behaviour is carried out.*

The model developed for this research[6] is shown in Figure 3. It includes social norms as one of the variables. The basic structure of this model has been adopted and modified from a model that was used by Proserpio et al. (2004). Their model was based on a multi-causality approach to determine software piracy factors in 76 countries (including Pakistan and Canada) and is therefore appropriate for this research.

RESEARCH HYPOTHESES

Economic Development

According to Marron and Steel (2000), intellectual property rights encourage novelty and economic growth. In a study on the relationship between national economy and piracy levels, they concluded that a strong inverse correlation existed between piracy rates and the income level of the country. Bezmen & Depken (2006) also found a negative relationship between software piracy and income, tax burdens, and economic freedom. Therefore, it is hypothesized that:

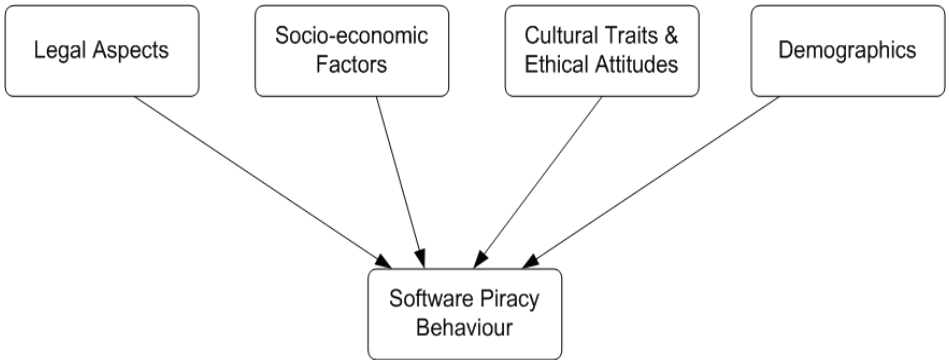
H1: Income will have an inverse influence on piracy intention of subjects.

H2: High price of original software will have a direct influence on the piracy intention of subjects.

Culture

Culture has been defined as "the collective programming of the mind which distinguishes the members of one group or category of people from another" (Hofstede, 1997, p.260); yet culture is a very broad concept, and has little power if it is used as a residual category (Child, 1981). Marron & Steel (2000) and Shin et al. (2004) concluded that the collectivistic culture is to blame for the high software piracy rates. Therefore, social norms and culture will be taken into consideration and the following is hypothesized.

Figure 3. Software piracy behavior model (Source: Butt, 2006)



H3: Social/Cultural norms will have a direct influence on the piracy behaviour of subjects.

Other Piracy Facilitating Factors

Besides the effects of social and cultural norms and poor economy on software piracy, the availability of pirated products is a very important factor that could be significantly related to higher piracy rates. For instance, Rainbow Market in Karachi, and Hafeez Centre in Lahore are two of the biggest and best-known hardware and software malls in Pakistan. Each of these malls comprises hundreds of retailers selling pirated software. The Canadian software market's situation is totally different from the one described above. There are few, if any retailers openly selling pirated software. However, according to a CAAST news release (CAAST, 2005), forty seven percent of the surveyed students admitted to pirating software.

Moores & Dhaliwal (2004) suggested that *high availability of illegal software, absence of legal punishments and high cost of legal software* reflect the high piracy rates of the regions being studied. Simpson et al. (1994) also included the element of legal factors in the piracy model they proposed, and found that these factors have an effect on the ethical decision process, which leads to the actual piracy behaviour. The following is therefore hypothesized in regards to the availability and legal factors.

H4: There will be a positive influence of the availability of pirated software on the intent of subjects.

H5: Legal enforcement will have a negative influence on the piracy intent of the subjects.

H6: Legal enforcement will have a positive influence on the social norms.

Previous studies have also suggested that gender is an important demographic factor that affects one's intention to pirate software. In a survey of moral intentions towards software piracy, Kini et al. (2003) found that males were more inclined towards pirating software, and similar results were proposed by Higgins and Makin (2004). Therefore, the following is considered as the null hypothesis for gender.

H7: There will be no difference between males and females regarding their software piracy behaviour.

"People's perceptions of a particular behaviour are shaped by the existing value system of the society" (Lau, 2003, p. 234). The decision-making model proposed by Jones (1991) suggested that an individual's attitude toward ethical issues will affect the individual's ethical judgement and then their ethical behaviour. It is therefore hypothesized that:

H8: There will be a direct relationship between attitudes towards piracy and the piracy behaviour of subjects.

The discussion that has been presented so far in this research elaborates on the fact that current literature regards piracy behaviour as intentional. To conform to the current literature, the following final hypothesis is made.

H9: Intention to pirate software will have a positive influence on the actual piracy behaviour of subjects.

RESEARCH METHODOLOGY

Site Selection

As is the case with many research projects, this study also had limited resources in terms of time and money. The sites for the study were therefore chosen with these factors taken into consideration.

For the Canadian part of the study, the authors' home university was chosen. For the Pakistani study, the city of Lahore was chosen since it has one of the biggest pirated software markets in Pakistan and also has several IT institutions. Five universities were chosen in Lahore.

Sampling Characteristics

Students were chosen as the target population in order to conform to the existing research, most of which is based on samples of college and university students. Students at both undergraduate and graduate levels from information technology and computer science departments were included in this study.

Pilot & Actual Studies

A self-administered survey instrument/questionnaire was developed. This questionnaire consisted of closed-ended questions that were used to collect demographic details about the research participants. The questionnaire also consisted of, 31 items, each rated on a 5-point Likert scale to assess respondents' attitude towards ethical, economical and demographic implications of software piracy. Negatively worded items were included to detect response patterns. Various items in the questionnaire were adopted from current literature including Moores & Dhaliwal (2004), Siegfried (2005) and Al-Jabri et al. (1997). Based on the feedback of pilot study (conducted in Canada), minor changes were made to the format and content of the questionnaire, and it was also modified to make it adaptable in Pakistan.

While conducting the study in Pakistan, hard copies of the questionnaire were physically distributed at the same time in four classrooms and one computer laboratory at each of the five universities. The questionnaire at the university in Canada was administered through the Internet with the use of a secure program written in PHP/CGI to capture responses.

DATA ANALYSIS

Descriptive Statistics

The online survey conducted at the university in Canada returned 208 responses out of which 196 were usable. Most of the Canadian respondents were under the age of 26 (n=172, 88%).[7] There were 122 (62%) males and 74 (38%) female respondents. The survey in five Pakistani universities returned 365 responses out of which 339 (n=339) were usable. As was the case in Canadian data, most of the respondents were under the age of 26 (n = 325, 96%). There were 221 (65%) males and 118 (35%) female respondents.

Hypotheses Testing

For testing hypothesis, the questionnaire items in both Pakistani and Canadian questionnaire were grouped together to make the statistical tests feasible. The groupings (Table 1) were made based on (1) the face validity, i.e. interpretability; (2) factor loadings; and (3) reliability aka Cronbach's alpha.

Structural Equation Modelling (SEM) with LISREL was used to test the relationships between the above groups/variables. Based on composite scores, the *'Norm-attd'* group in the Pakistani data was further split into '*socnorm*' for *'social norms*' and '*attit*' for *'attitude'*. In the SEM for the Pakistani group, a latent variable called ***'sociomor'*** was created which was composed of '*socnorm*' and '*attit*'**.**

Fitting Data on Structural Models

In addition to fit statistics, structural equation modelling (SEM) produces estimates for partial regression coefficients (referred to as path coefficients), standardized regression coefficients and estimates of squared multiple correlations" (Wagner & Sanders, 2001, p. 165). LISREL was used to fit the data on a structural equation model.

Table 1. Composite variables in the study[8]

Pakistani Study – Variable (Group) Names	Canadian Study – Variable (Group) Names
Availability	Availability
Legal	Legal
Intent	Intent
Norm-attd (i.e. Norms and socially or culturally mediated attitudes grouped together)	Ethical beliefs and Attitudes
Price	Price
Piracy Behaviour	Piracy Behaviour
-	Norms

The resulting path coefficients for Pakistan and Canada are shown in Figure 4.[9] The structural models developed in this research represent the direction and strength of independent variables' relationship with dependent variables.

In the Pakistani model, the price factor (price → piracy behaviour, 0.03) does not seem to have any effect at all on the piracy behaviour, therefore rejecting hypothesis 2. Price also has a fairly strong negative relationship (-.047) with the intent variable, thus contradicting hypothesis 1. In the Canadian model, price has very weak relationships with intent (.01) and piracy behaviour (.02), therefore rejecting hypotheses 1 and 2 for Canadian students. Legal issues have a fairly strong influence on intent (0.54) and not so strong (although it is in the right direction) on the *sociomor* variable (0.40) in the Pakistani model. Hypothesis 5 and hypothesis 6 are accepted and rejected respectively. The Canadian legal construct has a weak relationship with intent (0.07) and a fairly strong relationship with ethical beliefs and attitudes (0.46), thus rejecting hypothesis 5 but accepting hypothesis 6. The gender →piracy behaviour paths in both models have very weak coefficients indicating no difference in piracy behaviour between males and females, thus accepting hypothesis 7.

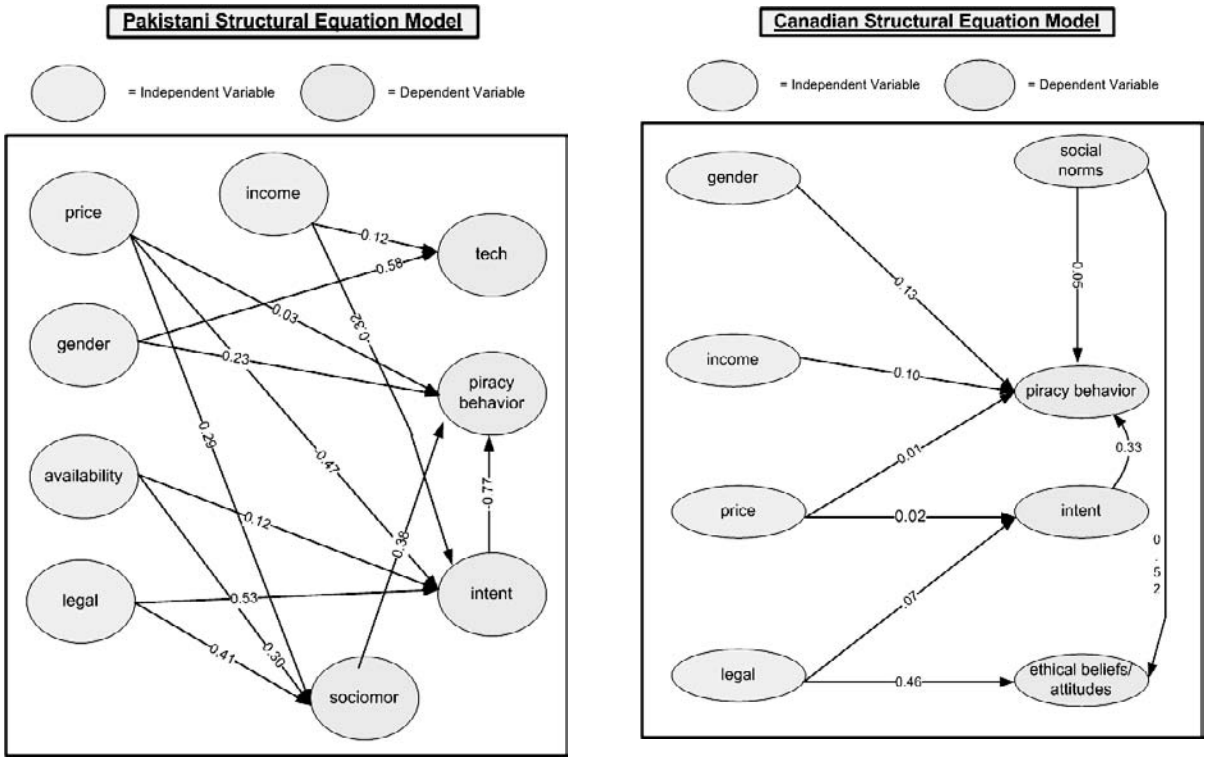
The availability of pirated software has a very small effect on the intent of Pakistani (0.13) and Canadian (0.23) students rejecting hypothesis 4. Intentions in the Pakistani model have a very high negative relationship (-0.76) with the piracy behaviour. Hypothesis 9 is therefore rejected. The *sociomor* construct, despite having a positive relationship (0.39) with the piracy behaviour, is not strong enough to accept hypotheses 3 and 8 in the Pakistani model. As far as the Canadian model is concerned, the *norms* variable has a strong influence (0.52) on ethics and attitudes toward piracy but negligible effect on piracy behaviour with a path coefficient of 0.05. Intent on the other hand shows a relationship in the correct direction (though not very strong) with a path coefficient of 0.34 with the piracy behaviour of the students.

DISCUSSION

This research has focused on the cultural dimension of software piracy and its effect on the behaviour of university students. Two structural models that incorporate social and cultural norms, economic conditions, ethical attitudes towards piracy and the availability of software piracy have been developed and tested. Since Canada and Pakistan are culturally and economically different, they were chosen to provide a contrasting view of the software piracy phenomenon.

The IT industry in Pakistan is progressing, though not at a rapid pace. Still in its infancy, it began in Pakistan with the introduction of

Figure 4. Pakistani and Canadian SEMs



the Internet in 1996. The analysis of economic factors (high price of legal software and low income) in this study provides a rationale for the reluctance of Pakistani government in enforcing intellectual property rights. Despite being aware of the rampant software piracy, governments of countries such as Pakistan are aware of the economic conditions of the mass population. People (students, in the context of this research) in the developing countries need to have cheap access to resources (software) in order to keep up with the rapid pace of technological advancement in the developed world. It can therefore be assumed that governments of developing countries are aware of this and therefore are always reluctant to enact and enforce strict IP protection laws.

The empirical evaluation provides support that social norms and positive attitudes towards piracy are correlated with the actual piracy behaviour of Pakistani students.[10] This finding is similar to that of Proserpio et al., (2004), Seale et al., (1998) and Al-Jabri, I. & Abdul-Gader (1997). Intentions proved to be stronger predictors of piracy behaviour of Canadian students and this finding conforms to the literature which regards piracy behaviour as intentional. The achieved results indicate that software piracy behaviour in Pakistan cannot be regarded as purely intentional. It should rather be conceptualized as a consequential behaviour resulting from various elements, with customs or social norms being the strongest of them all. This also indicates that in two culturally different countries, the conditions that are responsible for creating a piracy-favouring environment are essentially different. Due to a lack of IP related awareness (unlike the developed world), this culture of copyright infringement is deeply rooted in the Pakistani society in such a way that students buy or share pirated software without even realizing that their action might

be considered illegal and/or unethical. It is an established norm: a custom; the way an act is supposed to be normally carried by a member of the society.

Gopal and Sanders (1998) correctly identified the need for a behavioural model for software piracy activity that would help software publishers gain insight into the behavioural dynamics of software pirates. However, as they also found that their economical model was appropriate for the U.S. and not for India, caution should be practiced in all future research that attempts to study piracy behaviour. This study was an exploratory, cross-cultural investigation of piracy in two very different cultures. The applicability of Western constructs such as *'attitudes'* and *'intentions'* to collectivist societies must always be critically examined. Future research should look at the questions left unanswered by this study. Subjects from more countries should be included in future cross-country studies of software piracy behaviour so that the results of this study could not only be generalized for the general student population but also to the population at large.

Authors of this chapter feel necessary to reflect upon the fact that a discovery or invention of a new technology cannot be blamed for its eventual uses. Albert Einstein reportedly said, "My name is linked to the atomic bomb in two different ways. Almost fifty years ago I discovered the equivalence of mass and energy, a relationship which served as the guiding principle in the work leading to the release of atomic energy. Secondly, I signed a letter to President Roosevelt, stressing the need for work in the field of the atomic bomb. I felt this was necessary because of the dreadful danger that the Nazi regime might be the first to come into possession of the atomic bomb" (Nathan & Norden, 1960, p. 569). Einstein however considered signing the letter a "great mistake" (Tobar-Arbulu, 1984). Technology itself can therefore not be blamed for its eventual uses.

FUTURE TRENDS

Findeli (1994) correctly argued that technoethical issues have been continuously rising from the unparalleled developments in technology. One could argue that such issues can act as catalysts to invoke other socio-political and ethical dilemmas. It is not only in software or computer technology that further development will raise ethical questions, but other walks of life such as medicine, engineering, journalism, etc. can have equally grave issues: human cloning, embryonic stem cells research, and inappropriate use of media to shape public opinion during conflicts (e.g. in a war), are only a handful of examples. It is a matter of time before such conflicts will give birth to new unforeseen technoethical issues.

CONCLUSION

This study has found that there are more ways than one of understanding piracy behaviour across different countries. Although poor national economy plays a substantial role in software piracy rates, culture is also part of the equation. This study has also suggested that software piracy behaviour in a developing country such as Pakistan should not be conceptualized in terms of intentions alone. Caution should be practiced in all future research that attempts to study piracy behaviour as the applicability of Western constructs such as *'attitudes'* and *'intentions'* to collectivist societies must always be critically examined. Subjects from more countries should be included in future cross-country studies of software piracy behaviour so that the results of this study could not only be generalized for the general student population but also for the population at large. There is also a lack of longitudinal research and also of other forms of electronic piracy, such as the availability of pirated e-books on the Internet.

Future research could therefore attempt to study both of these domains as well.

REFERENCES

Ajzen, I. (1991). The theory of planned behavior. *Organizational Behavior and Human Decision Processes, 50*(2), 179-211.

Al-Jabri, I. & Abdul-Gader, A. (1997). Software copyright infringements: An exploratory study of the effects of individual and peer beliefs. *Omega, 25*, 335-344.

Bezmen, TL. & Depken, CA. (2006). Influences on software piracy: Evidence from the various United States. *Economic Letters, 90*, 356-361.

Blanke, JM. (2004). Copyright law in the digital age. In Brennan, L. L. & Johnson, V.E. (Eds.), *Social, ethical and policy implications of information technology*, (pp. 223-233). Hershey, PA: Idea Group Inc.

Business Software Alliance (BSA). (2005). *Second annual BSA and IDC global software piracy study.* Retrieved January 7, 2006 from http://www.bsa.org/globalstudy/upload/2005-Global-Study-English.pdf

Business Software Alliance (BSA). (2007). *Fourth annual BSA and IDC global software piracy study.* Retrieved July 27, 2007 from http://www.ifap.ru/library/book184.pdf

Butt, A. (2006). A cross-country comparison of software piracy determinants among university students: Demographics, ethical attitudes & socio-economic factors, emerging trends and challenges in information technology management. *Proceedings of the 2006 Information Resources Management Association International Conference.* Hershey: Idea Group Publishing.

Canadian Alliance Against Software Theft (CAAST). (2005). *Software piracy runs rampant on canadian university campuses.* Retrieved Nov 1, 2005 from http://www.caast.org/release/default.asp?aID=139

Cheng, HK. Ronald, RS. & Hildy, T. (1997). To purchase or to pirate software: An empirical study. *Journal of Management Information Systems, 13*(4), 49-60.

Child, J. (1981). Culture, contingency and capitalism in the cross-national study of organization. *Research in Organization Behavior, 3*, 303-356.

Christensen, AL. & Martha, M.E. (1991). Factors influencing software piracy: Implications for accountants. *Journal of Information Systems, 5* (spring), 67-80.

CNN.com. (2001). *Microsoft in China: Clash of titans.* Retrieved March 19, 2006 from http://archives.cnn.com/2000/TECH/computing/02/23/microsoft.china.idg/

Department of Justice Canada (2006). *Copyright Act, R.S., 1985, c. C-42.* Retrieved March 4, 2006 from http://laws.justice.gc.ca/en/C-42/index.html

Findeli, A. (1994). Ethics, aesthetics, and design. *Design Issues, 10*(2), 49-68.

Fishbein, M., & Ajzen, I. (1975). *Belief, attitude, intention and behavior: An introduction to theory and research.* Reading, MA: Addison-Wesley.

Forester, T. & Morrison, P. (1990). *Computer ethics: Cautionary tales and ethical dilemmas in computing.* Cambridge, MA: The MIT Press.

Freeman, L. A. & Peace, A. G. (2005). Revisiting Mason: The last 18 years and onward. In Freeman, L. A. & Peace, A.G. (Eds.), *Information ethics: Privacy and intellectual property*, (pp. 1-18). Hershey, PA: Idea Group Inc.

Gantz, J. & Rochester, J. B. (2004). *Pirates of the digital millennium.* Upper Saddle River, NJ: Prentice Hall.

Gan LL. & Koh. HC. (2006). An empirical study of software piracy among tertiary institutions in Singapore. *Information & Management, 43,* 640-649.

Givon, M., Mahajan, V. & Muller, E. (1995). Software piracy: Estimation of lost sales and the impact on software diffusion. *Journal of Marketing, 59*(1), 29-37.

Gopal, R. D. & Sanders, G. L. (1998). International software piracy: Analysis of key issues and impacts. *Information Systems Research*, *9*(4), 380-397.

Gopal, R. D., Sanders, G. L. & Bhattacharjee, S. (2004). A behavioral model of digital music piracy. *Journal of Organizational Computing and Electronic Commerce, 14*(2), 89-105.

Harrington, S. J. (2002). Software piracy: Are robin hood and responsibility denial at work? In Salehnia, A. (Ed.), *Ethical issues of information systems*, (pp. 177-188). Hershey, PA: IRM Press.

Hettinger, E. C. (1989). Justifying intellectual property. *Philosophy and Public Affairs, 18*(1), 31-52.

Higgins, G. E. & Makin, D. A. (2004). Does social learning theory condition the effects of low self-control on college students' software piracy? *Journal of Economic Crime Management, 2*(2), 1-22.

Hinduja, S. (2001). Correlates of internet software piracy. *Journal of Contemporary Criminal Justice, 17*(4), 369-82.

Hofstede, G. (1997), Cultures and organizations: Software of the mind. New York: McGraw Hill.

Husted, B. W. (2000). The impact of national culture on software piracy. *Journal of Business Ethics, 26,* 197-211.

International Intellectual Property Alliance (IIPA). (2004). *2004 special 301 report: Pakistan*. Retrieved November 20, 2005 from http://www.iipa.com/rbc/2004/2004SPEC301PAKISTAN.pdf

International Intellectual Property Alliance (IIPA). (2006). *2006 special 301: Canada*. Retrieved March 21, 2006 from http://www.iipa.com/rbc/2006/2006SPEC301CANADA.pdf

Jones, T. M. (1991). Ethical decision making for individuals in organizations: An issue contingent model. *Academy of Management Review, 16*(February), 366-395.

Karon, P. (1986). Software industry groups set sail against pirates in academe. *PC Week*, 9 December, 62.

Ki E., Chang B., & Khang K. (2006). Exploring influential factors on music piracy across countries. *Journal of Communication, 56,* 406-426.

Kini, R. B., Ramakrishna, H.V. & Vijayaraman, B. S. (2003). An exploratory study of moral intensity regarding software piracy of students in Thailand. *Behavior & Information Technology, 22*(1), 63-70.

Kwong, T. C. H. & Lee, M. K. O. (2002). Behavioral intention model for the exchange mode internet music piracy. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences* -Volume 7, 191. Washington, DC, USA.

Ladd, J. (1997). Ethics and the computer world: A new challenge for philosophers. *ACM Computers & Society, 27*(3), 8-13.

Lau, E. K. W. (2003). An empirical study of software piracy. *Business Ethics, 12*(3), 233-245.

LaRue, J. (1985). Steal this program. *Library Software Review, 4*(5), 298-301.

Leurkittikul, S. (1994). *An empirical comparative analysis of software piracy: The United States*

and Thailand. Unpublished doctoral dissertation, Mississippi State University.

Limayem, M., Khalifa, M., & Chin, W. W. (1999). Factors motivating software piracy: A longitudinal study. *International Conference on Information systems* (pp. 124-131). Association for Information Systems.

Locklear, F. Z. (2004). *IDC says piracy loss figure is misleading.* Retrieved March 21, 2006 from http://arstechnica.com/news.ars/post/20040719-4008.html

Mason, R. O. (1986). Four ethical issues of the information age. *MIS Quarterly, 10*(1), 5-12.

Marron, D. B. & Steel, D. G. (2000). Which countries protect intellectual property? The case of software piracy. *Economic Inquiry, 38*, 159-174.

Mishra, A. Akman, I. & Yazici, A. (2006). Software piracy among IT professionals in organizations. *International Journal of Information Management, 26*(5), 401-413.

Moores, T. T. (2003).The effect of national culture and economic wealth on global software piracy rates. *Communications of the ACM, 46*(9), 207-215.

Moores, T. T. & Dhaliwal, J. (2004). A reversed context analysis of software piracy issues in Singapore. *Information & Management, 41*, 1037-1042.

Nathan, O. & Norden, H. (1960) *Einstein of peace*. New York: Simon and Schuster.

Nolo. (2006). *Intellectual property.* Retrieved March 19, 2006 from http://www.nolo.com/definition.cfm/Term/519BC07C-FA49-4711-924FD-1B020CABA92/alpha/I/

Perrolle, J. (1987). *Computers and social change: Information, property, and power.* Belmont, CA: Wadsworth Publishing.

Proserpio, L., Salvemini, S. & Ghiringhelli, V. (2004). Entertainment pirates: Understanding piracy determinants in the movie, music and software industries. *The 8th International Conference on Arts & Cultural Management.* Retrieved January 7, 2006 from http://www.hec.ca/aimac2005/PDF_text/ ProserpioL_SalveminiS_GhiringhelliV.pdf

Rahim, M. M., Seyal, A. H. & Abd. Rahman, M. N. (2001). Factors affecting softlifting intention of computing students: An empirical study. *J. Education Computing Researc*h, *24*(4), 385-405.

Seale, D. A., Polakowski, M., & Schneider, S. (1998). It's not really theft: Personal and workplace ethics that enable software piracy. *Behavior & Information Technology, 17,* 27-40.

SearchCIO.com. (2006). *Digital Millennium Copyright Act.* Retrieved March 19, 2006 from http://searchcio.techtarget.com/sDefinition/0,290660,sid19_ gci904632,00.html

Seyoum, B. (1996). The Impact of intellectual property rights on foreign direct investment. *Columbia Journal of World Business*, 31[1], 50. Elsevier Science Publishing Company, Inc.

Siegfried, R. M. (2004). Student attitudes on software piracy and related issues of computer ethics. *Ethics and Information Technology, 6*, 215–222

Simpson, P. M., Banerjee, D. & Simpson Jr., C. L. (1994). Softlifting: A model of motivating factors. *Journal of Business Ethics, 13*(6), 431-438.

Sims, R. R., Cheng, H. K., & Teegen, H. (1996). Toward a profile of student software piraters. *Journal of Business Ethics, 15*, 839-849.

Siponen, M. (2004). A justification for software rights. *ACM Computers and Society, 34*(3), 3.

Slive, J. & Bernhardt, D. (1998). Pirated for profit. *Canadian Journal of Economics, 31*(4), 886-899.

Stallman, R. (1995). Why software should be free. In Johnson, D. & Nissenbaum, H. (Eds.), *Computers, Ethics & Social Values,* (pp. 190-199). Englewood Cliffs, NJ: Prentice Hall.

Swinyard, W. R., Rinne, H., & Kau, A. K. (1990). The morality of software piracy: A cross-cultural analysis. *Journal of Business Ethics*, *9*(8), 655-664.

Tang, J. & Fam, C. (2005). The effect of interpersonal influence on softlifting intention and behavior. *Journal of Business Ethics*, *56*, 149-161.

The Copyright Ordinance, 1962 (of Pakistan). Retrieved November 11, 2005 from http://www.pakistanlaw.com/Copyright_Ordinance_1962.php

The Economist (2005). *Dodgy software piracy data*. Retrieved March 21, 2006 from http://www.economist.com/research/articlesBySubject/displayStory.cfm?story_ID=3993427&subjectid=1198563

The Linux Information Project (LINFO). (2006). *The "software piracy" controversy.* Retrieved February 27, 2006 from http://www.bellevuelinux.org/ software_piracy.html

Tobar-Arbulu, J. F. (1984). Plumbers, technologists, and scientists. *Research in Philosophy and Technology*, *7*, 5-17.

Tobar-Arbulu, J.F. (n.d.). *Technoethics*. Retrieved September 23, 2007 from http://www.euskomedia.org/PDFAnlt/riev/3110811103.pdf

U. S. Department of Justice, National Institute of Justice (1989a). In J. T. McEwen (Ed.), *Dedicated computer crime units.* Washington, DC: U.S. Government Printing Office.

U. S. Department of Justice, National Institute of Justice (1989b). In D. B. Parker (Ed.), *Computer crime: Criminal justice resource manual* (2nd ed.). Washington, DC: U.S. Government Printing Office.

Wang, F., Zhang, H., Zang, H. & Ouyang, M. (2005). Purchasing pirated software: An initial examination of chinese consumers. *Journal of Consumer Marketing*, *22*(6), 340-351.

Weckert, J. (1997). Intellectual property rights and computer software. *Journal of Business Ethics*, *6*(2), 101-109.

Wikipedia - The free encyclopaedia. (2006). *Intellectual property*. Retrieved March 16, 2006 from http://en.wikipedia.org/wiki/Intellectual_property

KEY TERMS

Culture: Source: Hofstede (1997): "The collective programming of the mind which distinguishes the members of one group or category of people from another."

Individualism and Collectivism: Source: www.Geert-Hofstede.com, An individualistic society is one in which the ties between individuals are loose: everyone is expected to look after him/herself and his/her immediate family. On the collectivist side, we find societies in which people from birth onwards are integrated into strong, cohesive in-groups, often extended families (with uncles, aunts and grandparents) which continue protecting them in exchange for unquestioning loyalty. The word 'collectivism' in this sense has no political meaning: it refers to the group, not to the state. Again, the issue addressed by this dimension is an extremely fundamental one, regarding all societies in the world.

Intellectual Property: Source Forester & Morrison (1990): Results of intellectual activity in the industrial, scientific, literary or artistic fields.

P2P: Source (www.tech-faq.com): Peer-to-peer (P2P) file sharing is a system of sharing files

directly between network users, without the assistance or the interference of a central server.

Software Piracy: Unauthorized duplication of computer for personal and/or commercial purposes. Types of software piracy addresses in this chapter (Source: www.siia.net):

a. **Softlifting:** It occurs when a person purchases a single licensed copy of a software program and loads it on several machines, in violation of the terms of the license agreement. Typical examples of softlifting include, "sharing" software with friends and co-workers and installing software on home/laptop computers if not allowed to do so by the license. In the corporate environment, softlifting is the most prevalent type of software piracy - and perhaps, the easiest to catch.
b. **Hard-Disk Loading:** It occurs when an individual or company sells computers preloaded with illegal copies of software. Often this is done by the vendor as an incentive to buy certain hardware.
c. **CD-R Piracy:** It is the illegal copying of software using CD-R recording technology. This form of piracy occurs when a person obtains a copy of a software program and makes a copy or copies and re-distributes them to friends or for re-sale. Although there is some overlap between CD-R piracy and counterfeiting, with CD-R piracy there may be no attempt to try to pass off the illegal copy as a legitimate copy - it may have hand-written labels and no documentation at all.
d. **Internet Piracy:** It is the uploading of commercial software (i.e., software that is not freeware or public domain) on to the Internet for anyone to copy or copying commercial software from any of these services. Internet piracy also includes making available or offering for sale pirated software over the Internet. Examples of this include the offering of software through an auction site, IM, IRC or a warez site. Incidences of Internet piracy have risen exponentially over the last few years.

ENDNOTES

1 For a detailed discussion on technoethics, see Tobar-Arbulu (n.d.) and Findeli (1994).
2 These types of software piracy are defined at the end of this chapter.
3 There was only one Canadian empirical (scholarly) study found in the literature (see Limayem et al., 1999). This study however relied on 98 research participants only and therefore cannot be considered very extensive.
4 For more discussion on the controversial aspects of DMCA, see http://www.eff.org/IP/DMCA/20020503_dmca_consequences.pdf; http://chronicle.com/free/v48/i47/47b00701.htm
5 This chapter also cites BSA figures but does not rely on any one of them as a major argument.
6 It is important to emphasize here that the model shown in Figure 1 presents a very basic structure which represents the theoretical base of this research.
7 All percentages are rounded off.
8 Factors with Cronbach's alpha >= .70 were only retained.
9 The positive paths between two variables indicate a positive relationship between them and vice versa. The closer the coefficient is to 1, the stronger the relationship. Authors feel the need to emphasize that the structural models were modified until an acceptable fit was achieved. The model was modified because (1) some of the independent variables had significantly non-normal distributions and (2) the relatively small sample size in the

Canadian case made parameter estimation in more complex models more difficult. This is due to the under-identification problems that often arise when the number of degrees of freedom—a function of sample size and number of free parameters—is small. In SEM, structural coefficients between observed variables and latent variables and between latent variables and other variables are parameters to be estimated. For these reasons as well as for parsimony, a less complex SEM model was adopted.

[10] Absence of very strong relationships in SEMs can be attributed to small sample sizes in both studies. Nonetheless, the direction of relationships supported the underlying hypothesis: intentional vs. un-intentional piracy behaviour.

This work was previously published in the Handbook of Research on Technoethics, edited by R. Luppicini and R. Adell, pp. 354-372, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 7.13
Legal and Economic Justification for Software Protection

Bruno de Vuyst
Vrije Universiteit Brussel, Belgium

Alea Fairchild
Vrije Universiteit Brussel, Belgium

ABSTRACT

This chapter discusses legal and economic rationale in regards to open source software protection. Software programs are, under TRIPS[1], protected by copyright (reference is made to the Berne Convention[2]). The issue with this protection is that, due to the dichotomy idea/expression that is typical for copyright protection, reverse engineering of software is not excluded, and copyright is hence found to be an insufficient protection. Hence, in the U.S., software makers have increasingly turned to patent protection. In Europe, there is an exclusion of computer programs in Article 52 (2) c) EPC (EPO, 1973), but this exclusion is increasingly narrowed and some call for abandoning the exclusion altogether. A proposal by the European Commission, made in 2002, called for a directive to allow national patent authorities to patent software in a broader way, so as to ensure further against reverse engineering; this proposal, however, was shelved in 2005 over active opposition within and outside the European parliament. In summary, open source software does not fit in any proprietary model; rather, it creates a freedom to operate. Ultimately, there is a need to rethink approaches to property law so as to allow for viable software packaging in both models.

INTRODUCTION

Copyright Protection of Software

A software program is foremost a sequence of orders and mathematical algorithms emerging from the mind of the innovator, hence creating

a link with copyright law as a prime source of intellectual property protection.

According to Article 10 TRIPS, computer programs, whether in source or object code, shall be protected as literary works under the Berne Convention provided that they are (1) original and (2) tangible. In light of Article 9 TRIPS, which states that copyright protection shall extend to expressions, but not to ideas, procedures, methods of operation or mathematical concepts as such, copyright protects the actual code of the computer program itself, and the way the instructions have been drawn up, but not the underlying idea thereof (Overdijk, 1999).

Hence, an author can protect his original work against unauthorized copying. Consequently, an independent creation from another person would not automatically be seen as a copyright infringement (Kirsch, 2000a; Leijnse, 2003). With respect to software programs this could have as consequence that a person disassembles and decompiles an existing software program to determine the underlying idea and uses this idea to build his own program (reverse engineering). As he only uses the idea, which is not copyrightable, no infringement will result.

BACKGROUND

Patent Law Protection of Software

Software is a novel form in the technology world, and may make a claim to patent protection from that angle. The conditions to be met to enjoy patent protection are more stringent than those to enjoy copyright protection. In Europe[3], for example, an invention will enjoy protection from patent law provided that the invention (1) is new (i.e., never been produced before), (2) is based on inventor activity (i.e., not have been before part of prior art), and (3) makes a technical contribution (i.e., contribute to the state of the art). In the U.S., the patent requirements to be met are (1) novelty, (2) non-obviousness, and (3) the innovations must fall within the statutory class of patentable inventions.

Pursuant to patent law, a patent holder can invoke the protection of his patent to exclude others from making, using or selling the patented invention. As opposed to copyright protection, the inventor's patent is protected regardless whether the software code of the patented program was copied or not.

The Evolution of the Legal Protection of Software

Prior to the 1980s, U.S. courts unanimously held that software was not patentable and that its only protection could be found in copyright. Indeed, the U.S. Supreme Court ruled in two landmark decisions, Gottschalk vs. Benson (1972) and Parker vs. Flook (1978), that software was similar to mathematics and laws of nature (both excluded from being patented) and, therefore, was unpatentable.

In Diamond vs. Diehr (1981), however, the court reversed course, deciding that an invention was not necessarily unpatentable simply because it utilized software. Since this decision, U.S. courts as well as the US Patent Office gradually broadened the scope of protection available for software-related inventions (Kirsch, 2000). The situation evolved to the current status in which it is expected to obtain a patent for software-related inventions. Since the State Street Bank and Trust Co. vs. Signature Financial Group Inc. (1996) case even mathematical algorithms and business methods have been found to be patentable (see also the Amazon One-click case IPXL Holding, plc vs. Amazon.com, Inc., 2005; Bakels , 2003). As from this decision, the U.S. focus, for patentability, is "utility based," which is defined as "the essential characteristics of the subject matter" and the key to patentability is the production of a "useful, concrete and tangible result" (Hart, Holmes, & Reid, 1999). The evolution resulted in

a rush of patent applications for software-related inventions and business methodologies.

Contrary to the U.S., Europe has been unwilling to grant patents for ideas, business processes and software programs. The most important reasons are their (in-) direct exclusion from patent protection, as stated in Article 52 (2)(c) European Patent Convention (EPC)[4]. Nevertheless, the European Patent Office (EPO) also reversed course. Its view on patentability of software programs and, more particularly, the interpretation of the "as such" limitation as described below, has been under revision, especially driven by the context of computer programs (the so-called computer-implemented inventions).

Following three landmark cases, Vicom/Computer Related Invention (1987), Koch & Sterzel/X-ray apparatus (1988), and SOHEI/General purpose management system (1995), the European Patent Office concluded:

a claim directed to a technical process is carried out under the control of a program (whether implemented in hardware or software) cannot be regarded as relating to a computer program as such *within the meaning of Article 52 EPC,* (emphasis added)

and

an invention must be assessed as a whole. If it makes use of both technical and non-technical means, the use of non-technical means does not detract from the technical character of the overall teaching.

Notwithstanding this enlargement in European patent law, patens have, contrary to the U.S., never been granted for software programs "as such," the main reason being that in Europe an invention has to be technical in nature. This requirement of technicality is not explicitly stated in the EPC, but can be deduced from Article 52 (2) EPC. Indeed, this provision contains a list of subject matters that are not patentable "as such" (among them programs for computers). The list is not meant to be exclusive, as it only gives examples of materials that are non-technical and abstract in nature and, thus, cannot be patented (Sarvas & Soininen, 2002).

In the U.S. on the other hand, a patentable invention must simply be within the technological arts. No specific technical contribution is required. The mere fact that an invention uses a computer or software makes it become part of the technological arts if it also provides a "useful, concrete and tangible result" (Hart et al., 1999; Meijboom, 2002).

In Europe, a number of software developers desire patent protection to be enlarged in such a way that software programs become eligible. One of the arguments of supporters of the patentability of software is that patent law provides inventors with an exclusive right to a new technology in return for publication of the technology. Thus, patent law rewards innovators for the investment and encourages continued investment of time and money. Opponents of patent protection argue that such protection is not needed, indeed appropriate in an industry such as software development, in which innovations occur rapidly, can be made without a substantial capital investment and tend to be creative combinations of previously-known techniques (Pilsch, 2005).

The opponents of software patents also indicate practical problems in administering the patent system, as software is voluminous and incremental. Indeed, an invention can only enjoy patent protection provided that it is not part of the prior art. To verify whether this condition is met or not, it is required to know the prior art. However, knowledge about software is widespread and unbundled (very often either tacit or embedded) and may thus be insufficiently explicit for the patent system to work well. In other words, there is too much software, not enough information about it, and what there is, is hard to find (Kahin, 2003). As transaction costs are high, a

patent system will favor those with enough resources to verify whether their software can be patented and, afterwards, to search for and deal with possible infringers.

Next to these financial impediments, there are some theoretical issues that concern the installing of a system of software patents. These have to do, first, with the basic, global instrument for intellectual property protection, in other words, TRIPS, and second with the specific legislation in Europe and the U.S.

Although according to Article 10 TRIPS, computer programs are protected by copyright, it is the intention of TRIPS not to exclude from patentability any inventions, whether products or processes, in all fields of technology, provided that they are new, involve an inventive step, and are capable of industrial application (Article 27 TRIPS) (Janssens, 1998). Consequently, TRIPS states, implicitly, that computer programs may also be the subjects of patent protection.

From what is stated previously, it is clear that the U.S. legislation allows patentability of software. In Europe, however, Article 52 (2) EPC remains an obstacle for such a protection, however regretted by even EPO. Indeed, in its decision of February 4, 1999, the Board of Appeals of EPO (hereafter the "Board") stated[5]:

The fact that Article 10 is the only provision in TRIPS which expressly mentions programs for computers and that copyright is the means of protection provided for by said provisions, does not give rise to any conflict between Articles 10 and 27 TRIPS. Copyright and protection by patents constitute two different means of legal protection, which may, however, also cover the same subject matter (e.g., programs for computers), since each of them serves its own purpose. (…) The Board has taken due notice of the developments in the U.S. (and Japanese) patent offices, but wishes to emphasize, that the situation under these two legal systems differs greatly form that under the EPC in that it is only the EPC which contains an exclusion as the one in Article 52 (2) and (3). Nevertheless, these developments represent a useful indication of modern trends. In the Board's opinion they may contribute to the further highly desirable (world-wide) harmonization of patent law.

This decision makes it clear that if software "as such" must be protected on the basis of patents, the exclusion under Article 52 (2)c EPC shall have to be deleted. Which brings one to the question of whether one should want this to happen and whether one perceives its consequences favorably.

Supporters of software patents would like to win a first battle in the race for software patentability by endorsing the proposal for a directive on the protection by patents of computer-implemented inventions currently being discussed within the European Union. They are aware that approving this directive will not immediately result in patentability of software "as such," however, it will form "a new development that may contribute to the further highly desirable (world-wide) harmonization of patent law," which can end up in a deletion of the exclusion now stated in Article 52 EPC. Obviously, opponents will do anything to avoid this evolution, however oblique, from taking place.

Currently the latter have the wining hand: a proposal to allow patents for computer-implemented inventions was rejected on July 6, 2005 by the European Parliament, and any new proposal will take time to develop, if ever again (Perens, 2005)[6].

MAIN FOCUS OF THE CHAPTER

Economic Justifications for Software Protection as Part of Intellectual Property (IP) Protection?

If the European Union may want to strike a balance, it must on the one hand take into account

societal needs, and on the other hand the reward of the inventor.

The theoretical foundations of intellectual property rights are debatable, to say the least. Classical philosophy has attempted to explain why intellectual property rights exist, but neither Hegel—the Germanic, idealistic school—nor Locke—the English, empirical school—has been able to provide a coherent, suitable philosophical basis for intellectual property rights—not for lack of trying by themselves or their more recent adherents (Radin, 1982; Schnably, 1993; Hettinger, 1989; Gordon, 1993). The explanation may ultimately occur, not out of law, which is in any event but a mechanic's framework, and not out philosophical theory, but out of the theory of economic pragmatism. Indeed, it may be argued that intellectual property rights, among them software protection, are what they are because they are based on, and fundamentally about, incentives to create and invest.

The United States Supreme Court summed it up in Marer vs. Stein (1954):

"The copyright law, like the patent statutes, makes reward to the owner a secondary consideration." United States v. Paramount Pictures, 334 U.S. 131, 158 . However, it is "intended definitely to grant valuable, enforceable rights to authors, publishers, etc., without burden-some requirements; 'to afford greater encouragement to the production of literary [or artistic] works of lasting benefit to the world.'" Washingtonian Co. v. Pearson, 306 U.S. 30, 36 .

The economic philosophy behind the clause empowering Congress to grant patents and copyrights is the conviction that encouragement of individual effort by personal gain is the best way to advance public welfare through the talents of authors and inventors in "Science and useful Arts." ***Sacrificial days devoted to such creative activities deserve rewards commensurate with the services rendered.*** (emphasis added)

Hence, it appears that the utilitarian, economic incentive perspective is the key driver in the granting of intellectual property laws. But are these incentives really necessary to ensure and sustain creation and invention? In the 1970s, professor (and later justice) Stephen Breyer argued that lead time advantages and the threat of retaliation reduced the cost advantages of copiers, hence obviating if not eliminating the need for copyright protection of books (Breyer, 1970, 1972; Tyerman, 1971). Advances in technology may not have strengthened Breyer's argument. George Priest argued that economic analysis (in his case of patent law) is "one of the least productive lines of inquiry in all of economic thought" because of the lack of adequate empirical bases for the assessment of theoretical models of innovation (Priest, 1986). Still, this view does not undo the fact that the pragmatic utilitarian/economic incentive perspective may remain if not the only, then at least the most useful underpinning for IP rights. These rights inescapably clash with a libertarian view that "information wants to be free" (Barlow, 1994) while those arguing against such freedom cry insist that creation and incentive will be hampered by the diminishing of intellectual property rights. Extremism has polarized views on both sides of the argument, while in the end balanced IP laws may be what are being sought (Lessig, 2004).

Enter Open Source Software

The open source software approach differs radically from the IP protection approach stated above in that it, in the words of Richard Stallman, flips it over to serve the opposite of its usual purpose: instead of a means of "privatizing" software—as Stallman puts it—through IP (copyright or patent) protection, it becomes a means of keeping software free (Stallman, 1999).

In effect, it is meant to create, at least initially, what in patent terms would be deemed a freedom to operate. In other words, open source creates an at least initial space that is open for users. Whether

this is a public domain space or another, similar form may be debated. If it is a public domain space, such an approach does not necessarily keeps open all that it touches (Friedman & Kreft, 2000). Open source software as such does not, as it does with a General Public License (GPL), have this "viral" effect.

If open source may mean the creation of a space to all users, the effect of the different licenses granted to users limits the grant of freedom to operate. There may be free software under a GPL. There may also be Open Source Initiative (OSI) licenses, which require nine elements to qualify for approval as an OSI certified license:

1. Free redistribution (no royalties or fees)
2. Access for any party to a source code
3. The license must allow modifications and derived works.
4. The license may restrict a source code, however, from being distributed in modified form but only if certain conditions are fulfilled.
5. There may be no discrimination against groups or persons.
6. Or against fields of endeavour
7. It is prohibited to require any additional licenses from users to whom a program is redistributed.
8. The license must not be specific to a product.
9. The license must not restrict other software.

Next to OSI licenses, there are others that may be copying features of such licenses, but differ, for example, as to treatment of derivative work[7]. In effect, there appear to be over 50 different open source licenses, and no clear guide (Gormulkiewicz, 1999, 2002, 2004). This is a first challenge—one that has not yet been overcome by any centralized system or standardisation.

This first challenge leads to a second one, as the licensing of open source software poses a number of legal challenges that are not necessary resolved at present. First, there is the issue of validity—a classical one that is also known in the proprietary would and goes back to the use of shrink wrap and click wrap agreements to use—assuming acceptance of the user when she/he opens the software. If this constitutes sufficient an agreement remains a question (Trompenaars, 2000).

Coupled with validity is the issue of enforceability, which is more pregnant in a open source model, because the end user may not have, or even be aware, of any license agreements unless he downloads the source code and starts using the software.

Open source software moral rights—rights related to the inventor's personality—include in patent law the right to attribution (also known as a paternity right) and in copyright include in addition at least the right to resist deformation and defamation (de Vuyst & Steuts, 2005; Metger & Jaeger, 2001). Moral rights being inalienable (i.e., non transferable) they may never be put to a user. If a user were to apply a software package for a use that the author/inventor did not like, the latter could, particularly on the European continent, where the notion of moral rights remains strongest, enforce an injunction for such use (e.g., in violent games or pornographic displays).

A third issue that is particular to an open source software approach, from a legal viewpoint, is that of representations and warranties. In a proprietary, particularly a patented would, it is inherent that the invention has been reduced to practice—that is works—before patent publication. No such warranty can necessarily be given in an open source model. In effect, the GPL does not state explicitly that the GPL code can be run—paradoxically, it does state that one may modify and redistribute. As liability is inherent in a proprietary atmosphere, it is not so inherent in an open source model (Kennedy, 2001) where limitations of liability and disclaimers of warranties may be more rightfully expected—but many limit attractiveness to users.

However, open source licensing on the basis of the principle of "no liability" is paramount to the success of open source software development. The fact that a developer of open source code has the ability to distribute work accompanied by little or no warranties effectively shifts the risk from the licensor of the code to the recipient of the code.

This is important:

Valid reasons underlie this risk-shifting strategy … . Individual hackers are unwilling to assume the risk of a multi-million dollar class action law suit as the consequences of pursuing their passion for hacking code. "Low Risk" also means low barriers to entry; anyone can contribute code to the process, not just those that can afford insurance or lawyers … . (Gormulkiewicz, 2002)

If open source software developers were not able to disclaim liability on their code, it might substantially increase development costs on account of legal risks and greatly discourage open source development.

It is however questionable whether disclaimers and limitations of liability work in all jurisdictions. Choice of jurisdiction in business open source software licenses is therefore essential, but remains problematic in a consumer atmosphere and certainly in cross-border licensing.

Last but not least, the issue of derivative work is, in all cases but in the GPL, an issue. The GPL's "viral effect," in which any modification to the source code must also be under the GPL is unique—and hard to enforce: who can find out but a disgruntled customer faced with a violation? The explanations given by the Free Software Foundation[8] refer to the judiciary for an answer to the question as to what constitutes a contribution of two parts into another program. But the courts may not be the best, certainly not the most efficient means of interpreting open source model licenses (Costello, 2002). In effect, a single German case giving effect to the GPL may not yet be an end to the relative quiet for GPL infringers (Shankland 2004). The lack of clarity in GPL licensing (not to speak of other forms, e.g., BSD or LGPL licensing) makes for uncertainty in derivative work's proprietary or non-proprietary status.

In a proprietary model, the answer, under patent law, is forthright: any derivative work is an infringement of the patent owner's rights. In the open source model, the question is the reverse: can derivative work, including interoperable work as decompiled from the source code, become proprietary in nature? In other words, can reverse engineering lead to proprietary software as it is based only on the idea encompassed in open source software?

If one reverses this risk, one should acknowledge that open source software may have the ability to expose software developers to risk if they use open source licenses improperly. This may be a disincentive to use open source software as a base platform for future development.

FUTURE TRENDS

Discussion: Open Source and the Balance of IP Rights

Free distribution and an open source code propel the open source software movement. It is a fact that software patents are expensive to prosecute and take time to publish, hampering development effort—even being bypassed by events in a quickly developing world.

The benefits of open source software also provide it its soft legal underbelly. The plethora of open source licenses, the lack of clarity and a dependence on court interpretation are a disincentive to users, but more importantly, to developers.

The solution may be in a more formal description—a restatement or standardisation, if one wants—of open source software terms of use

and licenses. But this appears not to occur yet or in the near future, although efforts to state best practices are in the make (Kennedy, 2005).

More fundamentally to the discussion of open source versus proprietary rights, a restatement of the debate as one on excessive rent seeking and a consequent imbalance of rights may point to a way to set the stage for a meaningful discussion which may lead to long-term policy framing, likely at the level of a WTO's TRIPS—a new one, with a more globally accepted balance of interest. If one addresses, in this mindset, software protection and IP rights in general, in terms of its value as an economic good to society, one is bound to find a balanced view. As Judge Posner put it:

granting property rights in intellectual property increases the incentive to create such property, but the downside is that those rights can interfere with the creation of subsequent intellectual property (because of the tracing problem and because the principal input into most intellectual property rights is previously created intellectual property). Property rights can limit the distribution of intellectual property and can draw excessive resources into the creation of intellectual property, and away from other socially valuable activities, by the phenomenon of rent seeking.

Striking the right balance, which is to say determining the optimal scope of intellectual property rights, requires a comparison of these benefits and costs—and really, it seems to me, nothing more:

The problems are not conceptual; the concepts are straightforward. The problems are entirely empirical. They are problems of measurement. In addition, we do not know how much intellectual property is in fact socially useful, and therefore we do not know how extensive a set of intellectual property rights we should create. For all we know, too many resources are being sucked into the creation of new biotechnology, computer software, films, pharmaceuticals, and business methods because the rights of these different forms of intellectual property have been too broadly defined. (Posner, 2002)

The socio-economic measurements, in empirical studies to be undertaken, set a daunting task in terms of methodology as well as in terms of execution. But they point to the way forward: a need to measure the impact of IP rights to determine the optimally needed scope vis-à-vis society.

For software protection, a know-nothing attitude that denies all rights to inventors will be a disincentive to valorise. As became clear to the first author during his tenure at the Common Fund for Commodities, what is put in the public domain is most often left there, as it is difficult to invest the time and energy to shape a competitive advantage from an invention known to and ready to use by all.

That appears to be the case at present: open source would not exist if it was not out of dissatisfaction with an excessively proprietary business model that ignores societal needs. The balance is indeed one between rights and societal needs. Unbalancing in one way or another risks, the wrong investment, or disinvestments, in software development or any other form of IP.

It is open source software that pushes towards this balance by its very existence on alternative to a proprietary business model.

Indeed, it may in three ways contribute to this balancing act already: first, though the so-called Open Patent Review it may involve the citizen in making sure patents represent progress over prior art. This is being accomplished by an alerting system that activates from the USPTO Web site (front of the AppFT database). It will enhance a public view, and review, of applications and assists in preserving safeguards against flooding strategies by would-be patent holders.

Second, open source software Prior Art, an initiative by IBM, Novell, Red Hat and SourceForce aims at developing a system that stores

source code in an electronically searchable format, exposing open source software—millions of lines of publicly available computer source code—as prior art to examiners and the public, so as to assist in ensuring that patents are issued only for actual software inventions and not for appropriations—expropriations, if one wants—of existing open source software (Noveck, 2004).

Finally, a patent quality index, in other words, a numeric index in respect of the quality of patents and patent applications, as a resource for the patent system[9], may be of interest if it proves an objective and data-driven tool. Patent rights, if put to the test, might show their true worth—and there may be a readiness to re-evaluate European and U.S. patent law—if there is a tool which assists citizens and examiners in finding the necessary balance between property rights and innovative freedom to operate, through the application of economic tools, such as a rent-seeking methodology.

CONCLUSION

While it may not be a panacea, open source software is clearly a sufficient counterweight to excessive IP creation, which in the words of professor Jeremy Phillips, does to the public domain—and to innovation—what men do to the Amazon forest (Phillips, 1996).

REFERENCES

Bakels, R. B. (2003). Van software tot erger: Op zoek naar de grenzen van het octrooirecht. *IER, August*(4), 214.

Barlow, J. P. (1994). The economy of ideas. *2.03 Wired*, 84.

Breyer, S. (1970). The uneasy case for copyright: A study in copyright of books, photocopies and computer programs. *Harvard Law Review*, 281.

Breyer, S. (1972). Copyright: A rejoinder. *UCLA Law Review*, 75.

Costello, S. (2002). *Settlement nears in open source GPL suit.* Retrieved January, 20 2006, from http://www.networkworld.com/news/2002/0305settleGPL.html

Diamond vs. Diehr, 450 US 175. (1981).

De Vuyst, B., & Steuts, L. (2005). De notie morele rechten in een internationale, vergelijkende en transactionele context. *Intellectuele Rechten—Droits Intellectuels*, 8.

EPO. (1973). *Convention on the grant of European patents (European Patent Convention) of 5 October 1973.* Retrieved January, 20 2006, from http://www.european-patent-office.org/legal/EPC/e/ma1.html

Fink, M. (2003). *The business of economics of limited open source*. Upper Saddle River, NJ: Prentice Hall.

Friedman, D., & Kreft, B. M. (2000). *Open source software: Background, licensing and practical implications.* Retrieved May 10, 2000, from http://www.daviddfriedman.com/Academic/Course_Pages/21st_century_issues/legal_issues_21_2000_pprs_web/Kreft_Open_Source.html.

Gomulkiewicz, R. W. (1999). How copyleft uses licence rights to succeed in the open source software revolution and the implications for Article 2B. *Houston Law Review*, 179.

Gormulkiewicz, R. W. (2002). De-bugging open source software licensing. *University of Pittsburg Law Review*, 75.

Gormulkiewicz, R. W. (2004). *Entrepreneur open source software hackers: MYSQL and its*

dual licensing. Retrieved from http://www.law.washington.ed/faculty/gomulikiewicz/publicaters/entopensources1.pdf

Gordon, W. J. (1993). A property right in self-expression: Equality and individualism in the natural law of intellectual property. *Yale Law Journal*, 1533.

Gottschalk vs. Benson, 409 US 63 (1972).

Hart, R., Holmes, P., & Reid, J. (1999). *The economic impact of patentability of computer programs (Report to the European Commission).* Retrieved February 18, 2004, from http://europa.eu.int/comm/internal_market/en/indprop/comp/study.pdf (pp. 20-23 of full report)

Hettinger, E. C. (1989). Justifying intellectual property. *Phil. & Publ. Aff.*

IPXL Holding, plc vs. Amazon.com, Inc., US Fed. App. Ct. 05-1009, -1487, November 21, 2005.

Janssens, M. C. (1998). Bescherming van computerprogramma's: (lang) niet alleen maar auteursrecht. *T.B.H.*, 421-422.

Kahin, B. (2003). Information process patents in the U.S. and Europe: Policy avoidance and policy divergence. *First Monday, 8*(3). Retrieved March 6, 2004, from http://www.firstmonday.org/issues/issue8_3/kahin/

Kennedy, D. M. (2001). *A primer on open source licensing legal issues: Copyright, CopyLeft, copyfuture*. Retrieved from http://www.denniskennedy.com/opensourcedmk.pdf

Kennedy, D. (2005). *Best legal practices for open source software.* Retrieved November 20, 2005, from http://www.llrx.com/features/opensource.htm

Kirsch, G. J. (2000a). *Software protection: Patents versus copyrights.* Retrieved January 20, 2006, from http://www.gigalaw.com/articles/2000/kirsch-2000-03.html

Kirsch, G. J. (2000b). *The software and e-commerce patent revolution*. Retrieved February 18, 2004, from http://www.gigalaw.com/articles/2000/kirsch-2000-01.html on February 18, 2004.

Koch & Sterzel/X-ray apparatus T 0026/86, OJ EPO 1988, 19.

Leijnse, B. (2003, January 16). Een patente oplossing voor uw patentprobleem. Softwarepatenten.be. *Trends.* Retrieved February 27, 2004, from http://www.softwarepatenten.be/pers/trends_20030116.html

Lessig, L. (2004). Be wary of IP extremists. *Computerworld*. Retrieved March 26, 2004, from http://www.computerworld.com.au/index.php?id=43841790&fp=16&fpid=0

Marer vs. Stein 347 U.S. 201 (1954).

Meijboom, A. P. (2002). Bang voor software-octrooien. *Computerrecht, 2002*(2), 66.

Metger, T., & Jaeger, A. (2001). Open source software and German copyright law. *Int'l Journal of Industrial Property and Copyright Law, 32*(1).

Noveck, B. (2004). Unchat: Democratic solutions for a wired world. In P.M. Shane (Ed.), *The prospects of political renewal through the Internet.* Oxford, UK: Routledge.

Overdijk, T. F. W. (1999). Europees Octrooibureau verruimt mogelijkheden voor octrooiering van computersoftware. *Computerrecht, 1999*(3), 158-159.

Parker vs. Flook, 437 US 584 (1978).

Perens, B. (2005, January 31). The open-source patent comundrum. *News.Com.*

Phillips, J. (1996). The diminishing domain. *European Intellectual Property Review*, 429.

Pilch, H. (2005). *Quotations on software patents. Logical Patent Web site.* Retrieved February 4,

2006, from http://swpat.ffii.org/vreji/quotes/index.en.html

Posner, R. A. (2002). The law & economics of intellectual property. *Daedalus, 5*, 12.

Priest, G. (1986). What economists can tell lawyers about intellectual property. *Res. Law & Econ*., 19.

Radin, M. J. (1982). Property and personhood. *Stanford Law Review*, 957.

Sarvas, R., & Soininen, A. (2002, October 15-16). *Differences in European and U.S. patent regulation affecting wireless standardization.* Paper presented at the International Technology and Strategy Forum Workshop on Wireless Strategy in the Enterprise: An International Research Perspective, Berkeley, CA. Retrieved on March 9, 2004, from http://www.hiit.fi/de/core/PatentsWirelessStandardization.pdf

Schnably, S. J. (1993). Property and pragmatism: A critique of Radin's theory of property and personhood. *Stanford Law Review*, 347.

Shankland, S. (2004, April 22). GPL gains clout in German legal case. *News.com.*

Sohei/General Purpose Management System T 0796/92, OJ EPO 1995, 525.

Stallman, R. (1999). The GNU operating system and the free software movement. In C. Dibona et al. (Eds.), *Open sources: Voices from the open source revolution*. O'Reilly. Retrieved from (http://www.oreilly.com/catalog/opensources/book/stallman.html)

State Street Bank and Trust Co. vs. Signature Financial Group Inc., 927 F. Supp. 502, 38 USPQ2d 1530 (D. Mass. 1996).

Trompenaars, W. M. B. (2000). Legal support for online contracts. In B. Hugenholtz (Ed.), *Copyright and electronic commerce. Legal aspects of electronic copyright management.* Amsterdam, The Netherlands: Kluwer.

Tyerman, L. (1971). The economic for copyright protection for published books: A reply to Professor Breyer. *UCLA L. Revs.*, 1100.

Vicom / Computer Related Invention, T O208/84, OJ EPO 1987, 1.

WIPO. (1971). *The Berne Convention for the protection of literary and artistic works.* Paris.

WTO. (1994). *The Agreement on Trade-Related Aspects of International Property, Annex 1C of the Marrakech Agreement of April 15, 1994 establishing the World Trade Organization ("WTO").* Retrieved January 20, 2006, from http://www.wto.org/english/docs_e/legal_e/04-wto.pdf

KEY TERMS

Copyright: A set of exclusive rights regulating the use of a particular expression of intellectual property.

EPC: Convention on the grant of European patents (European Patent Convention) of October 5, 1973.

EPO: European Patent Office (München) established by the EPC.

Patent: A grant made by a government that confers upon the creator of an invention the sole right to make, use, and sell that invention for a set period of time, through letters patent which protect an invention by such a grant.

TRIPS: The Agreement on Trade-Related Aspects of International Property, Annex 1C of the Marrakech Agreement of April 15, 1994 establishing the World Trade Organization ("WTO").

USPTO: United States Patent and Trademark Office.

WTO: World Trade Organization, established on April 15, 1994.

ENDNOTES

1 The Agreement on Trade-Related Aspects of International Property, Annex 1C of the Marrakech Agreement of April 15, 1994 establishing the World Trade Organization ("WTO").

2 The Berne Convention for the Protection of Literary and Artistic Works (1971).

3 By Europe, we mean the European national patent and the system pursuant to the European Patent Convention so that patents can be applied centrally for all contracting states of the European Patent Office (EPO).

4 Article 52 (2) (c) EPC states that programs for computers shall not be regarded as inventions within the meaning of Article 52 (1) EPC and are, therefore, excluded from patentability. Article 52 (3) EPC establishes, however, an important limitation to the scope of this exclusion. According to this provision, the exclusion applies only to the extent to which a European patent application or a European patent relates to programs for computers "as such."

5 Technical Chamber of the Board of Appeals of EPO, February 4, 1999, *Computerrecht* 1999/6, 306-310 with Note of D.J.B. BOSSCHER, 310-312.

6 See also Free Software Foundation, Software patents in Europe at http://www.fsfeurope.org/projects/swput/swput.en.htm.

7 See An overview of "Open Source" Software License, Report of the Software Licensing Committee of the American Bar Association's Intellectual Property Section, at http://www.abanet.org/intelprop/opensource.html.

8 Free Software Foundation, FAQ on the GNU GPL, at http://www.fsf.org/licenses/GPL-faq.html.

9 See the work of Prof. Polk at www.law.upenn.edu/blogs/polk/pqi/documents/2006_1_presentation.pdf

This work was previously published in the Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 328-339, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.14
How Can We Trust Agents in Multi-Agent Environments? Techniques and Challenges

Kostas Kolomvatsos
National and Kapodistrian University of Athens, Greece

Stathes Hadjiefthymiades
National and Kapodistrian University of Athens, Greece

ABSTRACT

The field of Multi-agent systems (MAS) has been an active area for many years due to the importance that agents have to many disciplines of research in computer science. MAS are open and dynamic systems where a number of autonomous software components, called agents, communicate and cooperate in order to achieve their goals. In such systems, trust plays an important role. There must be a way for an agent to make sure that it can trust another entity, which is a potential partner. Without trust, agents cannot cooperate effectively and without cooperation they cannot fulfill their goals. Many times, trust is based on reputation. It is an indication that we may trust someone. This important research area is investigated in this book chapter. We discuss main issues concerning reputation and trust in MAS. We present research efforts and give formalizations useful for understanding the two concepts.

INTRODUCTION

The technology of Multi-agent systems (MAS) offers a lot of advantages in computer science and more specifically in the domain of cooperative problem solving. **MAS** are systems that host a number of autonomous software programs that are called agents. **Agents** act on behalf of their owners giving them access to information resources easily and efficiently. Users state their requirements and agents are responsible to fulfill them. Hence, MAS include many entities trying to solve their problems that are beyond of their capabilities. For this reason, in many cases, agents must cooperate with others in order to find the

appropriate information and services to achieve their goals.

It is obvious that **MAS** are dynamic and distributed environments where agents may co-operate and communicate with others in order to complete their tasks. A key challenge arises from this nature of MAS. In such open systems, entities change their behavior dynamically. Thus, there is a requirement for trust between agents when they must exchange information Therefore, the basic question in such cases is: How and when can we trust an agent? Agents, in the majority of cases are selfish and their intentions and beliefs change continually.

We try to address this dilemma throughout this chapter. Specifically, we cover the fields of reputation and trust in MAS. This is an active research area, which is very important due to the fact that these two concepts are used in commercial applications. However, open issues exist in many cases, as it is difficult to characterize an agent as reliable or not.

In our work, we try to provide a detailed overview of reputation and trust models highlighting their importance to open environments. Due to the abundance of the relevant models, only the basic characteristics of models are discussed. We discuss basic concepts concerning MAS, reputation and trust. Accordingly, we present efforts, formalizations, and models related to the mentioned concepts. Finally, we discuss about trust engineering issues and we present future challenges and our conclusions.

BACKGROUND

Multi-Agent Systems (MAS)

Software agents and agency have been active research areas for many years due to their importance in various domains. The Web and the recently emerged Semantic Web are the most appropriate examples of such systems. In this section, basic characteristics of MAS are described. Our goal is to provide necessary knowledge about these systems and their requirements for security.

With the rapid evolution of the Internet, Software agents are a very important research area in Computer Science. **Software agents** are components of software or hardware which are capable of acting on behalf of a user in order to accomplish tasks (Nwana, 1996). The owner of an agent may be a human or another computational entity. Tasks are requested by the owners of agents in order to fulfill their needs. There are different kinds of agents. One can meet information agents that search for information sources, mobile agents that move from an environment to another, intelligent agents that can learn from their owners and the environment and so forth. For an extensive discussion of the different types of agents one can refer to Nwana (1996).

In the most cases, agents must deal with complicated tasks that demand cooperation with others. A **Multi-agent system (MAS)** can be defined as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver (Durfee & Lesser, 1989). In such systems agents can cooperate or compete with others to complete their tasks. We must note that such systems are open. An open system is one in which the structure of the system is capable of dynamically changing (Sycara, 1998). In open MAS, the basic components may change over time such as information sources or agents' behaviors. From this point of view, it can be assumed that in open MAS (Huynh, Jennings, & Shadbolt, 2006):

- Agents have different owners and for this reason they are selfish and may be unreliable;
- There is no knowledge about the environment in which agents must interact with each other; and

- There is no central authority that controls the agents.

The last point is important for the cooperation among agents. Cooperation is often presented as one of the key concepts which differentiates MAS from other systems (Doran, Franklin, Jennings, & Norman, 1997). Through cooperation agents are able to obtain the necessary information needed for their tasks. Of course, interactions are the key issue for the cooperation.

It is obvious that there is an increasing need for the definition of trustworthy entities. In an open environment like MAS, agents change their intentions, goals and behaviors continually thus rendering imperative the need to define methods based on which each agent can be enabled to recognize nontrustworthy entities. The most important thing in such cases is to find ways to acquire information related to others' behavior. For example, an agent must communicate with the candidate partner or with others, in order to infer its trustworthiness. We describe methods to achieve this goal, and we give their basic characteristics.

Reputation in MAS

Reputation is an important factor in many research fields. Especially in computer science, reputation mechanisms are used either in research efforts or in commercial applications. In MAS, agents have to interact with others in order to fulfil their owners' needs for information. In such cases, reputation plays an important role.

According to a dictionary "**reputation is the state for a person of being held in high esteem and honour.**" From a social point of view "**reputation is the general estimation that the public has for a person**" (Wordnet, http://wordnet.princeton.edu).

In MAS, reputation refers to a perception that an agent has of the intentions and norms of another (Mui, Halberstadt, & Mohtashemi, 2002). This is critical for the cooperation among autonomous components in open environments, where the knowledge about the plans of others is limited.

One can find a categorisation of reputation in Wang and Vassileva (2003). Authors distinguish reputation models as centralised or decentralised, according to who has the responsibility to derive a reputation value. It should be noted that authors consider that trust is elicited through reputation. Therefore, their categorisation concerns both reputation and trust models.

- **Centralised.** In centralised reputation and trust models, the system is responsible to collect ratings for agents and publish them. Through this procedure, all ratings are evident to all members of the community and there is little need for communication among agents. Also, an aggregation procedure is performed by the system. The aggregation procedure aims to combine the different opinions in a final reputation level. Centralised models are characterised by simplicity and are mainly encountered in the area of e-commerce, where the main transactions are between sellers and buyers.
- **Decentralized or Distributed.** In decentralised systems there is not a central responsible authority and for this reason each agent develops its own reputation level for other community members. This means that there is an increased need for interactions between agents. Through them, agents form a subjective trust in their potential partners.

Mui et al. (2002) discern reputation based on which experiences and ratings are taken into consideration, and through what procedure information for the opponents is extracted. According to authors, reputation models can be divided into the following:

- **Individual Reputation.** Individual reputation is the description of the reputation level of a simple entity by another. This level is computed based on actions and information related to an agent and not a group of agents.
- **Group Reputation.** Group reputation depicts the social dimension of reputation. In these models, reputation is a function of the aggregated ratings taken from a group of entities. Entities rate others having their own experiences. These ratings may be utilised to provide information to an agent, when it needs to cooperate with others.
- **Direct Reputation.** Direct reputation is based on straight experiences with an entity. Usually, these observations are taken by interactions held between two entities. Direct reputation may be *observed* or *encounter-derived*. We have observed reputation when feedbacks through direct experiences of others consists a reference of the reputation of an agent. On the other hand, entities' ratings, after an interaction with others, may affect the reputation level of an agent. In this case, we have encounter-derived reputation.
- **Indirect Reputation.** With the lack of direct experiences for an entity, reputation can be derived from information gathered indirectly. There are three basic models for indirect reputation. The first model uses prior beliefs that agents carry about their interactions with strangers while the second model takes into consideration the group that an agent belongs to. Finally, the third model uses the information taken about an agent from the entities in the environment.

Trust in MAS

Trust is a common theme in computer science research, and refers to a range of different issues. It has important impact on domains such as security, e-commerce and Semantic Web. Trust is also an important concept for MAS. While in general trust refers to an aspect of the relationship of individuals, the concept has a completely different meaning depending on the context is used (Deriaz, 2006). Hence, trust has different meaning when we use it to characterize that humans' actions are trusted or when an agent decides to rely on another in order to obtain some resources.

Trust can be seen as the extent to which one entity intends to depend on somebody in a given situation (McKnight & Chervany, 1996). Trust can be defined as the belief that one can rely on someone else to accomplish a task. There is, however, a possibility that unfavourable issues can arise from interactions with a trusted person.

In this point, we describe a list of trust categories found in the literature.

According to Ramchourn, Huynh, and Jennings (2004) trust may be categorised, based on the part which decides the grade of trust, into the following:

- **Individual Level Trust:** Each agent decides which entity can be trusted based on its beliefs. These beliefs derive from interactions held between agents. Individual level trust can be further divided into:
  a. **Learning based.** Agents may interact with each other many times before deciding to trust someone. From this procedure, useful conclusions can be derived for the potential partners. Through repeated games, agents are able to analyze their opponents' moves in order to reach a conclusion. There are different kinds of metrics used in such models, as bi-stable values (good or bad) or fuzzy mechanisms with which one can decide to trust someone else for various acts.
  b. **Reputation based.** Reputation-based models use ratings from the members of a community in order to derive a trust level. Important issues concern-

ing this type of trust are the collection of ratings, their aggregation and the diffusion of members' opinions. First of all, an agent must collect ratings from the members of the community through the use of *referrals*. Referrals are the opinions of community members about a certain entity. After that, he must use an aggregation method in order to extract a result. In this point, there are issues that can complicate the whole procedure, as the lying witnesses or the absence of ratings. Finally, an important issue is the propagation of reputation in a community based on reputation scores that agents have for a set of other entities.

c. **Socio-cognitive based.** Contrary to previous types, where trust is computed taking into consideration the results and the components of interactions between agents, socio-cognitive-based trust is computed based on beliefs that an agent has for their opponents. These beliefs are: competence, willingness, persistence, and motivation belief.

- **System Level Trust:** Agents are selfish components which want to obtain as much profit as possible. For this reason, it is imperative to force them to follow some rules when interacting in the context of a system. This is the system level trust. In consequence, they will be trustworthy, thus minimizing the danger to interact with liars. System level trust can be further divided into:
  a. **Truth-eliciting protocols.** These protocols may be used to elicit trustworthy behaviour of an agent. Agents must conform to certain protocols' steps in order to complete transactions in the system.
  b. **Reputation mechanisms development.** Reputation may be used in system level trust. There are rules posed by the system concerning the three key elements of reputation models (collection of ratings, aggregation and propagation). In such systems, the entities responsible to store ratings may be centralised or decentralised. All agents working in the system have access to these entities either to read ratings or to publish their own.
  c. **Security mechanisms development.** In these model types, a number of features are taken into consideration in order to provide a reliable security mechanism that ensures trust in system entities. The essential elements that make an agent trustworthy can be identity proof, access permissions, content integrity and content privacy (Poslad, Calisti, & Charlton, 2002). Additionally, certificates may be used to provide a higher level of security. The system forces members to give the necessary information as for the aforementioned elements in order to have an acceptable degree of safety.

Artz and Gil (2006) note that there are two methods to derive trust:

- **Based on credentials.** Credentials are elements that can be used to elicit information for an entity. A credential may be simple as a signature or complex relationships between elements in an open environment as the Semantic Web. For example, an agent may have an identifier with which may interact with others. This identifier may be used in a system to provide to an agent permissions or rights to work with specific information sources. An extensive review of systems that use credentials-based trust models is presented in Artz and Gil (2006).
- **Based on Reputation.** This model uses reputation to assign trust to members of

a community. An agent utilizes personal experiences taken from interactions held between potential partners, and ratings from other members of the system. There are two ways to extract the trust level for an entity. One can rely on a central authority to have access in reputation ratings or on himself. Few efforts in literature use the first method. The second one describes the decentralised model where each entity must develop methods for the aggregation of ratings taken from the community.

In Osman (2006), authors specify the difference between trusting an agent, and trusting an interaction. They provide a categorization and a specific connection with the categories presented in Ramchourn et al. (2004). Two new categories are presented:

- **Local Deontic Level.** It concerns the constraints and permissions that an agent must follow when interacting in a multi-agent system. Agents are dynamic components and their goals, intentions and plans change continually. This means that when they work in an open environment, they have obligations, permissions and prohibitions, posed by the system. Through this, the system defines a security level concerning the transactions held among the potential partners.
- **Global Interaction Level.** Apart from the internal deontic model of each agent, there is another interaction model that specifies the rules based on which interactions are held. Every agent must conform to these rules in order to gain access to interactions with others, useful for the completion of its goals. Specifically, the interaction model is a protocol that determines steps to carry out interactions.

Grandison and Sloman (2000) presented a set of trust classes which are:

- **Provision Trust**. It describes the trust that an entity may have to a service provider.
- **Access Trust.** It describes the trust that an entity may have for the purposes of accessing resources.
- **Delegation Trust.** It describes the trust that an entity may have to an agent that works on its behalf.
- **Identity Trust.** It describes the belief that an identity is as claimed.
- **Context Trust.** It describes that the relying entity has confidence in a system in which transactions are held. Moreover, each entity can rely on system, when problems may arise in transactions.

Finally, another categorisation found in Wang and Vassileva (2003) has already been presented, where trust quantification may be held either by a central authority or by each agent individually.

A significant amount of work on trust has been performed in the area of Normative MAS (NMAS). NMAS is an extension of classical MAS which combines traditional MAS with normative systems where concepts such as obligations, commitments, permissions and rights are used to describe the behaviour of an entity (Boella, Van Der Torre, & Verhagen, 2006). Thus, every agent acting in a community has some obligations and commitments that should be fulfilled. In such systems, norms are defined to describe when the behaviour of an agent is acceptable. While the agent follows these norms its trust level increases in the community. In reverse, the agent's trust level decreases when its actions do not conform to the specified rules of normal behaviour. These rules aim to force agents to do the right thing cooperating with others in the broader environ-

ment. However, norms can be violated for various reasons and thus there is a dynamic trust valuation (Boella & Van Der Torre, 2005).

ISSUES CONCERNING REPUTATION

Reputation level can be derived based on three elements: the experiences of the evaluator, the referrals of others and the combination of the experiences and the referrals (Josang et al., 2006). There are methods that deal with all these three issues. Generally speaking, in MAS a set of agents $A=\{a_1, a_2, \ldots, a_n\}$ want to interact with others in order to complete their goals. For each potential partner, every agent must calculate its reputation degree which is extracted through a reputation function:

$$Reputation\_value=f(R, E, S) \quad (1)$$

where R represents ratings from other members of the community, E are the individual experiences taken from direct interactions with the target entities, and S represents ratings retrieved from the system.

The factor R is computed based on aggregation of ratings. Namely, there is an aggregation function which derives a final value from a set of witness agents $WA=\{wa_1, wa_2, \ldots, wa_m\}$.

$$R=g(r_1, r_2, \ldots, r_m) \quad (2)$$

where r_i denotes the referrals of the i_{th} witness agent. In literature, there are models that use only one of the above mentioned elements or a combination of them. As discussed below, every reputation model uses a function in order to calculate the final result, which follows the general form depicted in (1). For example, an agent may be based only on direct experiences with the target agent, without paying attention to the ratings of others. In order to define efficiently final reputation value, a model must be based on a combination of the above mentioned features (e.g., referrals, the system's ratings and direct interactions). The majority of systems in the literature follow this direction. Their difference is located to the form of the referred functions and the type of the computed values. Hence, in all models we can find a reputation function that produces a value that can be either discrete (e.g., "Confident," "Non-Confident") or continuous (represented through a real number).

A short description of reputation models follows. Furthermore, we present our point of view related to their advantages and disadvantages.

Simple Mathematical Models

They are the simplest models. In these models simple calculations are used in order to compute reputation values. For example, the system may store the number of positive and negative opinions for agents and compute the final score. If the positive opinions are $P=\{p_1, p_2, \ldots, p_k\}$ and the negative are $N=\{n_1, n_2, \ldots, n_m\}$ then the final score is:

$$Score = |P| - |N| \quad (3)$$

where |x| denotes the cardinality of set x. The higher the value of *Score* is the more reliable the agent is considered. For example, if an agent has received 10 positive and 2 negative referrals, it has a reputation degree of 8. This agent is more reliable than another that has a reputation degree of 5. However, these models do not take into consideration the initial numbers from which the final result is computed. Let us examine an agent that may have received 100 positive and 90 negatives opinions. This means that the agent has approximately 47% negative opinions in the community. Nevertheless, this agent is more reliable than another with 10 positives and 1 negative referral (approximately 9% negative opinions).

In order to cover these disadvantages, advanced mechanisms use a weighted sum to compute an average which shows the reputation level. These mechanisms use the information related to the ratings such as the age of each rating, the distance between rating and current reputation value, and so forth (Josang, Roslam, & Colin, 2006).

It should be noted that such systems do not take into consideration critical issues concerning the selfish and dynamic nature of agents. It is possible that agents may form coalitions in order to exchange positive marks thus achieving better reputation scores. Furthermore, the simple mathematical models do not examine in depth the referrals retrieved from the community members in order to extract useful information about the behaviour of an agent.

Bayesian Reputation Systems

Bayesian systems are based on statistics. They compute reputation using the Beta probability density function. This function can be used to describe probability distributions of binary events. For simplicity, we give only a short description of such systems.

A Bayesian reputation system takes binary ratings and uses the *a priori* reputation score and the current ratings to compute the *a posteriori* result (Josang & Ismail, 2002; Mui, Mohtashemi, Ang, Szolovtis, & Halberstadt, 2001; Whitby, Josang, & Indulska, 2004). In agent systems, we can describe the behaviour of an agent as "Honest" vs "Dishonest," or as "Reliable" vs "Unreliable" which constitute binary events. If, for one agent, there are x positive and y negative observations, then the reputation score can be computed as follows:

$$\alpha = x+1,\ \beta = y+1 \text{ with } x,y \geq 0 \qquad (4)$$

the probability expectation value is:

$$E(p) = \frac{\alpha}{\alpha + \beta} \qquad (5)$$

for the Beta distribution, which can be expressed as:

$$B(p|\alpha,\beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha) \cdot \Gamma(\beta)} p^{\alpha-1} \cdot (1-p)^{\beta-1} \qquad (6)$$

where $p \in [0..1]$, $\alpha,\beta>0$, $p \neq 1$ when $b<1$ and $p \neq 0$ when $a<1$.

When the a priori probability does not exist, then we consider $\alpha=1$ and $\beta=1$. From this function, each agent can compute the possibility that a potential partner is reliable based on previous values of reputation. For example, if the expectation value E(p) has a score of 0.9 means that the most likely value of positive outcomes in the future is 0.9, but the actual outcomes are uncertain.

Bayesian models give a computational theoretical framework for the reputation score good for autonomous computational entities as agents are. It is an efficient mechanism to combine evidences. An entity must keep track of the outcomes of others and compute the reliability possibility through the above referred functions. A full description of a system representative of this kind of model can be found in Josang and Ismail (2002). However, these models are complicated due to the calculations that must be performed in order to derive a final reputation value. Also, the definition of a priori probability used for the calculations is necessary. The probability value is important for these models and must be derived by a subjective method.

Social Networks

Social networks are originated in sociology. Social networks can be represented as graphs that depict relations between members of a community. Social networks analysis emerged as a set of methods for the analysis of social structures (Sabater & Sierra, 2002). In MAS, agents must retrieve data

concerning the relation among the members of the system in order to decide the reputation level of a potential partner. However, it is difficult to use methods taken from sociology in order to poll information for the network architecture. For example, sociologists use methods as the opinion poll or interviews. In cases where autonomous computational components are the nodes of a social network, more "computational" ways must be found to extract the necessary information.

The procedure that agents adopt for building the network is critical for the success of such systems. They must describe as much relational data as they can in order to have a significant view of the system. We must also note, that these networks change dynamically due to the open nature of MAS. Agents may enter or leave at every time and, moreover, may alter their goals, behaviours and intentions.

Generally speaking, in systems based on social networks there is a set $A=\{a_1, a_2, a_3, \ldots, a_n\}$ of agents that want to obtain information from others in order to complete their goals. Each agent builds a social network $G=\{d_1, d_2, \ldots, d_n\}$, with nodes d_i representing the members of the system. Edges that connect nodes show a relation between them. For example, edge $e_{i,j}$ denotes that there is a relation among nodes i and j. Next, it finds its potential partners and tries to collect information about them. It may be based on referrals or on the system or on a combination of them. Referrals are taken from other members that have interaction history with the target agents, and after a careful selection. These referrals must be aggregated in order to extract a final value through which reputation is computed.

Three critical factors must be taken into account: the possibility that agents may tell lies, the possibility that agents may conceal information about others and the careful selection of referral agents. In these cases, social networks may be constructed based on false values as for the reputation degree of each member. Also, there are cases in which agents ally with others and for this reason may hide the bad reputation that an examined agent acquired. A method to alleviate the problem of lying in MAS is presented in Schillo, Funk, and Rovatsos (2000).

As mentioned above, in MAS social networks agents appear as nodes and their relationships as edges. Each edge has a value which represents the weight of the relationship between the two connected agents. After the graph creation, the construction of the network and the computation of reputation follow. Such computation is based on the weights assigned to edges. An extensive survey on reputation mechanisms based on social networks can be found in Ramchourn et al. (2004).

In this point, we present two simple examples of social networks models. A first example of such model is presented in Pujol and Sanguesa (2002). Authors describe an algorithm which is named *NodeRanking* and is used to extract a reputation value for members in a community. Its main idea is that every node and, respectively, agents have an authority degree which is an importance measure. For example, the authority of a node x is computed as a function of the total measure of authority presented in the network and the authority of the nodes pointing in x. The main rationale is that if a node has a lot of edges pointing to it, this means that the node is important in the community because this means that it cooperates with many other members. Another critical issue is that every authority value is propagated through the out-edges. The reputation value is computed based on the authority value of each node, taking into consideration the importance and hence the number of agents that are related to the examined entity.

Another system that uses social information in order to compute reputation is REGRET (Sabater & Sierra, 2002). Reputation in REGRET has three dimensions, the *individual*, the *social*—according to the source of the information used to extract a

reputation value—and the *ontological* dimension, which helps to transfer the reputation between related contexts. While the individual dimension takes into consideration the results of direct interactions between potential partners, the social dimension utilizes information taken from the other members of the community. In the second dimension of reputation, agents may use witnesses from others or consider neighbourhood reputation. Furthermore, the system assigns a reputation level to every role defined in it. Hence, agents that have a specific role in the system inherit the reputation level assigned to the role.

System reputation is the easiest to compute but is dangerous because a role held by an agent does not convey information about its intentions. In the REGRET system, reputation is combined with a domain and calculated using a table in which rows are the potential roles and columns are the reputation types. More complicated are the remaining two methods. Witnesses reputation uses the referrals of others in order to establish a reputation level. REGRET gives the opportunity to an agent to define a set of witnesses and aggregate their referrals based on fuzzy rules. Of course, witnesses are entities that have interacted in the past with the target agent and they are taken into account based on the same event, if it is possible. Neighbourhood reputation is not related to the physical location of the agents but to the links created by interactions. These interactions and the relations between agents are very useful to compute reputation level for a target agent. Fuzzy rules are also used in this case.

Belief Theory Models

Belief theory characterizes the remaining of the subtraction between 1 and the summary of the possibilities of the all possible outcomes, as uncertainty. In these models, agents use their beliefs about the behaviour of another entity. In Yu and Singh (2002), authors propose the use of Dempster-Shafer theory (Dempster, 1968; Shafer, 1976) for the computation of reputation degree. There are two kinds of belief in their model: *Local* and *Total*. Local belief is obtained from direct interactions with the target agent and Total belief is extracted from the opinions of others combined with local belief. Witnesses from others are necessary when interactions are not available. Each agent models the information from others using belief functions. There are two outcomes related to the reliability of an agent: *Trustworthy* or *Not Trustworthy*, each of them has a belief value m(T) and m($\neg$ T) respectively, taken from the correspondent belief function. The reputation score for an agent A is:

$$\Gamma(A)=\beta_A(\{T_A\})-\beta_A(\{\neg T_A\}) \qquad (7)$$

where β_A is the cumulative belief result computed using the testimonies from a set of L neighborhoods. When no testimonies are available, then the reputation score is 0. Also, authors present the reputation value of a set of K agents, which is:

$$\text{Group\_Reputation} = \frac{1}{K}\sum_{i\in[1..K]}\Gamma(A_i) \qquad (8)$$

These models are able to exhibit the beliefs of agents, accumulated from past experiences or others, in functions that combine them and produce the final result. However, the definition of a threshold value is critical. This threshold defines when an agent may be characterized as trustworthy or not. Moreover, the assumption of only two possible outcomes limits the model.

Fuzzy Models

Fuzzy models try to catch the subjective point of view of an agent related to another member of a community. In these models, reputation is presented through linguistic fuzzy values in contrast to other models in which common reputation levels are defined by means of real numbers. For

example an agent may be characterised as "reliable" or "not reliable." Fuzzy logic (Zadeh, 1989) is very important because it provides reasoning techniques for the extraction procedure. Rules may have the next form:

IF andecent THEN consequent

where *andecent* is represented with fuzzy sets.

Fuzzy logic techniques depend on subjective criteria that may lead an agent to cooperate based on its thoughts about others. This means that if an agent has very optimistic views of the community he may rely on others that have bad intentions.

In Rubiera, Molina Lopez, and Muro (2001) a method for the computation of reputation based on a fuzzy model is presented. Each agent retrieves opinions only from entities that are highly appreciated. Based on their answers it computes a value that is extracted from a fuzzy set according to its point of view. The result is the weight of an agent's opinion. Furthemore, there is interest on combination of the new and the old reputation values. The old reputation score of the candidate partner is taken into consideration and, hence, the final result is the average of two fuzzy values: the old and the new one. The agent is responsible to decide on defection. Usually, if a threshold is reached, cooperation is held.

Another system that relies on fuzzy rules is REGRET, which was briefly described in the previous section.

Role-Based Reputation

In some models, reputation can be seen as a value of a role fulfilment. A role is a set of obligations and actions that an entity has in the community. If an agent acts and behaves as a role dictates, then it has the reputation level that this role offers. In Carter and Ghorbani (2004) a framework for the role fulfilment measurement is presented. Three roles are investigated: The Assistant, the Provider and the Citizen. A general overview of how measurements take place in each role is given by the authors. In order to compute the final value of reputation, authors examine the satisfaction degree of each role for a specific entity and combine these partial results. The reputation is a weighted sum of each value that reflects the fulfilment of each role.

The main problem with these models is that they do not examine in depth the intentions that agents have. Meeting the requirements of a role by an agent does not mean that the agent will not change its behaviour.

Unfair Ratings: Deception

As mentioned above in distributed reputation models, when a central authority is absent each agent that wants to cooperate with others must collect ratings from the environment in order to decide the reputation degree of an entity. In these cases important issues are:

- The possibility that some agents provide unfair ratings for others. These ratings may be unfairly positive or unfairly negative (Dellarocas, 2000).
- The possibility that an agent deludes others.

According to Whitby et al. (2004), methods of avoiding unfair ratings are divided into *endogenous* and *exogenous.*

Endogenous methods are based on the statistical analysis of the rating values. They can give or exclude ratings that are possible to be unfair. Classical examples of this kind of systems are Bayesian reputation systems. Authors describe an algorithm that filters unfair ratings in a Bayesian model. Exogenous methods are based on factors that are related to external elements such as the reputation of the witness. The main idea is that an entity with low reputation is likely to give unfair

ratings and vice versa. In the relevant literature, one can find a lot of works falling in the aforementioned categories.

Another algorithm for the detection of deceptive agents is proposed in Yu and Singh (2003). This method uses exogenous characteristics of the witnesses as we presented above. The algorithm assigns weights to witnesses and makes a prediction based on the weighted sum of their ratings. The second idea is to tune these weights when a prediction fails. In this case, the weight of successful witnesses is increased and the weight for the unsuccessful is decreased. We must note that the ratings that an agent takes are belief functions and for this reason the algorithm maps belief functions to probabilities in order to be able to compute and update the weights of each witness. Moreover, authors study the number of witnesses and its effect to the system's prediction values.

ISSUES CONCERNING TRUST

Trust is usually researched in the security domain. The main reason is that these two concepts are related, but they have different orientations. However, trust and security provide protection against malicious components. In this sense, trust can be considered as a *soft security* mechanism. This term first appeared in Rasmusson and Jansson (1996). Authors discern *hard* and *soft* security mechanisms. Hard tools are authentication, cryptography, and so forth, and soft tools are those that take into consideration social control issues, as they are trust and reputation.

In MAS, trust plays an important role because agents need to cooperate with other members of the community. The importance of trust in MAS is shown in Castelfranchi and Falcone (1998). Critical questions arise such as: When can I trust another entity? Which entity is trustworthy? What are the elements that can be used in order to conclude a trust level? What methods should be used to conclude trust level? Such questions are addressed in the fourth section of our chapter.

Discussion

The main differences between trust and reputation are:

a. Usually, trust is a score that reflects the subjective view of an entity from another, whereas reputation is a score that reflects the view of the community.
b. In trust systems, transitivity is considered explicitly while in reputation systems is seen implicitly (Wang, Hori, & Sakurai, 2006).

The common element between the two concepts is that both of them try to help someone that wants to find trustworthy partners to achieve its goals through cooperation. However, trust is more complicated concept that involves many parameters. For this reason, it is very difficult to assign a strict definition to trust.

As mentioned above, trust is a subjective view of an entity. It is based on some beliefs that an entity has for another, but it is not clear where such beliefs originated. This means that an agent may be reliable only for a set of other agents and not for all of them. The level of trust is also depended on the context in which it is being studied. For example, an agent may be trustworthy when providing information but it is nontrustworthy when selling products. These two factors are basic to open systems and must be taken into consideration. Moreover, trust is dynamic. An agent may consider another entity as reliable in a specific time but its opinion may change accordingly based on the behaviour of the target entity.

The simplest form of trust is centralized. In such systems, there is a central authority that keeps the trust level of entities who rate each other after every transaction. Soft mathematical

calculations are held to provide the final result. It is a scheme that must take into account issues concerning lying entities or unfair ratings. Also, there must be a high security level to prevent violations in central database where ratings are kept. On the other hand, decentralized trust models are complex and require effort from the side of each entity that tries to find partners. In such systems, critical issues are the storage of trust values, the location of witnesses and the inference procedure.

In general, a trust function has the following parameters: the beliefs of the examiner (B), the reputation of the examinee (R), previous trust values (P) and the context (C).

$$\text{Trust\_value}=f(B, R, P, C) \quad (9)$$

An interesting value is B. B may be extracted from direct experiences through communication or past experiences with the target entity. It may be a positive or a negative belief. Relation (9) concerns a general function form. As we discuss in the following paragraphs all the described models use a function that follows this general form and takes into consideration one or more values from B, R, P or C.

The result of the referred function may be discrete or continuous values. For example, in discrete models, as fuzzy models are, a trustworthy behaviour may be characterized as "Very Trustworthy," "Trustworthy," "Untrustworthy," or "Very untrustworthy," for direct trust between two entities, or "Very good," "Good," "Bad," or "Very bad" for recommender trust (Abdul-Rahman & Hailes, 2000). Either discrete or continuous, the final trust value reflects a confidence over the knowledge we have about an entity.

Trust mechanisms vary from these that use simple computations to those that use more complicated characteristics of the entities involved in such situations. However, the common procedure among them is that they map a set of features to trust information. In the following paragraphs, we give a description of some important categories of trust and examples of each one.

A key issue concerning trust is its dynamic nature. Trust evolves over time as entities cooperate with others. For this reason, it is critical to define a trust update procedure. Especially in open environments like MAS, where goals, intentions and beliefs of each agent change continually, there is a need for dynamic adaptation of the trust level. This means that in every model that is used to compute trust, developers must take into consideration how trust levels evolve over time and transactions. The evolution of trust may be based on the experiences of the trustor or on new information taken from other members of the community.

In conclusion, in the computing trust procedure, the phases that an agent may handle are:

a. Trust Discovery Phase (TDP);
b. Trust Aggregation Phase (TAP); and
c. Trust Evolution Phase (TEP).

In TDP, each agent tries to find the appropriate sources for referrals and may communicate with the target agents in order to elicit useful information about their behaviour. In this phase it is important to have mechanisms to identify if a group of agents have formed a coalition and share good referrals among them. In TAP, the most important issue is to use an appropriate aggregation function in order to derive the final value of trust. This function may take into consideration the results of direct experiences and of course the referrals of peers. Finally, the TEP is a continuous procedure through which an initial trust level is evolving over time based on observations. Its great importance relies on the dynamic nature of MAS. If an agent is trusted in a specific time this does not mean it is to be trusted forever. Agents are selfish and may change their behaviour without warning or may ally with others.

Trust Propagation

MAS can be viewed as graphs where their agents are represented by nodes. Edges consist of their relationship in the community and weights represent the trust value between the two connected nodes. Graphs may be used to transfer trust information among members. Every agent trusts some others in the network. *Estimated trust belief* is derived through the trust network based on inferences while *expected trust belief* is the ideal target (Ding, Kolari, Ganjugante, Finin, & Joshi, 2004). It is very difficult to achieve the expected trust belief due to the lack of global knowledge of the community and its members. As the trust network evolves over time and more information is gathered from the agents, the ultimate goal is gradually approached.

Propagation of trust is very important in such networks because it gives the opportunity to derive beliefs about agents through the combination of values taken from multiple sources. The most common method for trust propagation is referrals. Referrals have been investigated in reputation mechanisms (see Section "Issues Concerning Reputation"). An agent, having collected opinions from a set of peers, needs an aggregation method in order to define the final estimated value of trust. Through this procedure, trust information can be propagated over the network. Additionally, we must take into consideration that agents may ally with others and give positive recommendations for their allies. However, when an agent establishes trust based on recommendations from others, this trust value cannot be greater than the trust value between the agent and the recommender, and neither the trust value between the recommender and the target agent (Lindsay, Yu, Han, & Ray Liu, 2006). Another effort on trust propagation is discussed in Guha, Kumar, Raghavan, and Tomkins (2004).

Simple Trust Models

Simple trust models try to determine relations that depict the behaviour of an entity as a function of positive and negative opinions. The simplest form, found in Deriaz (2006), is:

$$\text{Trust\_score} = \frac{|\,Positive\ Ratings\,|}{|\,Positive\ Ratings\,| + |\,Negative\ Ratings\,|} \tag{10}$$

where |Positive Ratings| and |Negative Ratings| represent the number of positive and negative ratings, respectively.

Whenever an certain agent has only positive ratings and no negatives, then the trust score is equal to 1. Therefore, *Trust_score* can take values ranging from 0 to 1. An extension to this model that take into consideration the time in which these ratings are provided gives more efficient trust computation because it can exclude obsolete ratings. Furthermore, trust can be computed if we scale recent events. Accordingly, if in Deriaz's model we set a=b=c=1, which are the default values of parameters a, b, c, then the following holds:

$$Trust\_score = \frac{|\,PosRatings\,|\cdot(1+\delta_p) + A}{|\,PosRatings\,|\cdot(1+\delta_p) + |\,NegRatings\,|\cdot(1+\delta_n) + B} \tag{11}$$

where

$$A = \frac{1}{T}\sum_{i\in[1..|PosRatings|]} t(po_i) \tag{12}$$

$$B = \frac{1}{T}\left(\sum_{i\in[1..|PosRatings|]} t(po_i) + \sum_{i\in[1..|NegRatings|]} t(pn_i)\right) \tag{13}$$

and δ_p, δ_n are the statistical variance of the positive and negative outcomes, T is the current time, po_i, pn_i are positive and negative rating received at time i, |PosRatings| and |NegRatings| are the cardinality of positive and negative ratings, respectively.

The disadvantage of this mechanism is that it does not take into account the context in which these ratings were made. The above forms deal with the positive and the negative opinions without separating them into the context fields for which ratings are formulated. Furthermore, the model does not notice cases where entities may ally with others in order to elicit positive outcomes or the case that an entity is neutral to another.

Entropy-Based & Probability-Based Trust Model

In this section, we present two models of trust based on the work reported in Lindsay et al. (2006). Authors, influenced by information theory, present *Entropy-based* and *probability-based* trust. The trust relationship between two entities is represented by T(S,A,AC), where S is the subject which examines the trust level of the agent A for an action AC. Similarly, the probability that and agent A will perform the action AC in the subject's S point of view is represented by P(S,A,AC). The entropy based value of trust is calculated as follows:

$$T(S,A,AC) = \begin{cases} 1 + p \cdot \log_2(p) + (1-p) \cdot \log_2(1-p) & 0.5 \le p \le 1 \\ -p \cdot \log_2(p) - (1-p) \cdot \log_2(1-p) + 1 & 0 \le p < 0.5 \end{cases} \quad (14)$$

where

$$p = P(S,A,AC) \quad (15)$$

The final trust value is a real number in the interval [-1,1]. Some important examples that show the trust level of an agent are:

$$C = \begin{cases} \textit{Subject trusts the agent} & p = 1 \textit{ and } T = 1 \\ \textit{Subject distrusts the agent the most} & p = 0 \textit{ and } T = -1 \\ \textit{Subject has no trust} & p = 0.5 \textit{ and } T = 0 \end{cases} \quad (16)$$

In general, the following holds:

$$T(S,A,AC) = \begin{cases} < 0 & \textit{when } p \in [0..0,5) \\ > 0 & \textit{when } p \in (0,5..1] \\ = 0 & \textit{when } p = 0,5 \end{cases} \quad (17)$$

We should note that the probability P(S,A,AC) represents the view of a specific subject which means that different agents have different opinions about a target agent.

The entropy-based model depends on the trust value described above. Especially for the propagation of trust, a simple product is used where the two factors are the recommendation value of another agent multiplied by the trust value of the recommender. For multipath recommendations, the final result is the weighted sum of each recommendation. In the probability-based model, the probability that an agent will perform the specific tasks combined with the probability that a recommender make correct recommendations is adjusted.

Reputation-Based Trust Models

Reputation based trust models are used in distributed systems where there is little information on the overall network. If an entity has a high reputation level in a community then others may trust it more easily than another that has lower reputation value. For the computation of trust, an agent depends on opinions of a set of community members. Important issues in these cases are the collection method of ratings and the aggregation procedure. It should be reminded that trust is a concept derived from direct interactions between two entities, while reputation is the view of a member from the community side.

Reputation-based models rely on methods that give the opportunity to an entity to gather referrals from other members and apply an aggregation function in order to calculate the final value of trust. It is wiser to combine these results with values obtained from direct experiences. In Ramchourn et al. (2004), authors give an extensive review describing methods for the retrieval of ratings and the aggregation procedure from a social network point of view.

Reputation models are discussed in previous section.

Bayesian Network Trust Models

A Bayesian network is a network where probability relationships of some entities are presented (Ben-Gal, 2007). The final trust value is represented by the root of the network and leafs are the sources in which the beliefs of the examiner are based. A formalization of Bayesian trust networks is given in Melaye and Demazeau (2005). Each agent forms its basic beliefs investigating a number of belief sources. We have:

$$\text{Basic\_Belief}_i = f(B_{i1}, B_{i2}, \ldots, B_{iN}) \quad (18)$$

where $i \in [1..\text{number\_of\_basic\_beliefs}]$ and N are the number of belief sources. An important point is that the function f is depended on conditional probabilities, which means that a tree node may be more influential than another. Each trust component is associated with a probability of satisfaction. Accordingly, the final trust value is calculated as follows:

$$\text{Trust\_value} = g(BB_1, BB_2, \ldots, BB_N) \quad (19)$$

where BB_i is the i-th basic belief taken from (18). It is obvious that in such cases a bottom-up approach is adopted, starting from the belief sources and concluding with the final result.

Another representative example is shown in Wang and Vassileva (2003). Each agent builds a Bayesian network for every potential partner. Each network has a root with two values. The first represents the satisfaction level while the second the nonsatisfaction degree. The satisfaction level is derived from the number of the successful interactions divided by the total number of interactions. The leaf nodes in the network represent the different capabilities that potential partners have. In this model, the recommendations of other agents are taken into consideration.

Belief and Fuzzy Models

In belief models, we meet methods that use the beliefs of an agent that another entity is trustworthy or not. In belief theory, each opinion may be represented as a triplet (belief, disbelief, uncertainty). The sum of probabilities of these three values is 1:

$$\text{belief} + \text{disbelief} + \text{uncertainty} = 1 \quad (20)$$

Agents use their and others' beliefs in order to extract the final score. This score is computed through the use of belief theory and consists of a subjective certainty of the pertinent beliefs. An example of a belief trust model one can be found in Josang (1999, 2001). Josang names his trust model "subjective logic" and combines belief theory with Bayesian probabilities. The forms used for this purpose are:

$$\text{belief} = \frac{p}{p+n+2} \quad (21)$$

$$\text{disbelief} = \frac{n}{p+n+2} \quad (22)$$

$$\text{uncertainty} = \frac{2}{p+n+2} \quad (23)$$

where p and n are the positive and negative ratings, respectively. These two parameters are also used in the beta probability density function (see Section titled "Bayesian Reputation Systems"). For the combination of beliefs, external or internal, an aggregation function is used. "Majority consensus"

functions are well-known for handling beliefs with discrete values while numerical functions are more appropriate for handing beliefs with continuous values (Ding et al., 2004). The authors in Josang (1999, 2001) use operators for the combination of opinions that are not based on Dempster-Shaffer theory as Yu and Singh do (2002).

In fuzzy models trust and reputation are described with linguistic fuzzy values. Reasoning is used in order to achieve the definition of a trust level. The most important and completed example is the system REGRET, which is described in the "Social Networks" section.

Role-Based Trust

These models are based on the notion of role and its assigned permissions to operate in a system. Hence, credentials are used to define access to a system such as identity, authentication, and so forth. Thus, an entity can be uniquely identified by the system and can obtain a specific role. Roles are used to give information about an entity. The key is the trust management mechanism that employs different languages and engines for reasoning on rules for trust establishment. It is a model used in access control systems and tries to determine the trust level of an entity based on credentials and security policies. A framework based on roles is presented in Li and Mitchell (2003).

REPUTATION AND TRUST ENGINEERING

Modeling trust requirements is the most important issue in developing efficient systems. Especially in cases where open systems are examined (e.g., MAS), this feature receives more attention. This section of our work aims to show the basic elements in which a requirements engineering procedure must be based.

As mentioned, MAS are open systems with members that change their behaviours continually. They are characterized by openness, heterogeneity, and dynamic character. Selfish agents try to locate partners in order to achieve their goals. Main issues that must be taken into consideration in MAS are:

- **Agents' identity.** Each autonomous component that interacts in a system should have a unique name and should be able to prove its identity. Identity is a requirement when agents communicate with others, because it shows to the potential partner that the component is a registered user of the system. Of course, a critical issue is the administration of the names. For example, an agent having bad reputation in a community may change its name in order to avoid the consequences.
- **Agents' communication.** Agents' communication is also important. Developers of MAS should introduce standards for communication. A security policy is necessary in order to avoid problems in the exchange of messages. Corrupted messages should be recognised by the recipients. Mechanisms that can be use for safe communication are authentication, cryptography, and so forth.
- **Agents' context.** The context in which each agent is activated should be defined. Mechanisms that take the context into consideration should be developed. This could provide efficiency in the cooperation procedure.
- **Agents' behaviour.** Agents' behaviour should be observed by the interested members and from the system. It is imperative to have the opportunity to observe and recognize bad or good behaviours in the system. Based on behaviour, trust is developed and members obtain a high reputation value in the community. Constructive trust must be promoted through the observation of interactions of an agent in the society. Also, from the systems' point of view, mechanisms

that "punish" bad behaviours should be developed.

In Wong and Sycara (1999), the authors present a number of possible threats in MAS and their potential solutions. The basic threats that MAS may meet are related to corrupted naming of agents, insecure communication channels, insecure delegation and lack of accountability. Synoptically, the proposed solutions are:

- The use of trusted agent name servers and matchmakers.
- The provision of methods for the unique identification of each agent and proofs for their identifications.
- Secure the communication channels through the authentication of messages.
- Force agents to prove their owners.
- The provision of methods through which owners of the agents may be liable for their actions.

Authors give a full description of these solutions and explain the mechanisms with which these goals will be achievable.

A methodology for agent-based software development is described in Giorgini, Massacci, and Zannone (2005) and Giorgini, Mouratidis, and Zannone (2007). In TROPOS there are phases through which the trust establishment is feasible. The first phase is the requirement phase. In this stage, the functional and the nonfunctional requirements are determined in two subphases: the *early requirements phase* and the *late requirements phase*. The key concepts in secure TROPOS are:

- **Actor.** It is an entity that represents a physical or software agent as well as a role or position, having its goals and intentions.
- **Goal.** Represents actors' interests that they wish to accomplish.
- **Plan.** It concerns a number of steps targeting to achieve a goal.
- **Resource.** It is a physical or an information entity.
- **Dependency.** Indicates that an actor depends on some other entity in order to complete their goals.

Accordingly, in TROPOS four new relationships are defined, which are: **Ownership** which indicates that an actor has a goal, **provisioning** which is the capability of an actor to achieve a goal or to have a plan or to provide some information, **trust** which indicates the belief of an actor that another entity will perform some task according to their goals and plans and **delegation** which shows that an actor delegates to some other to achieve its goals.

In TROPOS methodology, basic operations are:

- The definition of the actor's model and the dependency model. The essential actors should be recognised as well as the dependencies among them from the point of view of achieving their goals.
- The trust and delegation models. They determine the relationships between actors.
- The goal and plan models. Such models identify, from the viewpoint of actors, goals and plans that are necessary to achieve (sub)goals. Moreover, the resources needed to this procedure are recognised.

An extension to classic TROPOS is the security constraint modeling, which involves security constraints posed by the actors and the system. The architectural design development process for this extension relies on the following:

1. Secure Architectural style model.
2. Actor model, Goal/plan model, and security constraint model.

3. Capability and secure capability model.
4. Agent model.

In this chapter, due to the space constraints, we have presented only a short description of the TROPOS methodology. For a full discussion the interested reader should refer to the Giorgini et al. (2005, 2007).

FUTURE DIRECTIONS

Reputation and trust are important concepts in today's dynamic systems. However, there are some open issues that must be addressed in order to develop efficient methods for adoption in MAS.

First of all, for the construction of a theoretical framework that covers all the aspects of reputation and trust generation, manipulation and propagation is necessary. This model will set basic specifications in which developers may be based on, in order to construct efficient and productive systems. Such a framework will set the essentials that will allow the comparison of existing models. Today, this comparison is very difficult to accomplish, because the existing models come from specific domains and they are not based on a common theoretical framework.

The social network dimension in MAS presents new opportunities in reputation and trust management. However, it must be validated in real applications in order to discover its advantages and disadvantages. In social networks, there is an extensive need to deal with problems related to strategic lying and strategic coalition formation. This domain must be further studied in order to produce effective methods to deal with. Moreover, there is a need to define basic mechanisms through which opinions of members can be stored and secured in order to provide a higher security level.

Interactions of agents are held in a system under specific context. Context must be taken into consideration when defining the trust and reputation level of an entity. To the best of our knowledge, only a few works deal with this issue. Additionally, concern should be posed in trust propagation in specific contexts. Propagation allows the building of relations between all the agents communicating in a system. New methods for propagation should be developed with regard to the combination of the aforementioned models such as statistical functions, belief and fuzzy theory.

CONCLUSION

This chapter introduces the reader to the domain of reputation and trust in Multi-agent systems. It presents the existing reputation and trust quantification methods that are used in commercial and research applications. We show the importance of these two concepts especially in open systems where control over the actions of agents is limited. Also, the environment, where agents act, may change at any time due to the nature of the involved entities, which are autonomous components trying to serve their owners. For this reason, they are selfish and change their behavior subject to new conditions. A lot of models have been proposed for reputation and trust extraction in specific domains. We shortly present the most important of them, giving their basic characteristics. Certain models are very simple, while others are more sophisticated and utilize statistical functions, belief or fuzzy theory. Finally, we discuss key contribution in the domain of trust and reputation engineering. It is a critical field that leads to more efficient and productive systems. The engineering process must drive developers to construct systems that pay special attention to issues that ensure secure communication and fair ratings for all.

Reputation and trust will play an important role in future systems. Such concepts will be

extensively adopted and used for gaining access to information sources in open environments like MAS and the Semantic Web.

REFERENCES

Abdul-Rahman, A., & Hailes, S. (2000). Supporting trust in virtual communities. In *Proceedings of the Hawaii International Conference on System Services* (Vol. 6, pp. 6007).

Artz, D., & Gil, Y. (2006). *Survey of trust in computer science and the Semantic Web.* Submitted for publication, Information Sciences Institute, University of Southern California. Retrieved April 3, 2008, from http://www.isi.edu/~dono/pdf/artz06survey.pdf

Ben-Gal, I. (2007). Bayesian networks. In F. Ruggeri, F. Faltin, & R. Kenett (Eds.), *Encyclopedia of statistics in quality and reliability.* John Wiley & Sons.

Boella, G., & Van Der Torre, L. (2005). Normative multiagent systems and trust dynamics. *Trusting agents for trusting electronic societies, LNAI 3577,* (pp. 1-17).

Boella, G., Van Der Torret, L., & Verhagen, H. (2006). Introduction to normative multiagent systems. *Computational & Mathematical Organisation Theory, 12*(2-3), 71-79.

Carter, J., & Ghorbani, A. A. (2004). Value centric trust in multiagent systems. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence,* (pp. 3-9).

Castelfranchi, C., & Falcone, R. (1998). Social trust: Cognitive anatomy, social importance, quantification, and dynamics. In *Proceedings of the First International Workshop on Trust,* Paris, France, (pp. 72-79).

Dellarocas, C. (2000). Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the ACM Conference of Electronic Commerce,* (pp. 150-157).

Dempster, A. P. (1968). A generalisation of BaysianBayesian inference. *Journal of the Royal Statistical Society, Series B, 30,* 205-247.

Deriaz, M. (2006). *What is trust? My own point of view.* University of Geneva. Retrieved April 3, 2008, from http://cui.unige.ch/ASG/publications/TR2006/

Ding, L., Kolari., P., Ganjugunte, S., Finin, T., & Joshi, A. (2004). Modeling and evaluating trust network inference. In *Proceedings of the 7th International Workshop on Trust in Agent Societies at AAMAS 2004,* New York.

Doran, J. E., Franklin, S., Jennings, N. R., & Norman, T. J. (1997). On cooperation in multi-agent systems. *The Knowledge Engineering Review, 12*(3), 309-314.

Durfee, E. H., & Lesser, V. (1989). Negotiating task decomposition and allocation using partial global planning. In L. Gasser & M. Huhns (Eds.), *Distributed artificial intelligence* (Vol. 2, pp. 229-244). San Francisco: Morgan Kaufmann.

Giorgini, P., Massacci, F., & Zannone, N. (2005). Security and trust requirements engineering. *Foundations of security analysis and design III—tutorial lectures, LNCS 3655* (pp. 237-272). Springer-Verlag.

Giorgini, P., Mouratidis, H., & Zannone, N. (2007). Modelling security and trust with secure TROPOS. *Integrating security and software engineering: Advances and future vision.* Hershey, PA: Idea Group.

Grandison, T., & Sloman, M. (2000). A survey of trust in Internet applications. *IEEE Communications Surveys and Tutorials, 4th Quarter, 3*(4), 2-16.

Guha, R., Kumar, R., Raghavan, P., & Tomkins, A. (2004). Propagation of trust and distrust. *In Proceedings of the 13th International Conference on World Wide Web (WWW 2004),* New York, (pp. 403-412).

Huynh, D., Jennings, N. R., & Shadbolt, N. R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems, 13*(2), 119-154.

Josang, A. (1999). Trust-based decision making for electronic transactions. In L. Yngstrm & T. Scensson (Eds.), *Proceedings of the 4th Nordic Workshop on Secure Computer Systems.* Stockholm, Sweden: Stockholm University Report 99-005.

Josang, A. (2001). A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 9*(3), 279-311.

Josang, A., & Ismail, R. (2002). The Beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce,* Slovenia, (pp. 324-337).

Josang, A., Roslam, I., & Colin, B. (2006). A survey of trust and reputation systems for online service provision. *Decision support systems.*

Li, N., & Mitchell, J. C. (2003). RT: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III).*

Lindsay, Y., Yu, W., Han, Z., & Ray Liu, K. J. (2006). Information theoretic framework of trust modelling and evaluation for the ad hoc networks. *IEEE International Journal on Selected Areas in Communications, 24*(2), 305-317.

McKnight, D. H., & Chervany, N. L. (1996). The meanings of trust (Tech. Rep.). University of Misessota, Management Information Systems Research Center. Retrieved April 3, 2008, from http://www.misrc.umn.edu/workingpapers/

Melaye, D., & Demazeau, Y. (2005). Bayesian dynamic trust model. In *Proceedings of Multi-agent Systems and Applications IV: 4th International Central and Eastern European Conference on Multi-agent Systems, CEEMAS 2005,* (pp. 480-489). Springer-Verlag, LNCS 3690.

Mui, L., Halberstadt, A., & Mohtashemi, M. (2002). Notions of reputation in multi-agent systems: A review. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems,* Bologna, Italy, (pp. 280-287).

Mui, L., Mohtashemi, M., Ang, C., Szolovtis, P., & Halberstadt, A. (2001). Ratings in distributed systems: A bayesian approach. In *Proceedings of the Workshop on Information Technologies and Systems (WITS),* Miami, Fl.

Nwana, H. S. (1996). Software agents: An overview. *The Knowledge Engineering Review, 11*(3), 205-244.

Osman, N. (2006). *Formal specification and verification of trust in multi-agent systems.* School of Informatics, University of Edinburgh. Retrieved April 3, 2008, from http://homepages.inf.ed.ac.uk/s0233771/trust.pdf

Poslad, S., Calisti, M., & Charlton, P. (2002). Specifying standard security mechanisms in multi-agent systems. In *Proceedings of the Workshop on Deception, Fraud and Trust in Agent Societies, AAMAS 2002,* Bologna, Italy, (pp. 122-127).

Pujol, J. M., & Sanguesa, R. (2002). Reputation measures based on social networks metrics for multi agent systems. In *Proceedings of the 4th Catalan Conference on Artificial Intelligence CCIA-01,* Barcelona, Spain, (pp. 205-213).

Ramchourn, S. D., Huynh, D., & Jennings, N. R. (2004). Trust in multi-agent systems. *The Knowledge Engineering Review, 19*(1), 1-25.

Rasmusson, L., & Jansson, S. (1996). Simulated social control for secure Internet commerce. In C. Meadows (Ed.), *Proceedings of the 1996 New Security Paradigms Workshop*, (pp. 18-26).

Rubiera, J. C., Molina Lopez, M. J., & Muro D. J. (2001). A fuzzy model of reputation in multi-agent systems. In *Proceedings of the 5th International Conference on Autonomous Agents*, Montreal, Quebec, Canada, (pp. 25-26).

Sabater, J., & Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, (pp. 475-482).

Schillo, M., Funk, P., & Rovatsos, M. (2000). Using trust for detecting deceitful agents in artificial societies. *Applied Artificial Intelligence, Special Issue on Trust, Deception and Fraud in Agent Societies, 14*(8), 825-848.

Shafer, G. (1976). *A mathematical theory of evidence*. Princeton University Press.

Sycara, K. P. (1998). Multiagent systems. *Artificial Intelligence Magazine, 19*(2), 79-92.

Wang, Y., Hori, Y., & Sakurai, K. (2006). On securing open networks through trust and reputation–architecture, challenges and solutions. In *Proceedings of the 1st Joint Workshop on Information Security*, Seoul, Korea.

Wang, Y., & Vassileva, J. (2003). Bayesian network-based trust model. In *Proceedings of IEEE International Conference on Web Intelligence*, Hallifax, Canada.

Whitby, A., Josang, A., & Indulska, J. (2004). Filtering out unfair ratings in bayesian reputation systems. In *Proceedings of the AAMAS 2004*, New York.

Wong, H. C., & Sycara, K. (1999). Adding security and trust to multi-agent systems. In *Proceedings of Autonomous Agents '99 Workshop on Deception, Fraud, and Trust in Agent Societies*, (pp. 149-161).

Yu, B., & Singh, P. M. (2002). An evidential model of distributed reputation management. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, (pp. 294-301).

Yu, B., & Singh, P. M. (2003). Detecting deception in reputation management. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia, (pp. 73-80).

Zadeh, L. A. (1989). Knowledge representation in fuzzy logic. *IEEE Transactions on Knowledge and Data Engineering, 1*(1), 89-100.

This work was previously published in Intelligence Integration in Distributed Knowledge Management, edited by D. Król and N. Nguyen, pp. 132-153, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 7.15
Improving Credibility of Machine Learner Models in Software Engineering

Gary D. Boetticher
University of Houston – Clear Lake, USA

ABSTRACT

Given a choice, software project managers frequently prefer traditional methods of making decisions rather than relying on empirical software engineering (empirical/machine learning-based models). One reason for this choice is the perceived lack of credibility associated with these models. To promote better empirical software engineering, a series of experiments are conducted on various NASA datasets to demonstrate the importance of assessing the ease/difficulty of a modeling situation. Each dataset is divided into three groups, a training set, and "nice/nasty" neighbor test sets. Using a nearest neighbor approach, "nice neighbors" align closest to same class training instances. "Nasty neighbors" align to the opposite class training instances. The "nice", "nasty" experiments average 94% and 20% accuracy, respectively. Another set of experiments show how a ten-fold cross-validation is not sufficient in characterizing a dataset. Finally, a set of metric equations is proposed for improving the credibility assessment of empirical/machine learning models.

INTRODUCTION

Software Project Management: State-of-Practice

Software project management has improved over the years. For example, the Standish Group, a consulting company, which has been studying IT management since 1994 noted in their latest release of the *Chaos Chronicles* (The Standish Group, 2003) that, "2003 Project success rates improved by more than 100 percent over the 16 percent rate from 1994." Furthermore, "Project

failures in 2003 declined to 15 percent of all projects. This is a decrease of more than half of the 31 percent in 1994."

Even with these successes, there are still significant opportunities for improvement in software project management. Table 1 shows several "state-of-practice" surveys collected in 2003 from IT companies in the United States (The Standish Group, 2003); South Africa (Sonnekus & Labuschagne, 2003); and the United Kingdom (Sauer & Cuthbertson, 2003).

According to the *Chaos Chronicles* (The Standish Group, 2003), *successful projects* refers to projects that are completed on time and within budget with all features fully implemented; *project challenged* means that the projects are completed, but exceed budget, go over time, and/or are lacking some/all of the features and functions from the original specifications; and *project failures* are those projects which are abandoned and/or cancelled at some point.

Applying a weighted average to Table 1 results in 34% of the projects identified as successful, 50% are challenged, and 16% end up in failure. Thus, about one-third of the surveyed projects end up as a complete success, half the projects fail to some extent, and one sixth end up as complete failures. Considering the role of computers in various industries, such as the airlines and banking, these are alarming numbers.

From a financial perspective,[1] the lost dollar value for U.S. projects in 2002 is estimated at $38 billion with another $17 billion in cost overruns for a total project waste of $55 billion against $255 billion in project spending (The Standish Group, 2003). Dalcher and Genus (2003) estimate the cost for low success rates at $150 billion per year attributable to wastage arising from IT project failures in the Unites States, with an additional $140 billion in the European Union. Irrespective of which estimate is adopted, it is evident that software project mismanagement results in an annual waste of billions of dollars.

Empirical Software Engineering

One of the keys for improving the chances of project development success is the application of empirical-based software engineering. **Empirical-based software engineering** is the process of collecting software metrics and using these metrics as a basis for constructing a model to help in the decision-making process.

Two common types of software metrics are project and product metrics. **Project metrics** refer to the estimated time, money, or resource effort needed in completing a software project. The Standish Group (2003) perceives software cost estimating as the *most effective way to avoid cost and schedule*. Furthermore, several studies (Jones, 1998; The Standish Group, 2003) have shown that by using software cost-estimation techniques, the probability of completing a project successfully doubles. Thus, estimating the schedule, cost, and resources needed for the project is paramount for project success.

Product metrics are metrics extracted from software code and are frequently used for soft-

Table 1. State-of-practice surveys

Year	Successful	Challenged	Failure	Projects Surveyed
United States (Chaos Chronicles III)	34%	51%	15%	13,522
South Africa	43%	35%	22%	1,633
United Kingdom	16%	75%	9%	421

ware defect prediction. Defect prediction is a very important area in the software development process. The reason is that a software defect dramatically escalates in cost over the software life cycle. During the coding phase, finding and correcting defects costs $977 per defect (Boehm & Basili, 2001). In the system-testing phase, the cost jumps to $7,136 per defect (Boehm et al., 2001). If a defect survives to the maintenance phase, then the cost to find and remove increases to $14,102 (Boehm et al., 2001).

From an industry perspective, Tassey (2002) estimates that the annual cost of software defects in the United States is $59.5 billion and that "feasible" improvements to testing infrastructures could reduce the annual cost of software defects in the United States by $22.2 billion. The Sustainable Computing Consortium (SCC), an academic, government, and business initiative to drive IT improvements, estimates that from a global perspective, defective computer systems cost companies $175 billion annually. Thus, there are major financial incentives for building models capable of predicting software defects.

There are primarily two methods for constructing models in empirical software engineering. The first adopts an **empirical approach** which emphasizes direct observation in the modeling process and results in one or more mathematical equations. Common examples of this approach include COCOMO I (Boehm, 1981), COCOMO II (Boehm et al., 2000), function points (Albrecht, 1979; Albrecht & Gaffney, 1983), and SLIM (Putnam, 1978) for effort estimation. The second method automates the observation process by using a **machine learning** approach to characterize relationships. Examples of machine learners include Bayesian belief networks (BBN), case-based reasoners (CBR), decision tree learners, genetic programs, and neural networks. The by-product of applying these learners include mathematical equations, decision trees, decision rules, and a set of weights as in the case of a neural network. For the purpose of this chapter, a cursory description will suffice; further details regarding the application of machine learners in software engineering may be found at Khoshgoftaar (2003), Pedrycz (2002), and Zhang and Tsai (2005).

Although the financial incentives are huge, empirical software engineering has received modest acceptance by software practitioners. A project manager may estimate a project by using a human-based approach, an empirical approach, or a machine learning approach. Even though popular models, such as COCOMO or function points, have existed for 25 or more years their application is rather low. In 2004, Jørgensen (2004) compiled a series of studies regarding the frequency of human-based estimation and found it is used 83% (Hihn & Habib-Agahi, 1991), 62% (Heemstra & Kusters, 1991), 86% (Paynter, 1996), 84% (Jørgensen, 1997), and 72% (Kitchenham, Pfleeger, McColl, & Eagan, 2002). It is evident that human-based estimation is the dominant choice relative to empirical or machine learning-based estimation. Furthermore, Jørgensen (2004) states the empirical-based estimation ranges from about 7 to 26% and machine learning-based estimation is only about 4 to 12%.

A key question is *"Why haven't empirical-based models and particularly machine learner models gained greater acceptance by software practitioners?"*

One possible answer to this question is the difficulty in assessing the credibility of these models. A model may boast accurate results, but these results may not be realistic. Thus, the lack of perceived credibility makes it difficult for software practitioners to adopt a new technology within their software development process.

It may be argued that "credibility processes" exist in the form of *n*-fold cross validation and accuracy measures. Unfortunately, none of these approaches address the issue of a dataset's difficulty (how easy or hard it is to model). An empirical model that produces spectacular results may be the consequence of a test set that closely resembles a training set rather than the capabil-

ity of empirical learner. The outcome is a set of unrealistic models that is unlikely to be adopted by industry. Without a mechanism for assessing a dataset's difficulty, empirical/machine learning models lose credibility.

This chapter introduces several credibility metrics which may be incorporated into the model formulation process with the goal of giving greater credibility. Before introducing these metrics, a series of experiments are conducted to demonstrate some of the difficulties in using a dependent variable for making sampling decisions. The **dependent variable**, also known as the **class variable**, is the attribute we are trying to predict from a set of independent (non-class) variables. The initial set of experiments demonstrates the importance of sampling on non-class attributes. These experiments examine the "best case" versus "worst case" for a NASA-based defect repository dataset.

Next, a second set of experiments demonstrates that a *ten*-fold cross validation may not be sufficient in building realistic models.

After demonstrating some of the inadequacies of current validation processes, several credibility metrics are introduced. These metrics measure the *difficulty* of a dataset. Adoption of these metric equations will lead to more realistic models, greater credibility to models, and an increased likelihood that software practitioners will embrace empirical software engineering.

RELATED RESEARCH

This section provides a general overview of different sampling/resampling techniques, a description of the K-nearest neighbor (KNN) algorithm, and an overview of assessing models for accuracy.

Traditional sampling typically adopts a **random**, **stratified**, **systematic**, or **clustered** approach. The randomized approach, which is the simplest, randomly selects tuples to be in the test set. One way to improve estimates obtained from random sampling is by arranging the population into strata or groups. A non-class attribute (e.g., age, gender, or income) is used to stratify samples. Systematic sampling orders data by an attribute, then selects every i^{th} element on the list afterwards. Cluster sampling is a method of selecting sampling units in which the unit contains a cluster of elements (Kish, 1965). Examples of cluster samples may include all students of one major from all university students or residents from a particular state.

When extracting samples for a test set, systematic and cluster sampling focus on the class (or dependent) variable. Stratified sampling may use a non-class variable, but typically this is limited to at most one of the non-class variables and does not consider the non-class attribute in light of the class attribute. The problem is that non-class (independent) variables contain a lot of vital information which needs to be considered when partitioning tuples into training and test sets.

An ***n*-fold cross validation** is a resampling method for validating a model. For this technique, data is partitioned into *n*-classes, and *n* models are constructed with each of the *n*-classes rotated into the test set. *N*-fold cross validation addresses the issue of data distribution between training and test sets, but does not consider difficulty in modeling the training data.

The **K-nearest neighbor** (**KNN**) algorithm is a supervised learning algorithm which classifies a new instance based upon some distance formula (e.g., Euclidean). The new instance is classified to a category relative to some majority of K-nearest neighbors. Traditionally, the KNN algorithm is viewed as a machine learner, rather than a method for dividing training and test data.

Regarding approaches for measuring accuracy, Shepperd, Cartwright, and Kadoda (2000) discuss the merits of various methods for measuring accuracy including *TotalError*, *TotalAbsoluteError*, *TotalRelativeError*, *BalancedMMRE*, *MMRE*, *Pred(X)*, *Mean Squared Error*, and R^2. There is validity in using one or more of these accuracy

methods. However, they do not provide any information regarding the difficulty in modeling a dataset.

A SIMPLE EXAMPLE

To illustrate why it is important to consider the relationship between non-class attributes over a training and test set, consider the dataset in Table 2. It consists of three attributes: source lines of code (SLOC); v(g), also known as cyclomatic complexity, which equals the number of decision points within a program plus one; and *defects*. For this example, defects refer to the number of software defects present in a software component. If a software component has zero defects, then it is classified as an *A*, otherwise it is classified as a *B*.

Figure 1 plots these points. The points form 4 clusters of *A*s and *B*s, respectively.

If the goal is to build a model for predicting software defects, a systematic sampling approach may be applied, which generates the training and test sets in Tables 3a and 3b.

Applying a nearest neighbor approach to the non-class attributes, it is clear that the *A*s in the test set match the *A*s in the training set. This is depicted as arrows between Tables 3a and 3b. The same is true for those defects in the *B* class. It is expected that such an experiment would produce very good results irrespective of the machine learner selected. Figure 2 highlights training samples with gray boxes.

Suppose a second experiment is conducted using stratified sampling and produces the training and test sets in Tables 4a and 4b.

Applying a nearest neighbor approach to the non-class attributes, it is clear that the *A*s in the test set match a *B* instance in the training set. Also, the *B*s in the test set match an *A* instance in the training set. It is expected that such an experiment would generate very poor results irrespective of the machine learner selected. Figure 3 highlights training samples with gray boxes.

By ignoring nearest neighbors when building models, success/failure may be a function of the nearest neighbor distribution rather than the capability of the algorithm/machine-learner.

To illustrate the importance of considering nearest neighbor in training/test distribution, a series of experiments are conducted using NASA-based defect data. Although the focus is on defect data, the idea easily extends to other types of software engineering data (e.g., effort estimation data).

NASA DATASETS

To demonstrate how non-class attributes dramatically impact the modeling process, a series of experiments are conducted against five NASA defect datasets. These experiments fall into two categories. The "nice" experiments use a test set where the non-class attributes of the test data have nearest-neighbors in the training set and are the same class value. The "nasty" experiments use a test set where the non-class attributes of the test data have nearest-neighbors in the training set with an opposite class value.

All experiments use five public domain defect datasets from the NASA Metrics Data Program (MDP) and the PROMISE repository (Shirabad & Menzies, 2005). These five datasets, referred to as CM1, JM1, KC1, KC2, and PC1, contain static code measures (e.g., Halstead, McCabe, LOC) along with defect rates. Table 5 provides a project description for each of these datasets.

Each dataset contains 21 software product metrics based on the product's size, complexity, and vocabulary. The size metrics include *total lines of code, executable lines of code, lines of comments, blank lines, number of lines containing both code and comments,* and *branch count.* Another three metrics are based on the product's complexity. These include *cyclomatic complexity, essential complexity,* and *module design*

Table 2. Simple dataset

SLOC	v(g)	Defects
81	13	A
87	13	A
182	33	A
193	32	A
53	10	B
58	10	B
140	30	B
150	27	B

Figure 1. Plot of tuples from Table 2

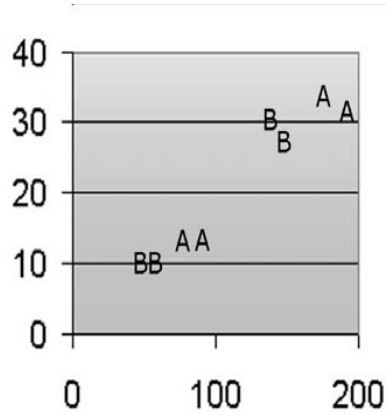


Table 3a. Training data

SLOC	v(g)	Defects
81	13	A
182	33	A
58	10	B
150	27	B

Table 3b. Test data

SLOC	v(g)	Defects
87	13	A
193	32	A
53	10	B
140	30	B

complexity. The other 12 metrics are vocabulary metrics. The vocabulary metrics include *Halstead length, Halstead volume, Halstead level, Halstead difficulty, Halstead intelligent content, Halstead programming effort, Halstead error estimate, Halstead programming time, number of unique operators, number of unique operands, total operators,* and *total operands.*

The class attribute for each dataset refers to the propensity for defects. The original MDP dataset contains numeric values for the defects, while the PROMISE datasets convert the numeric

Figure 2. Plot of training and test sets (Square boxes are training samples)

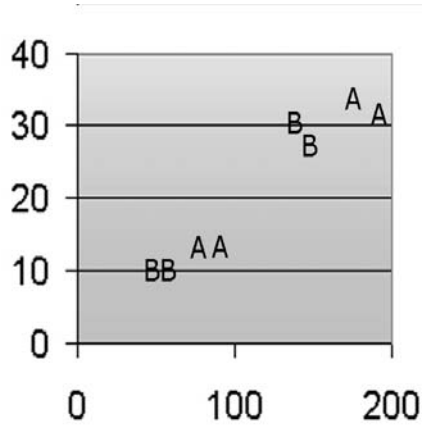


Table 4a. Training data

SLOC	v(g)	Defects
81	13	A
87	13	A
140	30	B
150	27	B

Table 4b. Test data

SLOC	v(g)	Defects
182	33	A
193	32	A
53	10	B
58	10	B

values to Boolean values where *TRUE* means a component has one or more defects and *FALSE* equates to zero defects. The reason for the conversion is that the numeric distribution displayed signs of an implicitly data-starved domain (many data instances, but few of interest) where less than 1% of the data has more than five defects (Menzies, 2005a).

DATA PRE-PROCESSING

Data pre-processing removes all duplicate tuples from each dataset along with those tuples that have questionable values (e.g., LOC equal to 1.1). Table 6 shows the general demographics of each of the datasets after pre-processing.

NEAREST NEIGHBOR EXPERIMENTS

Training and Test Set Formulation

To assess the impact of nearest-neighbor sampling upon the experimental process, 20 experiments are conducted on each of the five datasets.

For each experiment, a training set is constructed by extracting 40% of data from a given dataset. Using stratified sampling to select 40% of the data maintains the ratio between defect/non-defect data. As an example, JM1 has 8,911 records, 2,007 (22.5%) of which have one or more defects. A corresponding training set for the JM1 project contains 3,564 records, 803 (22.5%) of which are classified as having one or more defects (TRUE).

Figure 3. Plot of training and test sets (Square boxes are training samples)

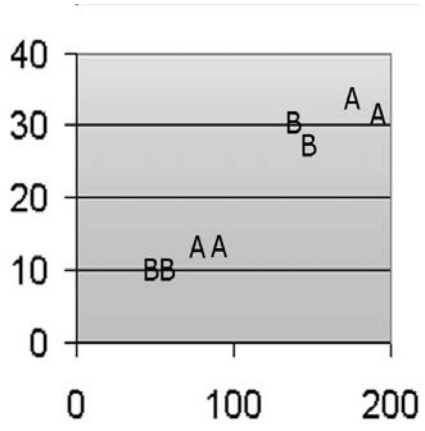


Table 5. Project description for each dataset

Project	Source Code	Description
CM1	C	NASA spacecraft instrument
KC1	C++	Storage management for receiving/processing ground data
KC2	C++	Science data processing. No software overlap with KC1.
JM1	C	Real-time predictive ground system
PC1	C	Flight software for earth orbiting satellite

It could be argued that a greater percentage (more than 40%) of the data could be committed to the training set. There are several reasons for choosing only 40%. First, Menzies claims that only a small portion of the data is needed to build a model (Menzies, Raffo, Setamanit, DiStefano, & Chapman, 2005b). Second, since the data is essentially a two-class problem, there was no concern about whether each class would receive sufficient representation. Finally, it is necessary to insure that there is a sufficient amount of test data for assessing the results.

Once a training set is established, the remaining 60% of the data is partitioned into two test groups. Prior to splitting the test data, all of the non-class attributes are normalized by dividing each value by the *Difference* ($Maximum_k - Minimum_k$ for each column k). This guarantees that each column receives equal weighting. The next step loops through all the test records. Each test record is compared with every training record to determine the minimum *Euclidean Distance* for all of the non-class attributes. If the training and test tuples with the smallest Euclidean Distance share the same class values (TRUE/TRUE or FALSE/FALSE), then the test record is added to the "nice neighbor" test set, otherwise add it to the "nasty neighbor" test set. Figure 4 shows the corresponding algorithm.

Essentially, this is the K-nearest neighbor algorithm that determines a test tuple's closest match in the training set. Nearest neighbors from the same class are considered "nice", otherwise they are classified as "nasty."

Table 6. Data pre-processing demographics

Project	Original Size	Size w/ No Bad, No Dups	0 Defects	1+ Defects	% Defects
CM1	498	441	393	48	10.9%
JM1	10,885	8911	6904	2007	22.5%
KC1	2109	1211	896	315	26.0%
KC2	522	374	269	105	28.1%
PC1	1109	953	883	70	7.3%

Figure 4. Nice/nasty neighbor algorithm

```
For j=1 to test.record_count
  minimumDistance = 9999999
  For i=1 to train.record_count
    Dist = 0
    For k=1 to train.column_count - 1
      Dist = Dist + (train_ik - test_jk)^2
    end k
    if (abs(Dist) < abs(minimumDistance))
     then if Train_i.defect = Test_j.defect
       then minimumDistance = Dist
       else minimumDistance = -Dist
  end i;
  if minimumDistance > 0
    then Add_To_Nice_Neighbors
  if minimumDistance < 0
    else Add_To_Nasty_Neighbors
  if minimumDistance = 0
    then if Train_i.defect = Test_j.defect
     then Add_To_Nice_Neighbors
     else Add_To_Nasty_Neighbors
end j;
```

All experiments use the training data to build a model between the non-class attributes (e.g., *size*, *complexity*, or *vocabulary*) and the class attribute *defect* (which is either *True* or *False*).

After constructing 300 datasets (one training set and two tests sets; 20 trials per software project; five software projects), attention focuses on data mining tool selection.

Data Mining Tool Selection

Since the data contains 20-plus attributes and only two class values (TRUE/FALSE), the most reasonable data mining tool in this situation is a decision tree learner. A decision tree selects an attribute which best divides the data into two homogenous groups (based on class value).

The split selection recursively continues on the two or more subtrees until all children of a split are totally homogenous (or the bin dips below a prescribed threshold). Decision tree learners are described as greedy in that they do not look ahead (two or more subtree levels) due to the associated computational complexity.

One of the most popular public domain data mining tools is the Waikato environment for knowledge analysis (Weka) tool (Witten & Franks, 2000). Weka is an open-source machine learning workbench. Implemented in Java, Weka incorporates many popular machine learners and is widely used for practical work in machine learning. According to a recent KDD poll (KDD Nuggets Web site, 2005), Weka was rated number two in terms of preferred usage as compared to other commercial and public domain tools.

Within Weka, there are many learners available. The experiments specifically use the naïve Bayes and J48 learners for analysis. There are reasons for adopting these tools. First, these tools performed very well in more than 1,000 data mining experiments conducted by the author. Second, success in using these particular learners was noted by Menzies et al. (2005b) in their analysis of the NASA defect repositories.

A naïve Bayes classifier uses a probabilistic approach to assign the most likely class for a particular instance. For a given instance *x*, a naïve Bayes classifier computes the conditional probability

$$P\,(C = c_i \mid x) = P(C = c_i \mid A_1 = a_{i1},\ldots A_n = a_{in}) \quad (1)$$

for all classes c_i and tries to predict the class which has the highest probability. The classifier is considered naïve (Rish, 2001) since it assumes that the frequencies of the different attributes are independent.

A second learner, J48, is based on Quinlan's C4.5 (Quinlan, 1992).

Assessment Criteria

The assessment criterion uses four metrics in all experiments to describe the results. They are:

- **PD**, which is the probability of detection. This is the probability of identifying a module with a fault divided by the total number of modules with faults.
- **PF**, which is the probability of a false alarm. This is defined as the probability of incorrectly identifying a module with a fault divided by the total number of modules with no faults.
- **NF**, which is the probability of missing an alarm. This is defined as the probability of incorrectly identifying a module where the fault was missed divided by the total number of modules with faults.
- **Acc**, which is the accuracy. This is the probability of correctly identifying faulty and non-faulty modules divided by the total number of modules under consideration.

Each of these metrics is based on simple equations constructed from Weka's **confusion matrix** as illustrate by Table 7.

PD is defined as:

$$PD = A / (A+B) \quad (2)$$

PF is defined as:

$$PF = C / (C + D) \quad (3)$$

NF is defined as:

$$NF = A / (A + B) \quad (4)$$

and *Acc* is defined as:

$$Acc = (A + D) / (A + B + C + D) \quad (5)$$

Based on the example in Table 7, the corresponding values would be:

$$PD = 50 / (50 + 200) = 20\% \quad (6)$$

$$PF = 100 / (100 + 900) = 10\% \quad (7)$$

$$NF = 200 / (200 + 50) = 80\% \quad (8)$$

$$Acc = (50 + 900) / (50 + 100 + 200 + 900) = 76\% \quad (9)$$

Results

Table 8 shows the accuracy results of the 20 experiments per project. As might be expected, the "nice" test set did very well for all five projects for both machine learners averaging about 94% accuracy. Its counterpart, the "nasty" test set, did not fare very well, averaging about 20% accuracy.

It is interesting to note that the JM1 dataset, with 7 to 20 times more tuples than any of the other projects, is above the overall average for the "nice" datasets, and below the overall average on the "nasty" datasets. Considering the large number of tuples in this dataset and how much of the solution space is covered by the JM1 dataset, it would seem that a tuple in the test set would have difficulty aligning to a specific tuple in the training set.

Regarding *PD*, the results as expressed in Table 9 for the "nice" test set are superior to the "nasty" test set for the learners. An overall weighted average is preferred over a regular average in order not to bias the results towards those experiments with very few defect samples.

The results in Table 9 can be misleading. Seventy-six of the 100 "nice" test sets contained zero defect tuples. Of the remaining 24 "nice" test sets, only 2 of these 24 had 20 or more samples with defects.

The "nice" test set did very well at handling false alarms as depicted in Table 10. The "nasty" test set triggered alarms about 18 to 37% of the time depending upon learner. Overall, the sample size is small for the "nasty" test sets. Ninety percent (from the 100 experiments) of the "nasty" test sets contain zero instances of non-defective data. For the remaining 10 "nasty" datasets, only three contain 10 or more instances of non-defective modules.

To better understand these results, consider Tables 11 and 12. These tables show the weighted averages (rounded) of all confusion matrices for all 100 experiments (20 per test group). In the "nice" test data, 99.6% are defined as having no defects (FALSE). Less than 1% of the tuples actually contain defects. Although the "nice" test sets fared better than the "nasty" test sets regarding defect detection, the relatively few samples having one or more defects in the "nice" test sets discount the results.

Analyzing the "nasty" datasets in Tables 11 and 12 reveal that 97.2% (e.g., (50+249)/(50+249+2+7)) of the data contains one or more defects.

Table 7. Definition of the confusion matrix

	A Defect is Detected.	**A Defect is not Detected.**
A Defect is Present.	**A = 50** Predicted=TRUE Actual= TRUE	**B = 200** Predicted= FALSE Actual= TRUE
A Defect is not Present.	**C = 100** Predicted= TRUE Actual=FALSE	**D = 900** Predicted= FALSE Actual= FALSE

Table 8. Accuracy results from all experiments

	"Nice" Test Set		"Nasty" Test Set	
	J48	**Naïve Bayes**	**J48**	**Naïve Bayes**
CM1	97.4%	88.3%	6.2%	37.4%
JM1	94.6%	94.8%	16.3%	17.7%
KC1	90.9%	87.5%	22.8%	30.9%
KC2	88.3%	94.1%	42.3%	36.0%
PC1	97.8%	91.9%	19.8%	35.8%
Average	**94.4%**	**93.6%**	**18.7%**	**21.2%**

Table 9. Probability of detection results

	"Nice" Test Set		"Nasty" Test Set	
	J48	**Naïve Bayes**	**J48**	**Naïve Bayes**
CM1	0.0%	0.0%	5.9%	37.4%
JM1	71.9%	92.9%	14.9%	16.4%
KC1	45.8%	87.5%	22.7%	31.0%
KC2	100.0%	100.0%	42.2%	36.0%
PC1	11.7%	75.0%	10.9%	30.8%
Overall Weighted Average	**45.6%**	**60.8%**	**16.8%**	**20.0%**

Table 10. Probability of false alarms results

	"Nice" Test Set		"Nasty" Test Set	
	J48	**Naïve Bayes**	**J48**	**Naïve Bayes**
CM1	2.6%	11.7%	0.0%	50.0%
JM1	5.1%	5.0%	61.7%	66.1%
KC1	9.0%	12.5%	46.4%	91.7%
KC2	11.8%	5.9%	0.0%	50.0%
PC1	1.7%	7.9%	1.9%	70.6%
Overall Weighted Average	**5.4%**	**6.3%**	**18.5%**	**37.1%**

Table 11. Confusion matrix, nice test set (rounded)

J48		Naïve Bayes	
2	3	3	2
58	1021	68	1011

Table 12. Confusion matrix, nasty test set (rounded)

J48		Naïve Bayes	
50	249	60	241
2	7	3	5

Table 13. KC1 data, KNN=3, 60% of training data

			Accuracy	
Neighbor Description	**# of TRUEs**	**# of FALSEs**	**J48**	**Naïve Bayes**
PPP	None	None	NA	NA
PPN	0	354	88	90
PNP	0	5	40	20
NPP	None	None	NA	NA
PNN	3	0	100	0
NPN	13	0	31	100
NNP	110	0	25	28
NNN	None	None	NA	NA

Referring back to the right-most column of Table 6, the percentage of defects to the total number of modules ranged from 7.3 to 28.1%. Considering that all training sets maintained their respective project ratio of defects to total components, it is quite surprising that the "nice" and "nasty" datasets would average such high proportions of non-defective and defective components, respectively.

To better understand these results, two additional experiments are conducted using the KC1 dataset. The first experiment randomly allocates 60% of the data to the training set, while the second allocates 50% of the data. Both experiments maintain a defect/non-defective ratio of 26% (see Table 6). For both experiments, the test data is divided into eight groups using a three-nearest neighbor approach. For each test vector, its three closest neighbors from the training set are determined. These neighbors are ranked based on first, second, to third closest neighbor. A "P" means that there is a positive match (same class), and an "N" means there is a negative match (opposite class). Thus, a "PPN" means that the first and second closest matches are from the same class and the third closest match is from the opposite

Table 14. KC1 data, KNN=3, 50% of training data

			Accuracy	
Neighbor Description	**# of TRUEs**	**# of FALSEs**	**J48**	**Naïve Bayes**
PPP	0	19	89	84
PPN	0	417	91	91
PNP	0	13	23	0
NPP	None	None	NA	NA
PNN	None	None	NA	NA
NPN	18	0	100	100
NNP	132	0	20	20
NNN	7	0	0	29

Table 15. Duplicate/no duplicate experimental results

	Accuracy Average of 29 Runs	
	With Duplicates	**No Duplicates**
CM1	88.07%	87.46%
JM1	79.68%	76.56%
KC1	84.29%	74.03%
KC2	81.65%	76.22%
PC1	93.43%	91.65%

class. Thus, the best case would be a "PPP" where the three closest training vectors are all from the same class.

Tables 13 and 14 show the results from these experiments. It is interesting to note that all eight bins contain homogenous (all TRUEs, or all FALSEs) data. There is a general trend for the bin configuration to change from all non-defective tuples (all FALSEs) to all defective tuples (TRUEs) as the neighbor status changes from all positives (PPP) to all negatives (NNN). Also, the accuracy seems positively correlated to the nearest neighbor classifications.

These last two sub-experiments confirm the results achieved in Tables 11 and 12.

Ten-Fold/Duplicates Experiment

The next experiment demonstrates that a *ten*-fold cross validation does not provide a total perspective regarding the validation of a dataset.

This experiment uses the five NASA datasets described in the earlier sections. For each dataset, two types of experiments are conducted: the first uses the original dataset (less bad data) with duplicates, and a second where duplicates are removed. Note that each original dataset had only one bad data sample. Each experiment uses a *ten*-fold cross validation for each of the 20 trials. A defect prediction model is constructed based on the C4.5 learner from Weka (J48) using the default settings.

Table 15 shows the results from these experiments. For all five NASA datasets, the learner produces a better model with the inclusion of duplicates. A t-test shows that these differences are statistically significant for all five NASA datasets.

In these experiments, the duplicates are the "nice neighbors" described in the previous set of experiments. Performing a *ten*-fold cross validation insures that datasets with duplicates will have a distinct advantage over datasets without duplicates.

DISCUSSION

In general, project managers are reluctant to embrace empirical-based models in their decision-making process. Jørgensen (2004) estimates more than 80% of all effort estimation is human-based and only about 4% is machine learning-based. If empirical software engineering is going to have any hope of gaining favor with project managers, then it is critical that the modeling process be understood very well.

The first set of experiments shows the extreme range of answers in corresponding best case/worst case scenarios. These experiments clearly indicate how significantly nearest-neighbor sampling influences the results despite the fact that no dataset had any duplicates.

The second set of experiments reflects a more realistic situation where an empirical software engineer may (or may not) include duplicates in the modeling process. The difference in results is statistically significant. For some learners, the issue of duplicates is not a problem. However, as seen with the C4.5 learner, it is a problem.

In order to avoid building artificial models, perhaps the best approach would be to not allow duplicates within datasets. Another attribute could be added to the dataset, *Number of Duplicates*, so that information regarding duplicates is not lost.

This solves the issue of duplicates within datasets, however it does not address the issue regarding the synergy between testing and training datasets ("nice" versus "nasty" test sets). The next section addresses this issue.

BETTER CREDIBILITY THROUGH NEAREST NEIGHBOR-BASED SAMPLING

As demonstrated in the first set of experiments, it is evident that nearest-neighbor test data distribution dramatically impacts experimental results. The question is *How may nearest-neighbor sampling be incorporated into the project development process in order to generate realistic models?*

There are at least two possible solutions: one of which adapts to an organization's current software engineering processes; the second solution offers an alternative process.

In the first approach, a software engineer determines the nearest neighbor for each of the tuples in the test set (based on non-class attributes), relative to the training set. If the test tuple's nearest neighbor in the training set shares the same class instance value, then add *1* to a variable called *Matches*. *Matches* will be used to define a metric called *Experimental Difficulty* (*Exp_Difficulty*) as follows:

$$Exp\_Difficulty = 1 - Matches / Total\_Test\_Instances \quad (10)$$

The *Experimental Difficulty* provides a qualitative assessment of the ease/difficulty for modeling a given dataset. Combining this metric with an accuracy metric would offer a more realistic assessment of the results. For example, a "*Exp_Difficulty * Accuracy"* would give a more complete picture regarding the goodness of a model leading to better model selection and more credible models. For an *n*-fold cross validation,

the *Experimental Difficulty* could be calculated for each fold, then averaged over the *n* folds.

A second approach starts with the whole dataset prior to partitioning into training and test sets. For each tuple in the dataset, its nearest neighbor (with respect to the non-class attributes) is determined. Add 1 to the *Match* variable if a tuple's nearest neighbor is from the same class. Modifying Equation 10 results in the following equation:

$$Overall\_Difficulty = 1 - Matches / Total\_Data\_Instances \quad (11)$$

This gives an idea of the overall difficulty of the dataset. A software engineer may partition the data in order to increase (or decrease) *Experimental_Difficulty*. In the context of industrial-based benchmarks, the *Experimental_Difficulty* may be adjusted to coincide to a value adopted by another researcher. This lends greater credibility to comparing experimental results.

Considering an estimated 99% of the world's datasets are proprietary, this approach provides an additional benchmark for those models constructed on private datasets. Furthermore, these metric equations provide a means for assessing the robustness of the results.

CONCLUSIONS

Most datasets are proprietary in nature, making it impossible to replicate results in this situation. As demonstrated by the NASA experiments, not all data distributions result in similar results. This work extends previous research in defect prediction (Khoshgoftaar, 2003; Porter & Selby, 1990; Srinivasan & Fisher, 1995; Tian & Zelkowitz, 1995) by conducting nearest-neighbor analysis for gaining a deeper understanding of how datasets relate to each other, and thus the need for developing more realistic empirical/machine learning-based models. In the first set of NASA experiments, the "nice" dataset experiments (easy datasets to model) resulted in an average accuracy of 94.0% and the "nasty" dataset experiments (difficult datasets to model) produced an average accuracy of 19.5%. These results suggest that success in modeling a training dataset may be attributable to the ease/difficulty of the dataset, rather than the capability of the machine learner.

Including duplicates within a dataset reduces the difficulty of a dataset since similar tuples may appear in both the training and test sets. This research proposes removing duplicates in order to eliminate any bias.

Finally, this work proposes a set of metric equations for measuring the difficulty of a dataset. Benefits of using these metric equations include:

- **The creation of more realistic models.** These metrics will help the software engineering community better gauge the robustness of a model.
- **Greater credibility for models based on private datasets.** Since most datasets are proprietary, it is difficult to assess the quality of a model built in this context. Using the proposed metrics will make it easier to compare experiment results when the replication is impossible.
- **Greater chances of adoption by the industrial community.** As mentioned earlier, human-based estimation is still the method of choice. By providing a difficulty metric with a set of results, project managers will be able to assess the goodness of an empirical model. This will make it easier for a project manager to trust an empirical/machine learning-based model. Thus making it easier for the industrial community to more readily adopt empirical software engineering approaches.

FUTURE DIRECTIONS

This work could be extended from a two-class to an *n*-class problem. For example, the NASA datasets could be divided into four classes, (0, 1, 2, 3+ defects).

Another common type of software engineering dataset estimates programming effort. Thus, a likely future direction would examine these types of datasets.

Finally, it would be interesting to see whether accuracy results could be scaled by the "dataset difficulty metrics" in order to make better comparisons over datasets of varying difficulty.

REFERENCES

Albrecht, A. J. (1979). Measuring application development productivity. In the *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium.*

Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering, 9*(2).

Boehm, B. (1981). *Software engineering economics.* Englewood Cliffs, NJ : Prentice-Hall. ISBN 0-13-822122-7.

Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2000). *Software cost estimation with Cocomo II.* Pearson Publishing.

Boehm, B., & Basili, V. (2001). Software defect reduction top 10 list. *IEEE Computer, 34*(1), 135-137.

Dalcher, D., & Genus, A. (2003). Avoiding IS/IT implementation failure. *Technology Analysis and Strategic Management, 15*(4), 403-407.

Heemstra, F. J., & Kusters, R. J. (1991). Function point analysis: Evaluation of a software cost estimation model. *European Journal of Information Systems, 1*(4), 223-237.

Hihn, J., & Habib-Agahi, H. (1991). Cost estimation of software intensive projects: A survey of current practices. In *International Conference on Software Engineering* (pp. 276-287). Los Alamitos, CA: IEEE Computer Society Press.

Jones, C. (1998). *Estimating software costs.* McGraw Hill.

Jørgensen, M. (1997). An empirical evaluation of the MkII FPA estimation model. In *Norwegian Informatics Conference*, Voss, Norway, Tapir, Oslo (pp. 7-18).

Jørgensen, M. (2004) A review of studies on expert estimation of software development effort. *Journal of Systems and Software, 70*(1-2), 37-60.

KDD Nuggets Website. (2005). *Polls: Data mining tools you regularly use.* Knowledge Discovery and Data Mining Poll. Retrieved 2005, from http://www.kdnuggets.com/polls/data_mining_tools_2002_june2.htm

Khoshgoftaar, T. M. (Ed.). (2003). *Computational intelligence in software engineering, Annals of software engineering.* Kluwer Academic Publishers, ISBN 1-4020-7427-1.

Khoshgoftaar, T. M., & Allen, E. B. (2001). Model software quality with classification trees. In H. Pham (Ed.), *Recent advances in reliability and quality engineering* (pp. 247–270). World Scientific.

Kish, L. (1965). *Survey sampling.* New York: John Wiley and Sons, Inc.

Kitchenham, B., Pfleeger, S. L., McColl, B., & Eagan, S. (2002). A case study of maintenance estimation accuracy. To appear in the *Journal of Systems and Software.*

Menzies, T. (2005a). Personal conversation.

Menzies, T., Raffo, D., Setamanit, S., DiStefano, J., & Chapman, R. (2005b). Why mine repositories. Submitted to the *Transactions on Software Engineering.*

Paynter, J. (1996). Project estimation using screenflow engineering. In the *International Conference on Software Engineering: Education and Practice*, Dunedin, New Zealand (pp. 150-159). Los Alamitos, CA: IEEE Computer Society Press.

Pedrycz, W. (2002). Computational intelligence as an emerging paradigm of software engineering. In the *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy (pp. 7-14). ACM.

Porter, A. A., & Selby, R. W. (1990). Empirically guided software development using metric-based classification trees. *IEEE Software*, 46-54.

Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, 345-361.

Quinlan, J. R. (1992). *C4.5: Programs for machine learning.* San Francisco, CA: Morgan Kaufmann.

Rish, I. (2001). *An empirical study of the naive Bayes classifier, T. J. Watson Center.* In the IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence, Seattle.

Sauer, C., & Cuthbertson, C. (2003). *The state of IT project management in the UK*. Oxford: Templeton College.

Shepperd, M., Cartwright, M., & Kadoda, G. (2000). **On building prediction systems for software engineers. In** *Empirical Software Engineering* (pp. 175-182). Boston: Kluwer Academic Publishers.

Shirabad, J. S., & Menzies, T. J. (2005) *The PROMISE repository of software engineering databases school of information technology and engineering.* University of Ottawa, Canada. Retrieved 2005, from http://promise.site.uottawa.ca/SERepository

Sonnekus, R., & Labuschagne, L. (2003). *IT project management maturity versus project success in South Africa*. RAU Auckland Park, Johannesburg, South Africa: RAU Standard Bank Academy for Information Technology, ISBN: 0-86970-582-2.

Srinivasan, K., & Fisher, D. (1995). Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, 126-137.

The Standish Group. (2003). *The Chaos Chronicles III.*

Tassey, G. (2002). *The economic impacts of inadequate infrastructure for software testing* (Planning Report 02-3). Prepared by RTI for the National Institute of Standards and Technology (NIST). Retrieved 2005, from http://www.nist.gov/director/prog-ofc/report02-3.pdf

Tian, J., & Zelkowitz, M. V. (1995). Complexity measure evaluation and selection. *IEEE Transaction on Software Engineering*, *21*(8), 641-649.

Witten, I., & Franks, E. (2000). *Data mining: Practical machine learning tools with Java implementations.* San Francisco, CA: Morgan Kaufmann.

Zhang, D., & Tsai, J. J. P. (2005). *Machine learning applications in software engineering.* World Scientific Publishing, ISBN: 981-256-094-7.

ENDNOTE

[1] All monetary amounts are depicted in U.S. dollars.

This work was previously published in Advances in Machine Learning Applications in Software Engineering, edited by D. Zhang and J. Tsai, pp. 52-72, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.16
Morality and Pragmatism in Free Software and Open Source

Dave Yeats
Auburn University, USA

ABSTRACT

This chapter analyzes the differences between the philosophy of the Free Software Foundation (FSF) as described by Richard Stallman and the open source movement as described in the writings of Eric Raymond. It argues that free software bases its activity on the argument that sharing code is a moral obligation and open source bases its activity on a pragmatic argument that sharing code produces better software. By examining the differences between these two related software movements, this chapter enables readers to consider the implications of these differences and make more informed decisions about software use and involvement in various software development efforts.

INTRODUCTION

As governments around the world search for an alternative to Microsoft software, the open source operating system Linux finds itself in a perfect position to take market share from Microsoft Windows. Governments in France, Germany, The Netherlands, Italy, Spain, and the United Kingdom use Linux to encourage open standards, promote decentralized software development, provide improved security, and reduce software costs (Bloor, 2003). The Chinese government strongly supports Linux as its operating system of choice because Chinese experts have complete access to the source code and can examine it for security flaws (Andrews, 2003). In Brazil, leftist activists gathered to promote the use of open source software (OSS) (Clendenning, 2005).

There is a connection between the technological reasons for choosing open source software and the political ones. Many governments see open source as a way to promote a socialistic agenda in their choices of technology. Open source advocates, however, do not necessarily make these connections between the software development methods involved in open source and political movements of governments. There is evidence,

however, that leaders in the open source movement have expressed their rationale for advocating opening the source code of software.

The open source movement can trace its roots back to an alternate, still very active, software movement known as free software. While open source and free software can (and do) coexist in many ways, there are some essential differences that distinguish the two groups from one another. Perhaps most notably, the free software movement is based on a belief in a moral or ethical approach to software development, while open source takes a much more pragmatic view. While both groups argue for the open sharing of source code, each has its own reason for doing so. Understanding the differences between open source and free software can help open source researchers use more precise terminology and preserve the intent of each of these groups rather than assuming that they are interchangeable.

The following chapter begins with a brief historical overview of the free software and open source movements and highlights some of the main beliefs of each. The chapter then offers an examination of both the moral and pragmatic aspects of open source software. The conclusion invites readers to consider the implications of the differences between the two viewpoints and suggests ways for readers to apply this information when making choices about software.

BACKGROUND

The open source movement grew out of the software development practices in academic settings during the 1970s. During those early years of software development, computer scientists at colleges and universities worked on corporate-sponsored projects. The software developed for these projects was freely shared between universities, fostering an open, collaborative environment in which many developers were involved in creating, maintaining, and evaluating code (Raymond, 1999).

In his *A Brief History of Open Source* article, Charlie Lowe (2001) describes the end of open and collaborative methods of developing computer software in the 1980s when the corporate sponsors of academic software projects began to copyright the code developed for them. Corporations claimed that the university-run projects created valuable intellectual property that should be protected under law. This, of course, was just one of the signs of the shift from the commodity-based economy in the U.S. to a knowledge-based one. The wave of copyrights threatened to end the collaboration between computer scientists and slow the evolution of important projects. It looked as if the computer scientists would be required to work in smaller groups on proprietary projects.

Richard Stallman (1999) reports that he created the GNU General Public License (GPL) to maintain the ability to collaborate with other computer scientists on software projects, without restriction. The name GNU is a self-reflexive acronym meaning "GNU's Not UNIX," a play on words that pays homage to and differentiates itself from the UNIX legacy.[1] Stallman was concerned that the UNIX operating system, created during the collaborative era of the 1970s, would no longer be supported by new programs that used its stable and robust architecture when access to the source code was cut off. Stallman started the GNU initiative (which enabled the establishment of the Free Software Foundation [FSF]) to ensure that new software would be freely available.

The GNU GPL gave programmers the freedom to create new applications and license them to be freely distributable. Specifically, the GNU GPL gives anyone the right to modify, copy, and redistribute source code with one important restriction: Any new version or copy must also be published under the GNU GPL to insure that the improved code continues to be freely available. Many programmers (both those accustomed to the academic practices of the 1970s and new computer enthusiasts) adopted the GNU GPL and continued to work in open, collaborative systems.

Arguably the most important piece of software developed under the GNU GPL is the Linux operating system. Linus Torvalds, while still a student at the University of Helsinki in 1991, created a new operating system based on the ideas found in the UNIX operating system. This new piece of software, Linux, not only proved the success of GNU GPL, but it also represented a further shift toward a widely cooperative effort in software development. According to Eric Raymond, Linux debunked the myth that there were software projects with an inherent "critical complexity" that necessitated a "centralized, *a priori* approach" (Raymond, 2001, p. 21). With wide adoption among software developers and computer scientists, Linux proved to be a stable and powerful system despite its complexity and rapid development schedule.

In 1998, when Netscape decided to make its source code public as part of the Mozilla project, Eric Raymond and Bruce Perens suggested the use of the term "open source" in response to confusion over the term "free." Stallman, in many cases, found himself explaining that he was using the term "free" in the sense of "freedom" or "liberty" rather than "without monetary cost." Raymond and Perens founded the Open Source Initiative (OSI) to differentiate the new group from the FSF.

While there are many other active voices in the free software and open source movements such as Linus Torvalds (originator of the Linux operating system) and Robert Young (co-founder and CEO of Red Hat), Richard Stallman and Eric Raymond continue to be the most influential and widely cited. While Stallman and Raymond do agree at some level that software development benefits from the free distribution of source code, they see this free distribution in two completely different ways.

THE DEBATE

Many people have written about the debate between Eric Raymond and Richard Stallman. It is widely reported (Williams, 2002) that Stallman disagrees with Raymond's pragmatic reasons for promoting the term "open source" over "free software." In fact, Raymond's *Shut Up and Show Them the Code* and Stallman's *Why "Free Software" is Better Than "Open Source"* are two examples of the heated exchange between the two writers, each defending his own position on the issue of freely available source code. Bruce Perens reports that it is "popular to type-case the two as adversaries" (Perens, 1999, p. 174). While most studies emphasize that the term "open source" was adopted simply to avoid confusing the word "free" in "software" (DiBona, Ockman, & Stone, 1999; Feller & Fitzgerald, 2002; Fink, 2003), others are careful to point out that the shift in terminology really signaled a shift in strategy for open source advocates.

Perhaps the best work done on the differences between these groups is David M. Berry's (2004) work, *The Contestation of Code: A Preliminary Investigation into the Discourse of the Free/Libre and Open Source Movements.* By analyzing the discourse of the two movements (in the words of Stallman and Raymond), Berry concludes that the discourse of the free software movement more closely identifies with the user, is more utopian, and advocates a communal, socialist approach. The discourse of the open source movement, on the other hand, advocates a more individualistic approach that identifies with the "owners" or creators of software, resulting in a more libertarian emphasis.

In any case, the rift between those who choose to use the term free software and those who choose to use the term open source has resulted in some scholars choosing sides on the issue. Lawrence

Lessig (2004), an important scholar in the area of intellectual property law, discusses open source at great length in his work, *Free Culture*. However, he quotes only Stallman's writings, not Raymond's. To Lessig, at least, the open source movement is more about Stallman's rhetoric of freedom than Raymond's pragmatism. Understanding the rationale behind such choices is important in understanding the impact of open source software outside of the software industry.

OPEN SOURCE AND FREE SOFTWARE

In *The GNU Operating System and the Free Software Movement*, Stallman suggests that the two terms "describe the same category of software … but say different things about the software, and about values" (Stallman, 1999, p. 70). The following sections examine the work of Richard Stallman and Eric Raymond to investigate the different philosophical approaches to software development espoused by each.

Free Software: The Works of Richard Stallman

Stallman's (Stallman, 2002c) theorizing about software rests on what he identifies as the four main "freedoms" of his "Free Software Definition." According to Stallman (2002c, p. 18), these freedoms are:

- **Freedom 0:** The freedom to run the program, for any purpose
- **Freedom 1:** The freedom to study how the program works, and adapt it to your needs (access to the source code is a precondition to this)
- **Freedom 2:** The freedom to redistribute copies so you can help your neighbor
- **Freedom 3:** The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (access to the source code is a precondition to this)

In other words, Stallman directly relates his views about software development to a set of freedoms for users of that software. In the rhetoric of these main "freedoms," at least, Stallman is concerned more with the users of a software program than with the program itself.

In *The GNU Manifesto*, Richard Stallman makes impassioned arguments about his stance toward software development. "I consider that the golden rule requires that if I like a program I must share it with other people who like it" (Stallman, 2002a, p. 32), he writes. "So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free" (Stallman, 2002a, p. 32). He constructs his call for radical change in the way software development occurs with several ideological claims. Specifically, Stallman claims that sharing is fundamental and that free software offers the only ethical alternative for software programmers.

Stallman's rationale for calling programmers to work on an alternative to proprietary software is based on what he calls the "fundamental act of friendship of programmers": the "sharing of programs" (Stallman, 2002a, p. 33). Stallman suggests that "[m]any programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades" (Stallman, 2002a, p. 32-33). More than simply suggesting that the sharing of programs is ideal or simply important, Stallman argues that it is a fundamental imperative and a source of conflict. He goes so far as to suggest that programmers "must choose between friendship and obeying the law" (Stallman, 2002a, p. 33), implying that the law, on the issue of software availability, is in error.

The metaphors Stallman uses to expand on the idea of the centrality of sharing among developers makes it sound as if restricting software use is against nature itself. He writes: "Copying … is as natural to a programmer as breathing, and as productive. It ought to be as free" (Stallman, 2002a, p. 34). He goes on to equate software with air itself: "Once GNU is written, everyone will be able to obtain good system software free, just like air" (Stallman, 2002a, p. 34). Denying people the right to free software, in other words, would be like trying to regulate and restrict breathing itself. According to Stallman, restricting software use results in a kind of "police state" employing "cumbersome mechanisms" (Stallman, 2002a, p. 34) in the enforcement of copyright law.

While Stallman characterizes a software development community that shares all of its resources as a utopian society, he harshly criticizes proprietary software. He claims that restricting use of a program through intellectual property law constitutes "deliberate destruction" (Stallman, 2002a, p. 36) and a failure to be a good citizen. Stallman's rhetoric sets a scene with only two alternatives: free software based on the ideas of camaraderie, friendship, freedom, good citizenship, community spirit, and sharing of proprietary software based on the ideas of restriction, destruction, commercialization, and materialism. Clearly, Stallman's purpose is to set up a binary in which the only good is free software and the only evil is proprietary software. Any programmer who chooses to develop software in the capitalist proprietary software environment is choosing to be less moral than his or her free software counterparts.

Open Source Software: Raymond's Cathedral and Bazaar

Eric Raymond's *The Cathedral and the Bazaar* (2001), promotes open source software using two pragmatic claims that compliment each other: the promotion of the individual and the conscription of others. Unlike Stallman's emphasis on sharing and morality, Raymond emphasizes the practical aspects of open source that leads to its technical superiority. Specifically, Raymond describes the importance of the lead developers of projects while at the same time emphasizing the necessity of using others to complete work on projects. In neither case does Raymond express a belief in the moral superiority of open source development. Instead, all of the benefits of open source are described in terms of the development of a superior technological artifact.

Throughout *The Cathedral and the Bazaar*, Raymond (2001) promotes an egocentric view of technological development that emphasizes the role of the individual in the process. This egoistic approach is revealed in many ways—from Raymond's own self-congratulation, to his description of how developers find incentive to volunteer to participate in projects. Raymond's tendency to promote individuals over the group begins with his own tendency to describe himself as a gifted individual. While some of his claims may be true, it is unusual for a person to sing their own praises quite as blatantly as Raymond does. Usually, modesty does not allow for such open self-congratulation. In describing the personality traits common to good leaders for open source software projects, Raymond points to his own abilities. "It's no coincidence that I'm an energetic extrovert who enjoys working a crowd and has some of the delivery and instincts of a stand-up comic" (Raymond, 2001, p. 49). His infatuation with his own charming personality illustrates how much he values the individual over the group. More than once, Raymond cites instances where his superior programming skills enabled him to make extraordinarily wise decisions that a lesser programmer might miss. For his programming and writing skills, Raymond mentions that he got "fan mail" (Raymond, 2001, p. 38) and "help[ed] make history" (Raymond, 2001, p. 61). Clearly, Raymond's focus on the individual begins with himself.

Raymond goes beyond his own egoism, however, when he generalizes about what constitutes a good open source software project. According to Raymond, "every good software project starts by scratching a developer's personal itch" (Raymond, 2001, p. 23). This aphorism is the first of the 19 rules of open source software development. It is interesting that Raymond recognizes that the motivation behind good software comes not from a need in the community but rather from a personal interest or desire. Raymond reiterates this emphasis on the individual developer's primacy in starting a project in tenet 18: "To solve an interesting problem, start by finding a problem that is interesting to you."(Raymond, 2001, p. 49) Again, nowhere in Raymond's writing does he refer to moral behavior or developers and a need to share. Instead, he believes that the curiosity of an individual developer is enough to justify work in the open source model.

The natural conclusion to a system that encourages individuals to involve themselves in only those projects which they find personally interesting is a hierarchical system that promotes these individuals. Open source developers who choose to take on a particular software problem promote themselves to the role that Raymond calls the "core developer" (Raymond, 2001, p. 34). This role bestows the leadership upon a single individual who is, in turn, supported by a "halo of beta-testers" who exist to serve the needs of the leader (Raymond, 2001, p. 34). Naturally, this leader wields considerable power over his or her user community. And, according to Raymond, not every developer possesses the skills to be a good project leader. Raymond presupposes that any good project leader has superior technical abilities that are generally recognized in the open source community. Further, he suggests that the core developers have skills "not normally associated with software development"—people skills (Raymond, 2001, p. 48).

What Raymond calls people skills is actually an ability to provide incentive to other developers to enlist their help with a project. Raymond posits that the success of the Linux project came largely from core developer Linus Torvalds' ability to keep his volunteer developers "stimulated and rewarded" by giving them "an ego-satisfying piece of the action" (Raymond, 2001, p. 30). In his own fetchmail project, Raymond says he made a habit of "stroking [users] whenever they sent in patches and feedback" (Raymond, 2001, p. 38). In his analysis of how to encourage participation in members of the open source community, Raymond asserts that hackers find rewards in "the intangible of their own ego satisfaction and reputation among other hackers" (Raymond, 2001, p. 53). A project leader must "connect the selfishness of individual hackers as firmly as possible to the difficult ends" involved in software development (Raymond, 2001, p. 53). Rather than provide monetary incentive, then, Raymond encourages an approach that enables project leaders to conscript users' assistance through a coercive appeal to their egoistic, selfish desire for glory. This approach simultaneously reinforces the leader's domination over other developers and de-emphasizes any development practice based on goals related to benefit to the community.

Raymond discusses how the paradigm of encouraging egoistic behavior of volunteer developers affects the individual reputation in the leader in the following passage:

Interestingly enough, you will quickly find that if you are completely and self-deprecatingly truthful about how much you owe other people, the world at large will treat you as though you did every bit of the invention yourself and are just being becomingly modest about your innate genius. (Raymond, 2001, p. 40)

It is difficult to believe that Raymond would ever be mistaken as "becomingly modest." Even when he encourages leaders to give credit to those that assist with the project, he reveals the underly-

ing motive of additional glory and recognition in the open source community.

The dominating force that goes hand-in-hand with Raymond's suggestion that project leaders should appeal to a volunteers' selfishness is the idea that these users must be recruited and conscripted in order to create a successful open source project. Raymond quotes Linus Torvalds as saying, "I'm basically a very lazy person who likes to take credit for things that other people actually do" (Raymond, 2001, p. 27). While Torvalds is obviously speaking tongue-in-cheek here, it reveals a common theme that Raymond continues to espouse. Two of the 19 development practices include, "If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource" (Raymond, 2001, p. 38), and "The next best thing to having good ideas is recognizing good ideas from your users" (Raymond, 2001, p. 40). Both of these statements imply that the volunteer developers belong to and work for the project leader. In addition, the project leader can use these volunteers for his or her own purposes like a natural resource.

The idea of individual ownership extends beyond the volunteers on a particular project to the software itself. Despite the fact that projects are "co-developed" by many individual developers, the lead project coordinator actually "owns" the technology. This idea is present in another one of Raymond's main tenets: "When you lose interest in a program, your last duty to it is to hand it off to a competent successor" (Raymond, 2001, p. 26). Therefore, the technology can be bequeathed and inherited much like a traditional patriarchal succession of ownership. And when a software project is passed down to the next generation of leadership, the volunteer user base comes with it.

Speaking of this volunteer user base, Raymond suggests that "[p]roperly cultivated, they can become co-developers" (Raymond, 2001, p. 26). In addition to cultivating, Raymond suggests that users can be "harnessed" (Raymond, 2001, p. 50) to do work for the lead developer. Essentially, Raymond espouses conscripting volunteers to do the work of the lead developer. Tenet 6 summarizes his position: "Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging" (Raymond, 2001, p. 27). The implication of that statement is not that users *really are* co-developers but rather that users should be treated *as if they were* co-developers in order to ensure that they will do work for the improvement of the system. Raymond seems to believe that core developers could build open source software projects on their own, but enlisting the help of users provides a less difficult way to achieve the goal of creating a powerful system. Conspicuously absent in this method of project management is the idea that these volunteer users are better served by participating in the development process. Instead, Raymond's main concern is with the system itself.

According to Raymond, the true benefit of this conscription model of development comes from the advantages of using a large body of volunteers to detect and fix bugs in the system. Tenet 7 is, "Release early. Release often. And listen to your customers" (Raymond, 2001, p. 29). However, Raymond's description of the value of this rule does not include a plea for technology that is sensitive to users' needs. Instead, he asserts that this innovation is simply an effective way to test the software for technological bugs, not usability problems. The goal is to "maximize the number of person-hours thrown at debugging and development, even at the possible cost of instability in the code" (Raymond, 2001, p. 30). Raymond suggests that a "program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented" (p. 47). Therefore, he promotes systems exposed to "a thousand eager co-developers pounding on every single new release. Accordingly you release often in order to get more corrections, and as a beneficial side effect you have less to lose if an

occasional botch gets out the door" (Raymond, 2001, p. 31).

His suggestion that less-than-usable software can be released shows that his interest is not in the value of the software to users. His interest is in the value of the users to the software.

MORALITY AND PRAGMATISM

While Stallman's emphasis in advocating for the free software movement is clearly one of moral behavior and obligation, Raymond's characterization of the open source movement emphasizes the technological superiority of a decentralized development process. Stallman's argument sets up free software as a superior and more ethical alternative to proprietary software development that focuses on the rights and freedoms of users and developers. Nowhere in Raymond's writings does he suggest that proprietary software is less ethical. That is not to say Raymond isn't critical of proprietary software. However, his main concern is always the technological implications of software rather than the moral.

Table 1 outlines a few of the more important differences between the two movements:

Faced with two very different value systems surrounding these related movements, open source software users should pay careful attention to the software they choose to use or the software communities in which they participate. While the two approaches to software development adopt similar practices, they represent two different viewpoints that are often at odds with one another. If a user is making a choice to use open source software because of a belief that it is more moral to support open intellectual property policies, they may want to seek out like-minded projects that use the Stallman approach. If a user is more concerned about the technological superiority of open source software even if that superiority comes at the cost of an emphasis on equality among users, then they may want to seek out projects that are run by maintainers that use the Raymond style.

In either case, users should be aware that the choices they make in their affiliations also signal to others that they adopt the worldview represented in those choices. While there may be many instances of individual developers and software projects that blend the ideas and beliefs of both Raymond and Stallman, it is still important to understand that these philosophies often result in development approaches at odds with each other. Though the practices are admittedly similar, a difference in *why* one would choose to develop open source software can affect *how* one carries out that choice.

Perhaps the most important thing to realize, however, is that neither the Raymond nor the Stallman approach is inherently superior for all users. Instead, the choice to adopt one of the approaches over the other rests entirely upon the needs and situation of the individual user. While the rhetoric of both Stallman and Raymond suggest that their understanding of software development represents a truly enlightened and superior approach, neither one can be said to offer the final word.

FUTURE TRENDS

The most inclusive and technically accurate description of software with freely available source code is free/libre open source software (F/LOSS or FLOSS) because it accurately maintains portions of each of the various movements in the software community. However, the term *open source* has proven to be the most popular, partly because of the deliberate attempt by open source advocates to make the licensing structures more business-friendly. Apart from its popularity, many choose the term open source almost exclusively due to its influence on the broader culture; open source has been used as a descriptor for everything from yoga to t-shirt designs.[2] When these non-technological instances of open source are used, the suggestion is that the development of these particular creative

Table 1. The morality and pragmatism of the free software and open source movements

Morality ***Free Software/Stallman***	**Pragmatism** ***Open Source/Raymond***
Defines the benefit of free software as a superior moral choice.	Defines the benefit of open source as a pragmatic way to develop superior software.
Emphasizes developers' moral obligation to share with others.	Emphasizes satisfying developers' personal and individual desires.
Understands the development process as a shared, communal, group effort based on socialistic principles.	Understands the development process as one driven by one or a small group of leaders who conscript volunteers to assist with the project.

endeavors is open to many. More than FLOSS, open source represents both a software development phenomenon and a cultural one.

However, when popular culture adopts open source to mean an open, sharing community of creative invention, it misses the main emphasis of the movement. According to Raymond, the movement is less about the moral imperative to share with others and more about the benefit of harnessing the creative energy of individuals who are free to choose their own work. Rather than seeing open source as good for the public, Raymond emphasizes the benefit of the process for the technology. In other words, Raymond is more interested in product than people.

Unfortunately, it is likely too late to correct the trend in popular culture to equate the term open source with "sharing intellectual property" even though open source refers to a process and value system much more complicated than simply sharing. While the term open source government" certainly carries with it a grandiose image of participatory, shared government in which each member of a community has a voice, but it is unclear how open source government is different from a healthy democracy.

Members of the open source community can contribute to the increased use of the term open source by helping others understand where new uses of the term open source resonate with similarities in the software movement and where they miss the mark. Very few creative endeavors have anything akin to the source code found in software development, so making the source code freely available, the essential meaning of the phrase open source, cannot be replicated in other fields. However, the idea that creative work should be shared and that sharing can be protected with creative licensing is a contribution from the open source movement that can be adopted by others. Rather than adopting open source, other communities may benefit by using more precise and applicable language such as Creative Commons to refer to the sharing of intellectual property.

CONCLUSION

While many researchers and developers of open source software (myself included) typically lump free software and open source software together, both Stallman and Raymond adamantly insist that there are fundamental differences between the two groups. In his essay *Why "Free Software" is Better than "Open Source,"* Stallman explains that the two groups "disagree on the basic principles, but agree more or less on the practical recommendations" (Stallman, 2002b, p. 55). In other words, free software and open source software are essentially the same in *practice*, but not in *principle.*

Because open source and free software appear to operate the same way to outside observers, there is very little insight into when a piece of software

should be labeled open source and when it should be labeled free software. Often, both use the same licensing structures. The essential difference is not a technological one; it is one of philosophies. Only the developers themselves can attest to the reasons they choose to develop software with freely available source code. However, it is useful for outside observers to be more precise in their allegiances. It could mean the difference between freedom and pragmatism.

REFERENCES

Andrews, P. (2003, November 24). Courting China. *U. S. News and World Report*, p. 44-45.

Berry, D. M. (2004). The contestation of code: A preliminary investigation into the discourse of the free/libre and open source movements. *Critical Discourse Studies, 1*, 65-89.

Bloor, R. (2003, June 16). Linux in Europe. *IT-Director.com*. Retrieved February 7, 2005, from http://www.it-director.com/article.php?articleid=10929

Clendenning, A. (2005, January 30). Activists urge free open-source software. *World Social Forum*. Retrieved February 7, 2005, from http://www.commondreams.org/headlines05/0130-03.htm

DiBona, C., Ockman, S., & Stone, M. (1999). Introduction. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 1-17). Sebastopol, CA: O'Reilly & Associates.

Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development*. London: Addison-Wesley.

Fink, M. (2003). *The business and economics of Linux and open source*. Upper Saddle River, NJ: Prentice Hall PTR.

Lessig, L. (2004). *Free culture: How big media uses technology and the law to lock down culture and control creativity*. New York: Penguin Press.

Lowe, C. (2001). A brief history of open source: Working to make knowledge free. *Kairos: A Journal for Teachers of Writing and Webbed Environments, 6*(2). Retrieved October 25, 2005, from http://english.ttu.edu/KAIROS/6.2/news/opensource.htm

Luman, S. (2005, June). Open source softwear. *Wired 13.06*.

Pallatto, J. (2005, May 13). Yoga suit settlement beggars open source ideals. *eWeek*. Retrieved October 25, 2005, from http://www.eweek.com/article2/0,1759,181,5971,00.asp

Perens, B. (1999). The open source definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 171-188). Sebastopol, CA: O'Reilly & Associates.

Raymond, E. (1999). A brief history of hackerdom. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 19-30). Sebastopol, CA: O'Reilly & Associates.

Raymond, E. (2001). The cathedral and the bazaar. *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary* (rev. ed., pp. 19-64). Sebastopol, CA: O'Reilly & Associates.

Stallman, R. (1999). The GNU operating system and the free software movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 53-70). Sebastopol, CA: O'Reilly & Associates.

Stallman, R. (2002a). The GNU Manifesto. In J. Gay (Ed.), *Free software free society: Selected*

essays of Richard M. Stallman (pp. 31-39). Boston: Free Software Foundation.

Stallman, R. (2002b). Why "free software" is better than "open source." In J. Gay (Ed.), *Free software free society: Selected essays of Richard M. Stallman* (pp. 55-60). Boston: Free Software Foundation.

Stallman, R. (2002c). The GNU Project. In J. Gay (Ed.), *Free software free society: Selected essays of Richard M. Stallman* (pp. 15-30). Boston: Free Software Foundation.

Williams, S. (2002). *Free as in freedom: Richard Stallman's crusade for free software*. Sebastopol, CA: O'Reilly & Associates.

KEY TERMS

Free/Libre and Open Source Software (FLOSS): A more inclusive term for all software with freely available source code.

Free Software (FS): Software with freely available source code developed in the tradition of the Free Software Foundation and influenced by the writings of Richard Stallman.

Morality: An appeal to the fundamental goodness of an act; primary rationale behind the free software movement.

Open Source Software (OSS): Software with freely available source code developed in the tradition of the Open Source Initiative (OSI) and influenced by the ideas of Eric Raymond and Bruce Perens.

Proprietary Software (PS): Software without publicly available source code, commonly seen as the opposite of free and open source software.

Pragmatism: An appeal to the usefulness of an act; primary rationale behind the open source movement.

ENDNOTES

1 It is common for developers to use reflexive acronyms, partly as a tongue-in-cheek recognition of the overuse of acronyms in technology. Other examples include PHP (PHP hypertext protocol) and WINE (WINE Is Not an Emulator).

2 For more information about some of the ways open source is being used outside of software, see Stuart Luman's article "Open Source Softwear" (2005, *Wired 13*[06]) and John Pallatto's article "Yoga Suit Settlement Beggars Open Source Ideals" (2005, *eWeek*, May 13).

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant and B. Still, pp. 23-33, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.17
A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications

W. K. Chan
Hong Kong University of Science and Technology, Hong Kong

S. C. Cheung
Hong Kong University of Science and Technology, Hong Kong

Karl R. P. H. Leung
Hong Kong Institute of Vocational Education, Hong Kong

ABSTRACT

Testing the correctness of services assures the functional quality of service-oriented applications. A service-oriented application may bind dynamically to its supportive services. For the same service interface, the supportive services may behave differently. A service may also need to realize a business strategy, like best pricing, relative to the behavior of its counterparts and the dynamic market situations. Many existing works ignore these issues to address the problem of identifying failures from test results. This article proposes a metamorphic approach for online services testing. The off-line testing determines a set of successful test cases to construct their corresponding follow-up test cases for the online testing. These test cases will be executed by metamorphic services that encapsulate the services under test as well as the implementations of metamorphic relations. Thus, any failure revealed by the metamorphic testing approach

will be due to the failures in the online testing mode. An experiment is included.

INTRODUCTION

The service-oriented architecture (SOA) is an architectural reference model (Bass et al., 2003) for a kind of distributed computing such as the Web services (W3C, 2002). It promises to alleviate the problems related to the integration of heterogeneous applications (Kreger et al., 2003; Mukhi et al., 2004). In this reference model, a SOA application is denoted by a collection of self-contained communicating components, known as *services*. The model also emphasizes that each service should make little or no assumption about its collaborating services. This setting advocates the dynamic composition of a service by using different configurations of supportive services, creating behavioral differences amongst different invocations of a service. Typical end-users of a SOA application, such as bank customers using an online foreign exchange trading service, may expect consistent outcomes each time they use the service. The customers may further compare the online foreign exchange service of a bank to similar services of other banks to judge whether the service is of good quality. If a B2B service provider is driven by a predefined business strategy to, for example, maintain its market share, then the criteria to define functional correctness of the service may vary according to its environment. In other words, testers need to integrate the environment of a service to check test results.

Services may be subject to both the off-line testing and the online testing. Unlike the testing of conventional programs, services bind dynamically to other peer services when it is tested online. While the off-line testing of services is analogous to the testing of conventional programs, the online testing of services needs to address new issues and difficulties.

Testers may generally apply certain static analyses or testing techniques to assure the correctness of a software application. The evaluation criteria of the functional correctness, on the other hand, must be predefined. For a unit testing, testers also want to address the test case selection problem and the test oracle problem (Beizer, 1990). We restrict our attention to the latter problem in this article.

A test oracle is a mechanism that reliably decides whether a test succeeds. For services, as we will discuss, formal test oracle may be unavailable. The expected behavior of a service that represents business goods and services changes according to the environment. Such an expected behavior is relative to the behaviors of competing services or other services. Intuitively, it is hard to define the expected behavior explicitly in the first place. Tsai et al., (2004) for example, suggest using a progressive ranking of similar implementations of a service description to alleviate the test oracle problem. The behaviors of different implementations of the same service vary in general. Test results of a particular group of implementations cannot reliably be served as the expected behavior of a particular implementation of the same service on the same test case. Also, a typical SOA application may comprise collaborative services of multiple organizations, knowing all the implementations are impractical (Ye et al., 2006). For example, a travel package consultant may wrap up services of various hotels, airlines, and entertainment centers to personalize tour packages for a particular client. Without the implementation details, static analysis appears infeasible to assure the implementation quality. The black box approach for test results checking remains viable and popular to assure the correctness of a software application.

Metamorphic testing is a promising approach to the test oracle problem in the testing of conventional scientific programs (Chan et al., 1998; Chen et al., 1998, 2002). Instead of relating an

output to its input, metamorphic testing relates multiple input-output pairs via well-defined relations, called metamorphic relations.

This article extends the preliminary version (Chan et al., 2005a) to propose an online testing approach for testing services. The main contributions of the preliminary version (Chan et al., 2005a) include:

1. It proposes to apply the notion of metamorphic testing to services computing to alleviate the test oracle problem. It constructs more test cases to reveal faults than those ordinarily required when test oracles are known.
2. It proposes to realize the metamorphic testing mechanism as a metamorphic service in services computing that encapsulates a service under test, executes test cases and cross-validates their test results. Such realization seamlessly integrates the existing SOA framework. It automates the construction of follow-up test cases and their test results checking.

The main extended contributions of this article include:

3. It devises the use of the successful test cases for off-line testing as the original test cases for online testing. Since, in an off-line testing, the environment of services can be controlled by testers, test oracle could be defined. Thus, any failure revealed by our metamorphic testing approach will be due to the failures in the online testing mode.
4. It refines the testing approach presented in the preliminary version of this article to take the successful test cases from the off-line testing phase into account.
5. It evaluates the revised proposal against a control experiment having no prior confirmation of test results through an experiment. The experimental result indicates that our proposal is superior to the control experiment. The control approach suffers from an extra 16% effort to check test results and a 13% reduction of failure detection.

The rest of the article is organized as follows. The next section introduces the concepts of services, metamorphic testing and other preliminaries. After that, we describe our testing proposal. Our approach will be illustrated by a sample scenario, followed by an experiment on the feasibility of applying the proposal in a SOA application. Based on the experimental results, we compare our approach to related work, and discuss our experience. Finally, we conclude the article and outline the future work.

PRELIMINARIES

In this section, we briefly introduce the notion of service, metamorphic relation, and metamorphic testing. We also discuss how a metamorphic relation can be aligned with the notion of service and discuss our assumptions. Finally, we clarify our terminologies on online testing and off-line testing.

Service

A *service* in SOA is a self-contained application function, packaged as a reusable component, for the use in a business process (Colan, 2004). It can describe itself so that other services may discover and locate the service. Services also use simple and self-contained messages with well-defined interfaces for communications. Those interfaces are neutral to different hardware or software platforms to support the execution of a service. Meta-models, such as XML Schema (W3C, 2001), are often used to govern the syntactic validity of the messages.

For example, the Web services (Curbera et al., 2002; W3C, 2002) use the *Web service defini-*

tion language or WSDL (W3C, 2005) to let each service of an application define the syntactic and interoperability requirements. It also uses the *universal description, discovery and integration* or UDDI (OASIS, 2005) to offer applications a unified and systematic way to locate services from common registries. The *simple object access protocol* or SOAP (W3C, 2003) is then used to support XML[2]-based messaging between peer services.

In general, there are two types of services, namely *stateless* and *stateful*. They are analogues to conventional programs and object-oriented programs, respectively. Both types are popular in these days.[3] Intuitively, in terms of modeling, one may use a stateless service with self-addressing stateful messages to simulate a stateful service. We thus restrict ourselves to consider stateless services in this article.

Metamorphic Relation (MR)

As a service is a self-contained and function-oriented component, some functional properties of the service should be identifiable; otherwise, it would be difficult for collaborating services to identify the usefulness of the service. One way to display the functional properties of a service is to show the connection of multiple invocations of the service.

A metamorphic relation, or MR, is an existing or expected relation over a set of distinct inputs and their corresponding outputs for multiple executions of the target function (Chen et al., 1998, 2002, 2003). For example, consider a sorting program *Sort*( ) that sorts the input list of integers into a list of integers in ascending order. The expected result of *Sort*(⟨*1,4,3*⟩) is ⟨*1,3,4*⟩, which is same as that of *Sort*(⟨*1,3,4*⟩). Generalizing the scenario would lead to the formulation of the relation: *Sort*(*X*) = *Sort*(*Sort*(*Y*)) *if X = Y.*

Readers can easily observe that in the above example, the expected output of the input ⟨*1,4,3*⟩ is only induced implicitly. This relieves testers to predetermine the expected behavior of a test case in an absolute term. It is particularly important when a tester is conducting the online testing of a service, for the reason that the concrete expected behavior of a service may depend on other peer services, which may not be under the control of the software developers or testers of the service under test.

A metamorphic relation can be formally described as follows:

> Suppose that *f* is an implemented function under test. Given a relation *r* over *n* distinct inputs $x_1, x_2, \ldots, x_n$, the corresponding *n* computation outputs $f(x_1)$, $f(x_2)$, …, $f(x_n)$ must induce a necessary property r_f.

A metamorphic relation MR_f of *f* over *n* inputs and *n* outputs can be defined as follows:

$$MR_f = \{(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n)) \mid r(x_1, x_2, \ldots, x_n) \Rightarrow r_f(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n))\}$$

We observe that, when a service is seen to be a function, and all input and output messages are self-contained, the notion of metamorphic relation readily applies to services. We will give an example after the introduction of metamorphic testing in the following section.

Metamorphic Testing (MT)

Metamorphic testing or MT (Chen et al., 1998, 2002, 2004) is a program testing technique that employs the mathematical relations, namely metamorphic relations, to conduct testing. It has been applied to numerical applications (Chan et al., 1998) and context-aware applications (Chan et al., 2005c).

It uses *successful test cases* to alleviate the test oracle problem when a test oracle is unavailable or expensive. Given a program *P* of target function *f* with input domain *D.* A set of test cases *T,*

$\{t_1,\ldots,t_k\}$ ($\subset D$), can be selected according to any test case selection strategy. Executing the program P on T produces outputs $P(t_1), \ldots, P(t_k)$. When they reveal any failure, testing stops and debugging begins. On the other hand, when no failure is revealed, these test cases will be termed as *successful*. A set of successful test cases is called a successful test set.

Testers may apply the metamorphic testing approach to continue to verify whenever some necessary property of the target function f is satisfied by the implementation P. The metamorphic testing approach constructs the *follow-up* test set T', $\{t'_1,\ldots,t'_m\}$ ($\subset D$), automatically from the initial successful test set T, with the reference to some given metamorphic relation.

Let us cast an example in the services computing setting. Suppose that S is an expected USD−HKD exchange service, accepting deal orders in USD and returning deal orders in HKD; x and y are two deal orders; $g(\ )$ is a function that accepts a deal order and returns its deal order amount. Further suppose that the following metamorphic relation sample is given, meaning that doubling the deal order in USD, doubling the resultant amount in HKD:

$MR_a(x, y, S(x), S(y))$:
$2g(S(y)) = g(S(x))$ if $g(x) = 2g(y)$

Consider a successful test case x_1 = "a deal order of US$20,000." The metamorphic testing approach constructs automatically another test case y_1 = "a deal order of US$10,000" based on the condition $g(x) = 2g(y)$ of MR_a. Suppose P is an implementation of S. If the comparison $2g(P(y_1)) = g(P(x_1))$ fails, the MT approach reveals a failure due to the pair of failure-causing inputs x_1 and y_1. As x_1 is successful in the first place, the identified failure should be due to the test case y_1.

In the MT terminology, x_1 is termed as the *original test case*, and y_1 is termed as the *follow-up test case*. By checking the results amongst multiple input-output pairs, metamorphic testing bypasses the need to define the expected result explicitly to alleviate the test oracle problem. We refer readers to (Chan et al., 2005c; Chen et al, 2002, 2003, 2004; Tse et al., 2004) for more details about metamorphic testing.

In this article, we assume that MRs are provided by testers. Verification is generally undecidable, thus, we further assume that the provided MRs describe some necessary conditions of the expected behavior of the program under test. In practice, we recommend testers to define their MRs in a style so that they are directly coded in certain executable specification languages such as Prolog. As such, the implementation of MRs is a non-issue when testers use our methodology. In the next section, we further describe a few major assumptions of our proposal.

Assumptions and Terminologies

In this section, we list out a few major assumptions to establish our model for the online testing of services. These assumptions facilitate us to propose an infrastructure, namely metamorphic service, to be discussed in the metamorphic service section. We also clarify our terminologies on off-line testing and online testing.

We make the following assumptions: A service could be wrapped up by another (wrapper) service. It is based on the understanding that services are loosely coupled amongst themselves and they recognize one another through the common service registries. We assume that in the common service registries, the entries of the former service are replaced by those of the latter service for the testing purpose. This allows the latter service to query and catch messages for the former one. In our model, we use a message as a *test case* or *test result* of a service. This kind of treatment is also adapted by other researchers (Tsai et al., 2004; Offutt & Xu, 2004). It enables the wrapper service to construct test cases and evaluate their test results.

Furthermore, we agree with other researchers that a service is self-contained and independent to the contexts of other services. It also outputs results on every input.

We refer the term *online testing* to the testing of a service under the SOA environment, and the term *off-line testing* to the testing of a service without interaction with other services relevant to the service under test. We also term *off-line testing* and *testing in the off-line mode* interchangeably, and *online testing* and *testing in the online mode* interchangeably. Moreover, for the off-line testing, we refer a *stub service* as a service, since it is obvious in the context that the service under test does not interact with any other peer services of the SOA application when it is subject to an off-line testing. In the next section, we will present our testing proposal.

AN ONLINE TESTING APPROACH

In this section, an online testing methodology will be presented. We propose to test a service in two distinct modes, namely off-line mode and online mode. Informally, we propose to use the test oracle available to the off-line testing and make it also available to the online testing via metamorphic relations. In the overview section, we first introduce the two modes of testing. Next, in the metamorphic service section, we propose a core design artifact of our approach, the metamorphic service, which serves as a surrogate of the services under test to relay messages. At the same time, it constructs follow-up test cases and evaluates results accordingly. It then summarizes the methodology in the testing in the online mode section.

Overview

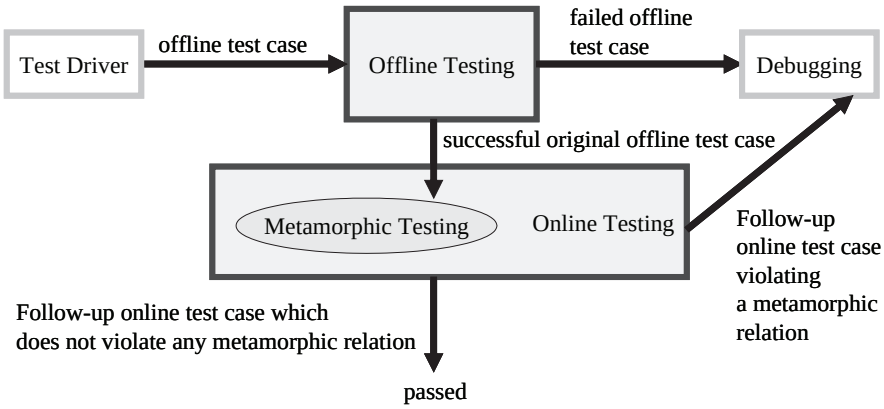
The off-line mode tests a service in the absence of interacting services. We view that it strongly resembles the convention techniques for the unit test of a conventional program. Many test data selection strategies (such as dataflow testing (Beizer, 1990; Zhu et al., 1997)) and test oracle construction for the unit test of conventional programs have been researched for years. As testers could control the off-line environment (such as providing test drivers and stub services, and messages), it is relatively easier to define test sets to test a service and validate the test results in such a mode. Thus, whether a test case reveals a failure or not, for off-line testing, can be determined through the checking of their test results against certain test oracle.

The above observation motivates us to apply the successful test cases of an off-line testing as the original test cases for the online testing mode. In the sequel, we restrict ourselves to discuss the online testing mode. We presume that a set of successful test cases for the off-line testing is available.

For online testing, as these original test cases are determined to be successful in advance, any violation of a corresponding metamorphic relation should be due to the failures of the follow-up test cases. Thus, when their corresponding follow-up test cases are for online testing, our approach pinpoints the online failure-causing inputs automatically even if the test oracle for online testing is not available. Figure 1 depicts the relation between off-line testing and our proposal for online testing.

Interested readers may compare the above strategy and the strategy that an original test case is unknown to be successful. In the latter strategy, when a violation of a metamorphic relation is detected, testers still need to further evaluate whether the violation is due to the original test case (no matter if it is in the off-line or online testing mode) or the follow-up (online) test case. Therefore, this alternative strategy may incur more overheads than our proposal. We will further study this issue in our empirical experiment in the experiment section.

Figure 1. The Integration between off-line and online testing



We will introduce a fundamental building box of our approach in the metamporphic services section, and then elaborate our online testing approach in the rest of this section. To ease our discussion, in the sequel, we use a metamorphic relation with two input-output pairs. The extension of the approach to more than one input-output pair is not hard.

Metamorphic Service (MS)

We formulate a metamorphic service as a service that has the characteristics of being an access wrapper (Mecella & Pernici, 2001; Umar, 1997) and being an agent to conduct metamorphic testing. A metamorphic service imitates all the data and application access paths of the service being encapsulated. It also embodies the program logics, which is the implementation of metamorphic relations, to compute follow-up test cases based on an incoming (or outgoing) message, and evaluates test results according to the implemented metamorphic relations.[4]

Since a metamorphic service is a service, which dynamically discovers a service implementation to receive a follow-up test case. The result of the follow-up test case will be sent to the metamorphic service for the detection of failures. The metamorphic service checks the test results against other test cases by using its metamorphic relations. Any violation of an implemented metamorphic relation reveals a failure.

Testing in the Online Mode

The testing in the online mode validates whether a service can interact correctly with other services. The following shows the self-explanatory methodological steps for this mode.

(ON-1). For a service under test S, collect the set of service descriptions D_S that represents the services interacting with S.
(ON-2). Design a metamorphic relations MR_i applicable to test S in the online mode.
(ON-3). Implement MR_i in the metamorphic service MS of the service S.
(ON-4). Repeat Steps (ON-2) to (ON-3) until no additional metamorphic relation is required for online testing.
(ON-5). For each available successful off-line test case t_o, do
  i. MS uses applicable MR_i to construct the following-up test case t_f of t_o.
  ii. MS invokes S to execute t_f.
  iii. MS obtains the corresponding results t_f

iv. If *MS* detect a failure by using MR_i, then report the failure and go to Step (ON-7).
v. Repeat Steps (On-5-i) to (On-5-iv) until no more applicable MR_i.

(ON-6). Report that no failure is found.
(ON-7). Exit

Step (ON-1) collects the service description that the service under test intends to collaborate. They facilitate testers to define and implement relevant metamorphic relations[5] in Steps (ON-2) and (ON-3) respectively.

If a tester does not identify any further metamorphic relation, Step ON-4 naturally stops, because there is no additional metamorphic relation in the tester's context. On the other hand, if a tester knows a metamorphic relation, but is incompetent to implement such a relation, it is obvious that the tester cannot use such a metamorphic relation according to our methodology; unless the tester seeks helps to implement the metamorphic relation.

Step (ON-5) uses an original test case to construct a follow-up test case. It also invokes the service under test to execute the follow-up test case, and collects the test result of the follow-up test case. Next, it evaluates the results until no more implemented and applicable metamorphic relation is available. When it detects a violation of any metamorphic relation, in Step (On-5-iv), it reports the failure and stops the testing. Otherwise, the process iterates, using another original test case until no more original test case is available. If no failure could be detected by any test case, the process will report such a case in Step (ON-6) and stop at Step (ON-7).

After presenting our approach in this section, the next section will demonstrate a way to use metamorphic testing to reveal failures related to the relative correctness for the testing of a service. We will then present an empirical experiment of our proposal in the experiment section.

AN ILLUSTRATION SCENARIO

In this section, our proposal for online testing will be further illustrated by an example. We first describe an application and its faults. Then, we illustrate how these faults can be revealed.

Figure 2 shows a foreign exchange dealing service application with five services, namely FXDS1 to FXDS5. In particular, FXDS2 is embraced by a metamorphic service. We denote the metamorphic service as *MS*. It has three metamorphic relations, namely MR1, MR2 and MR3. To ease our discussion, we restrict ourselves to discuss the exchange of US dollars to Renminbi.

A bank normally offers cross-currency rates inquiry services. A sample USD−RMB exchange rate is a pair of values such as 8.2796/8.2797. The first value and the second value in the pair refer to the *bid rate* and the *ask rate*, respectively. The difference between the two values in such a pair is known as the spread. We will use this rate for the following illustration, and assume these rates to be provided by the service FXDS4.

Suppose the expected behaviors of FXDS2 include:

1. A uniform exchange rate for any deal order.
2. A better, or at least the same, exchange rate to its clients than its rivals (e.g., the service FXDS3).
3. Checks on the exchange rates from central banks dynamically (e.g., the service FXDS4 for Central Bank of China, or FXDS5 for European Central Bank).

Also suppose that the implementation FXDS2 contains the following two faults (see (Table 2):

a. It uses the bid rate or the ask rate to process a deal order non-deterministically.
b. The rate provider logic has faults to cause it to use the minimum (that is, the worst rate)

instead of the maximum (that is, the best rate) for its rate calculations.

To test the service FXDS2, testers can apply our testing proposal, which is illustrated as follows.

Testers first formulate metamorphic relations. For the requirement (i), testers can check whether the output amount of service FXSD2 for a deal order is proportional to the deal order size. It forms the first metamorphic relation:

MR1: nFXDS2(x) = FXDS2(nx).

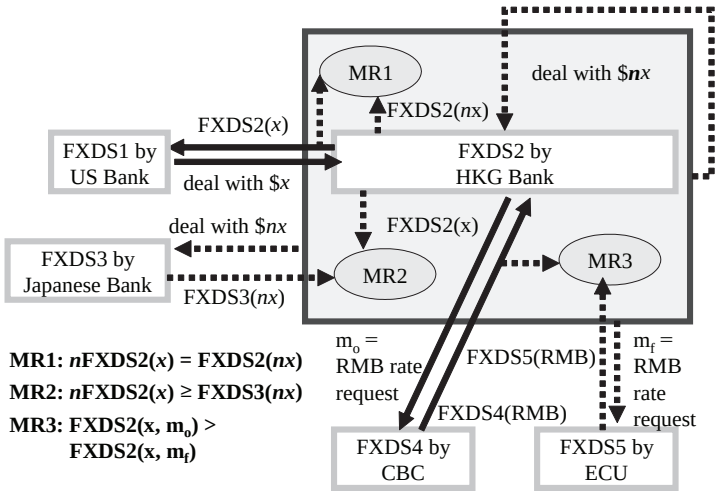
Consider an original test message t_o: a deal order of x (= US$100). FXDS2 correctly uses the above bid rate to output a message FXDS2(US$100) = 827.96 = 8.2796 × 100 in the off-line testing mode by using stubs. The metamorphic service *MS* constructs a follow-up test case t_f: a deal order of US$200 = 2 × x. It then passes this message to FXDS2 to execute in the online testing mode. (This is shown as the top-right dotted arrow in Figure 2).

Suppose, unfortunately, FXDS2 incorrectly uses the above ask rate and outputs FXDS2(US$200) = 1655.94 = 2× 827.97. Since both messages FXDS2(US$100) and FXDS2(US$200) can be checked by *MS* via MR1, we have, 2 × FXDS2(US$100) = 1655.92 ≠ 1655.94 = FXDS2(US$200). It violates the equation MR1. Hence, a failure related to the fault (*a*) is revealed and reported by the metamorphic service.

Readers may be interested to know the results of other combinations of using bid and ask rates for either the original or the follow-up test cases. When the original test case and the follow-up test cases use the bid rate or the ask rate, the equation MR1 will be ineffective to reveal the fault. However, when the original test case incorrectly uses the ask rate and the follow-up test case correctly uses the bid rate, MR1 would reveal the fault.

Let us continue to describe the scenarios to reveal the fault (*b*). We also begin with the formulations of two metamorphic relations. For the requirement (ii), testers may enlarge or shrink a deal order size by a multiplier. (In practice, a deal order size applicable to a global bank may not be applicable to a small foreign exchange shop at a street corner.) Such a follow-up deal order will be forwarded to a competing service (e.g., FXDS3). Next, one can determine whether the output of

Figure 2. A foreign exchange services system.



FXDS2 of the same order size is better than or equal to that provided by a rival service. This formulates the metamorphic relation:

MR2: nFXDS2(x) ≥ FXDS3(n x).

For the requirement (iii), testers may formulate the metamorphic relation:

MR3: value(FXDS2(x)) > value(FXDS2(y)) if centralbank(FXDS2(x)) = m_o and centralbank(FXDS2(y)) = m_f

Alternatively, in a shorthand notation:

MR3: FXDS2(x, m_o) > FXDS2(x, m_f)

MR3 means that if the target exchange is between USD and RMB, then the rate, provided by the Central Bank of China via the rate request m_o, should be strictly better than that due to any other rate request m_f from other central banks. We note that we choose to show the outgoing messages that interact with other services as parameters in the metamorphic relation in the shorthand notation to ease our discussion. We also deliberately use a different notation (that is, m_o and m_f , instead of t_o and t_f) for outgoing messages that serve as test cases for the metamorphic testing being illustrated.

According to MR2, a follow-up test case m_f can be formulated: Deal order of US$60 = 0.3 × 200. Suppose that an implementation of the service FXDS3 is discovered dynamically, and the latter correctly receives m_f and returns a message FXDS3(US$60) = 60 × 8.2796 = 496.776 to *MS*. Both messages FXDS2(US$200) and FXDS3(US$60) are verified by the metamorphic service via MR2. We have, 0.3 × FXDS2(US$200) = 496.782 > 496.776 = FXDS3(US$60). It satisfies MR2. Hence, no failure will be reported by *MS* for this particular follow-up test case.

On the other hand, for online testing, in the execution of a deal order as the input, FXDS2 needs to communicate with services of central banks to collect relevant exchange rates. Thus, the original test case t_o will trigger an outgoing message m_o, a USD−RMB rate request, for such a purpose. Suppose that FXDS2 discovers both FXDS4 and FXDS5 to provide the quotes of exchange rates for USD−RMB. For the illustration purpose, further suppose that the exchange rate provided by the European Central Bank via FXDS5 for USD−RMB is 8.2795/8.2798. This spread is wider than that provided by FXDS4, and thus is poorer.

MS uses the metamorphic relation MR3 to produce the follow-up test case m_f of m_o. Owing to the faults (*a*) and (*b*), FSDS2 incorrectly selects the ask rate of the service FXDS4 to process the deal order. FSDS2 will output a message FXDS2(US$100) = 827.98 = 8.2798 × 100. We have, in the shorthand notation, FXDS2(US$100, m_o) = 827.96 < 827.98 = FXDS2(US$100, m_f). It violates MR3, and *MS* reports a failure due to the combination of faults (*a*) and (*b*).

We have illustrated the usefulness of using a metamorphic service for the testing of services. In reality, there are domain-specific rounding practices compliant to the international accounting practice. Since rounding rules are a part of the application domain, it is natural for the designed metamorphic relations to take this type of rounding rules into account. This type of domain-specific rounding does not affect the metamorphic approach for services testing. It is worth mentioning that there could be implementation-specific rounding errors also. We refer readers to a previous work on metamorphic testing (Chen et al., 2003) for using a metamorphic approach to identify failures in the presence of the implementation-specific rounding errors. In the next section, we report an experiment that applies our proposal and discuss observations.

EXPERIMENT

In this section, we examine two issues in our proposal through an empirical experiment. We would like to study the testing overhead when some of the original test cases have failed. This is to validate our proposal to use successful test cases applicable to off-line testing as the original test cases for online testing. Our rationale is that if the overhead, on the contrary, is marginal; it is unnecessary to predetermine the successfulness of original test cases. This also helps reduce the testing effort for the online testing of services.

The Subject Program

The subject service implements a service-oriented calculator of arithmetic expressions. It is developed in C++ on Microsoft Visual Studio for .NET 2003 as a Web services. It consists of 16 classes with 2,480 lines of code.

Functionally, the calculator accepts an arithmetic expression consisting of constants of designated data type and arithmetic operators. The set of supported operators are {+,–,×,÷} in which each operator is overloaded to enable every operand belonging to different data types. Five data types of operands are supported: integer, long integer, decimal, floating-point number, and floating-point number with double precision. Each operator is implemented as a supportive service of the subject service.

The subject service parses the inputted arithmetic expressions, locates other services, and composes the results from the results returned by its individual supportive services. In the case where different data types are associated with an operator in a sub-expression, the subject service is responsible to resolve the appropriate data types and to pass the sub-expression to an applicable supportive service. In the design of the application, we choose to allow at most three concurrent instances of the above logic in the subject service. They, together with an expression dispatcher instance, compose our subject service. The expression dispatcher discovers the above supportive services of the subject service, and sends outgoing and receives incoming messages for the latter services. The metamorphic service for the subject service is composed of the dispatcher and a program unit that implements a set of metamorphic relations, which will be explained in the next section.

All the above services are developed by a team of five developers. They have completed certain formal software engineering training. Before doing the experiment, they have gained at least one year of research-oriented and application software development to develop location-based systems, mobile positioning systems and common business applications.

Experimental Setup

In this section, we describe the setup configuration for the testing of the subject service. Specifically, we present the selection of the original test cases, the metamorphic relations, and our experience in finding and implementing the metamorphic relations for the experiment.

As described previously, every instance of services is executed on a designated machine. In total, 14 personal computers are used. All the machines to collect our data are located in the Compuware Software Testing Laboratory of the Hong Kong Institute of Vocational Education (Tsing Yi).

The selection of the set of the original test cases is a black-box combinatorial testing approach. This allows the test cases to serve as the original test cases for both the off-line testing mode and online mode. Consider an arithmetic expression of length 3, which we mean to have an expression having three values and two operators (e.g., 13 + 0.45 – 7). In our experiment, there are five types of data type and four types of operator. In total, there are 2,000 (that is, $5^3 \times 4^2$) combinations, not counting the possible choices of value to initialize

each variable. Since the length of an arithmetic expression could be ranged from zero to a huge figure, we choose to fix the length to three in this experiment. This minimizes the potential problem of using a very long expression to compare with a very short expression in our subsequent analysis. We also initialize test cases that do not cause integer value overflow, and the overflow of other data types alike. We are aware that we have implemented some design decisions in the test case selection process. We minimize these highlighted threats by designing our metamorphic relations *neutral* to these design decisions.

In ideal cases, a user would perceive every calculation as if it is derived analytically using mathematics. However, some expressions would incur rounding errors due to the limitation of programming or system configuration. Different orders of evaluations of an expression could thus produce inconsistent results. A good calculator would provide consistent results to its users. For example, the arithmetic expression "(1 ÷ 3) × 3" would be evaluated as "(1 × 3) ÷ 3" by a good calculator.

We follow the above general guideline to formulate our metamorphic relations. We first describe how we determine and implement metamorphic relations. Next, we will describe the chosen metamorphic relations.

The arithmetic operators for the calculator application domain naturally define associative and commutative properties amongst arithmetic operators. We use these properties as the basis to design our metamorphic relations. This aligns with our objective to design such relations neutral to the selection of the original test cases. Since these properties naturally occur in the domain application, we experience a marginal effort to design associative and commutative relations and convert them into their corresponding formats in the sense of metamorphic relation. Furthermore, since the .NET framework for C++ directly supports the chosen operators, the effort to implement the metamorphic relations in our experiment is also marginal.

The types of metamorphic relation used in the experiment are as follows. Suppose *A, B* and *C* are operands of some data type in the set {integer, long integer, floating point number, floating point number with double precision}, and θ_1 and θ_2 are overloaded operators in the set {+, –, ×, ÷}. Consider the expression "(A θ_1 B) θ_2 C." A commutative-property rule to generate the follow-up test cases is: "(A θ_1 B) θ_2 C" ⇒ "C θ_2 (A θ_1 B)". It derives the follow-up test case "C θ_2 (A θ_1 B)" based on the original test case "(A θ_1 B) θ_2 C."

Let us denote our subject service by *S*. The corresponding metamorphic relation of the above commutative rule would be: *S*("(A θ_1 B) θ_2 C") = *S*("C θ_2 (A θ_1 B)"). A metamorphic relation derived from an associative rule is *S* ("(A θ_1 B) θ_2 C") = *S*("A θ_1 (B θ_2 C)"). We also design variants of the follow-up test case derivation rules to deal with the division operator: "(A × B) ÷ C)" ⇒ "(1 ÷ C) × (A × B)," and other similar rules alike. The generation of test cases is implemented as logic in our program.

Let us continue the discussion on the execution of test cases. Even a program is ideal; messages could still be lost or corrupt in a distributed environment. Our subject application does not implement a sophisticated fail-and-retry strategy to handle these anticipated scenarios. To compensate the lacking of this kind of strategy, we choose to abort the execution of a test case when a service returns no output after a timing threshold. After some trials-and-errors in our laboratory environment, we choose the threshold to be 50 seconds per execution of a pair of (original and follow-up) test cases. A typical test case will yield the result with three seconds.

Based on the implementation of the subject application, we create additionally six consecutive faulty versions of the set of supportive services. Each version injects one additional mutation fault to its immediate ancestor. The six consecutive

faults are created in the following order: changing a "+" operator to the operator "–"; changing a "–" operator to the "+" operator'; changing a "×" operator to a "–" operator; swapping the operand of a "÷" operator; changing a "–" operator to the "×" operator; and changing a "×" operator to a "+" operator. These faults support operands of the following data types respectively: floating-point number with double precision, integer, decimal, floating-point number with single precision, long integer, and floating-pointer number with single precision.

The following configurations were deployed for off-line testing and online testing.

- The subject application (the subject service with the original version of the supportive services) simulates an off-line testing environment. The original non-faulty versions of the supportive services serve as stub services for the off-line testing, in the sense of conventional testing. Inexperienced developers implement the subject service and, thus, it naturally contains real faults. Some random test cases do reveal failures from the subject service. We however have reviewed that a set of test cases for the subject service for off-line testing are successful. This set of test cases could be used as the original test cases for online testing according to our approach. We refer this set of test cases to as the set of original test cases in the experiment.
- A faulty application (the subject service with a faulty version of the supportive services) simulates an online testing environment. A faulty version of the supportive services is not identical in behavior to its original counterpart (the test stub used in the off-line testing). The six faulty versions therefore facilitate us to re-use the above set of original test cases yet provide failed test results for some of the elements. This allows us to compare the effect of failed original test cases. In order to avoid biases towards a particular faulty implementation, we put all original test cases and their results of the faulty versions in the same pool for analysis and treat them homogenously.

In total, we executed 22,503 follow-up test cases and also 22,503 original test cases. These two figures are the same because we use metamorphic relations with two input-output pairs in this experiment. For the original version, we execute 3,987 pairs of test cases, and for each faulty version, we execute 3,086 pairs of test cases. To facilitate us to evaluate the effectiveness of test cases, we also mark every test case to be successful or failed, according to the expected result of the arithmetic expression. The result is shown in the next section.

Empirical Results

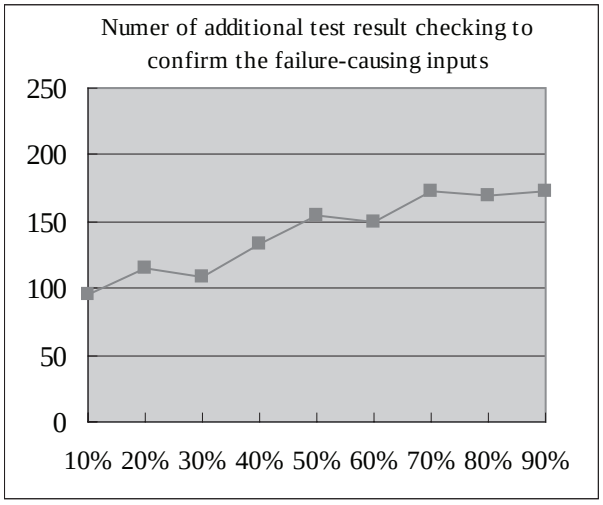
In this section, we will present the empirical results of the experiment and discuss our observations from the results. In summary, our experimental result shows that our approach uses 16% less in terms of effort to detect 13% more in terms of failures, compared to the control experiment having no prior confirmation of test results of original test cases.

We first analyze the overhead, including failed original test cases for online testing. According to the previously mentioned experimental setup, we have collected a pool of 18,516 (= 3,086 × 6) pairs of test cases from the six faulty versions. Some of the pairs contain failed original test cases. We evenly partition the pool into 18 groups so that each group consists of around 1,000 test cases. For the i-th group (for i = 1 to 18), we use database queries to help us randomly draw around 5i percent of its elements having failed original test cases and all elements having successful original test cases to calculate the overhead in terms of test result checking. The total number of elements drawn from each group is shown in Table 1.

Table 1. No. of elements drawn from selected groups

Group	2	4	6	8	10	12	14	16	18	Total
Percentages of failed test cases drawn	10%	20%	30%	40%	50%	60%	70%	80%	90%	-
No. of elements drawn	767	791	809	839	885	904	945	973	997	7,910

Figure 3. The overheads for different percentages of failed original test cases



The calculation of the *overhead* of a test pair is as follows.

- If no violation of the associated metamorphic relation is detected, then the overhead value is zero. It is because the test pair cannot reveal any failure.
- If a violation of the associated metamorphic relation is detected, then the overhead value is equal to the number of actual failure-causing test case(s) in the test pair. In other words, if both test cases are failure-causing inputs, the overhead value will be two; otherwise, it will be one.

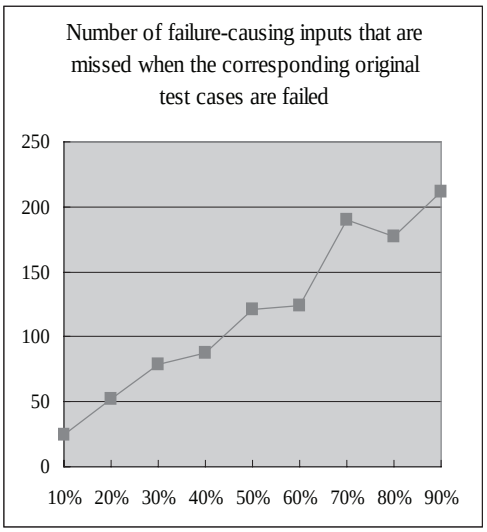
We further define that the number of additional test result checking to confirm the failure-causing inputs for a set of test pairs, denoted by Ω, is the sum of the overhead of every test pair in the set.

Figure 3 shows the value of Ω for 10%, 20% and up to 90% of total number of the failed original test cases of a group for the calculation. The mean value is 141 and the standard derivation is 29.4. (We are aware that there are inadequate numbers of data to calculate a good standard derivation statistically. We show the value just to give readers an impression about the skewness of data set.)

As expected, the value of Ω increases when the percentage increases. When more failed original test cases are added to a set for the calculation of Ω, the chance to detect a failure increases.

However, as we increase the number of test cases from 762 (for group 2) to 997 (for group 18), the change in Ω is moderate and is equal

Figure 4. The number of missed detection of failure-causing inputs



to 77 (= 173 – 96). When around 10% of failed original test cases are included, the value of Ω is 96. This initial value, 96, is even larger than the cumulated change, 77, increasing the percentages from 10% to 90%.

It confirms our assertion that the inclusion of small percentages of failed original test cases for the online testing of a service accounts for a major source of overhead in terms of additional test result checking. This substantiates our recommendation of using successful original test cases only for online testing to alleviate the test oracle problem.

We also observe that the overall percentage is not high. The mean is 15.9%. It suggests that even testers had to include inevitably some failed original test cases; measures should be taken to minimize such an inclusion.

We further examine the chance that we would miss to detect a failure if some failed original test cases are used for online testing. In our experiment, we use simple equations (instead of inequality) as metamorphic relations deliberately. We recall that, in our methodology, an original test case is known to be successful. It follows that any failure detected by the metamorphic testing approach would be due to the failure of the follow-up test case. On the other hand, when no violation of a metamorphic relation (in the form of equation) could be detected, the follow-up test case will be regarded as successful in the experiment.

For a set of test pairs, we define ω as the number of test pairs that each pair consists of one failed original test case and one follow-up test case, and at the same time, the associated metamorphic relation cannot detect the failure. The ω thus measures the missed opportunities to detect failures for online testing.

We also use the data sets reported in Table 1 to calculate ω for each group. The results are shown in Figure 4. The mean value is 118.7, that is, 13% of all failure-causing cases. The corresponding standard derivation is 64.2. The minimum is 25 for Group 2 (that is, the 10% group) and the maximum is 212 for Group 18 (that is, the 90% group).

The trend of missed detection of failures ω is increasing as the percentage of failed original test cases included in a group increases. Moreover, the number of missed detection appears proportional to the percentages of failed original test cases in the data set. This observation looks interesting and warrants further investigations.

RELATED WORK

Literatures on function testing research for SOA applications or Web services are not plenty. Bloomberg (2002) overviews a few major types of testing activity for testing Web services. We refer interested readers to the work of Bloomberg (Bloomberg, 2002) for the overview. In the rest of the section, we review selected related work on providing tools for testing, exception handling testing, generation of test cases and test result evaluation for service-oriented programs.

The use of the testing tool is an indispensable part to automate testing activities. Dustar and Haslinger (2004) propose to develop an experimental testing tool prototype to conduct batch mode and online mode testing. Their tool is reported to expect to handle several types of testing such as functional test and reliability test. Deng et al., (2004) extend AGENDA, a database application testing tool that populates database instances as a part of a test case, to test Web-based database applications. Rather than populating test database instances, SOATest (Parasoft, 2005) also reads data from different data sources to conduct testing. On the Internet, there are quite a few tools to conduct load and stress tests for Web services. Their discussions are not within the scope of this article.

To generate test cases, Offutt and Xu (2004) propose a set of mutation operators to perturb messages of Web services. They also suggest three types of rules to develop test cases based on the XML schema of messages. Their initial result on a sample application shows that 78% of seeded faults can be revealed. These faults include communication faults in SOAP, faults in statements for database query languages and conventional faults. The mutated messages could be considered as a method to construct follow-up test cases in the sense of metamorphic testing. Their empirical study of their approach is valuable. Rather than injecting faults to messages, Looker and Xu (2003) inject faults in SOAP-based programs to test the programs for robustness.

Chen et al., (2004a) also aim at testing for robustness. They focus on testing the exception handling of Java Web services through a data flow coverage approach. They propose to inject faults to system calls to trigger exception throws and require test cases to cover the exception-oriented def-use relations. Through their subject programs are Java-based Web service program; their techniques are not specialized to services testing.

Tsai et al., (2004, 2005) propose an approach to testing Web services that each service has multiple implementations with the same intended functionality. They apply test cases to a set of implementations of the same intended functionality progressively. Their test results are ranked by a majority voting strategy to assign a winner as the test oracle. A small set of winning implementations is selected for the integration testing purpose. At an integration testing level, they follow the same approach except using a weighted version of the majority voting strategy instead. To deal with the test case selection problem, their research group (Tsai et al., 2002a) proposes to generate test cases based on WSDL. Although WSDL-based testing has been proposed for a few years and implemented in testing tools such as SOATest (Parasoft, 2005), their version of WSDL (Tsai et al., 2002b) extends the standardized WSDL (W3C, 2001) to include features to express semantics. It is not difficult to use other voting strategies instead of the majority voting strategy in their approach.

The advantages of semantics checking are also observed by other researchers. An immediate example is our adaptation of metamorphic relation in this article. Keckel and Lohmann (2005), on the other hand, propose to apply the notion of design by contract to conduct testing for Web services. They suggest defining formal contracts to describe the behavior of functions and interactions of Web services. Based on the

contracts, combinatorial testing is suggested to apply to conduct conformance testing against the contracts of intended services.

Our approach uses metamorphic relations to construct follow-up test cases. It is not a fault-based approach; whereas Looker and Xu (2003) suggest a fault-based approach to testing for program robustness. In addition, unlike the work of Offutt and Xu (2004), these follow-up test cases in our approach are intentional to allow automated test result evaluations. A follow-up test case triggers a service to produce an output. The output of a follow-up test case is allowed to be different from the output of the original test case. Hence, it can be applied to configurations when multiple implementations of the same functionality are too expensive to be used. This distinguishes our work from Tsai et al., (2005). Our approach checks test results amongst themselves; whereas the work of Keckel and Lohmann (2005) checks the test results against some formal contract specifications. Chen et al., (2004a) is a white-box-centric approach; whereas ours is a black-box-centric approach. The tools (Dustar & Haslinger, 2004; Parasoft, 2005) appear to be developed as an integrated testing tool that includes a number of different testing modules. Their architectures are unclear to us at this moment. Our metamorphic service is an access wrapper to facilitate off-line and online testing of a service. We aim at reducing the amount of non-core elements from our metamorphic service. In this way, when exposing the function to other services, we would like to develop the approach further so that it enjoys a good scalability.

DISCUSSION

We have presented an approach to online testing and an experiment on a simple application to study metamorphic testing for services testing in the previous sections. The approach is useful for the online testing of a service if testers have difficulties to obtain the expected results of a test case in a cost-effective manner. The test oracle problem has been identified for many years (see also (Beizer, 1990)). In practice, many testers validate their software without a formal test oracle. Metamorphic testing is an approach towards the problem when a metamorphic relation exists and could be identified. We believe that this trend will continue for services testing.

In the rest of this section, we discuss the threat to validity of the experiment. The evaluation of using the metamorphic testing approach for services testing in the online mode is conducted in a small application. The application domain of the subject application is generic that it uses the basic computing functionality, namely a fundamental set of arithmetic operations and basic (communicative and associative) properties to define metamorphic relations. It is unknown about the impact on the results when certain domain-specific knowledge is taken into account to both define metamorphic relations and selections of test cases. We merely use a small number of faulty versions to conduct the experiment to discuss our findings. The statistics may be different if other faulty versions are used. Our subject program does contain real faults and hence some metamorphic relation instances are violated even if the original version is used. We believe that it is common for a typical testing of a newly-coded software program. However, the assumption is not valid if other rigorous quality assurance techniques such as rigorous regressions, code inspections or formal methods have been applied to a software development project. We have evaluated our approach on the Microsoft .NET 2003 platform only. The present implementation of the subject program is not portable to other platforms. There is a threat to interpret the empirical results in other configurations. We have reviewed the set test cases and the set follow-up test cases, and use a commercial spreadsheet to help us to check whether the target relation is maintained in the reviewed samples.

CONCLUSIONS AND FUTURE WORK

Testing services in a services-computing environment needs to deal with a number of issues. They include: (i) The unknown communication partners until the service discovery; (ii) the imprecise black-box information of software components; (iii) the potential existence of non-identical implementations of the same service; and (iv) the expected behavior of a service potentially depending on the behavior of competing services. In this article, we treat a service as a reusable software module with a well-defined function. A service can introduce itself so that other services can discover and use the service. Services communicate amongst themselves through well-defined messages and interfaces. A message is an input or an output of a service.

We have presented a testing approach to the online testing support of service-oriented applications. We formulate the notion of metamorphic service; the service that has the characteristics of being an access wrapper, the wrapper encapsulates the access for the service under test and implements the metamorphic testing approach. We also propose to use the successful test case for off-line testing as the original test case for online testing, as test oracle is much more likely to be available for off-line testing. Using our online testing methodology, testers builds a bridge between the test oracle available in the off-line testing mode and the test oracle problem encountered in the online testing mode. The services approach to conducting an online test alleviates the problem (i). It delays the binding of communication partners of the follow-up test cases after service discovery. Our realization of the metamorphic testing approach alleviates the problems (ii), (iii) and (iv).

We have also conducted an experiment to evaluate the feasibility of our proposal. The experimental results encouragingly indicate that, on average, when the set of original test cases are unknown to be successful, an extra 16% effort to check test results and a 13% reduction of failure detection are observed. This supports our proposal that original test cases should be (much) better to be successful, particularly when the checking is (much) less costly when it can be conducted in the off-line testing mode.

There are quite a number of future directions of research. We have not evaluated our proposal extensively. We plan to conduct more experiments. The way to control the chain reaction of the follow-up test cases generations due to interferences of multiple metamorphic services warrants more researches. We also plan to measure the degree of code coverage or fault coverage of our approach.

ACKNOWLEDGMENT

We would like to thank the program co-chairs of The First International Workshop on Services Engineering (SEIW 2005) for inviting us to extend the preliminary version (Chan et al., 2005a) to contribute to the special issue. We would also like to thank anonymous reviewer of the article. Special thanks should be given to Mr. Kwong–Tim Chan, Mr. Hoi-Shun Tam, Mr. Kwun-Ting Lee, Mr. Yuk-Ching Lam and Mr. King–Lun Yiu of the Hong Kong Institute of Vocational Education (Tsang Yi) who do the experiments presented in this article. Part of the research was done when Chan was with The University of Hong Kong. This research is supported by grants of the Research Grants Council of Hong Kong (Project Nos. HKUST 6170/04E and CITYU1195/03E).

REFERENCES

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison Wesley.

Beizer, B. (1990). *Software Testing Techniques*. New York:Van Nostrand Reinhold.

Bloomberg, J. (2002). *Testing web services today and tomorrow.* Retrieved from http://www-106.ibm.com/developerworks/rational/library/content/rationaledge/oct02/webtesting_therationaledge_oct02.pdf .

Chan, F.T., Chen, T.Y., Cheung, S.C., Lau, M.F., & Yiu, S.M. (1998). Application of metamorphic testing in numerical analysis. In *Proceedings of IASTED International Conference on Software Engineering (SE 1998),*(pp. 191–197). Calgary, Canada: ACTA Press.

Chan, W. K., Cheung, S. C., & Leung, K. R. P. H. (2005a). Towards a metamorphic testing methodology for service-oriented software applications, The First International Workshop on Services Engineering (SEIW 2005). In *Proceedings of the 5th Annual International Conference on Quality Software (QSIC 2005).* Los Alamitos, CA:IEEE Computer Society.

Chan, W. K., Cheung, S. C., & Tse, T. H. (2005b). Fault-based testing of database application programs with conceptual data model. In *Proceedings of the 5th Annual International Conference on Quality Software (QSIC 2005).* Los Alamitos, CA:IEEE Computer Society.

Chan, W. K., Chen, T. Y., Lu, Heng, Tse, T. H., & Yau, S. S. (2005c). A metamorphic approach to integration testing of context-sensitive middleware-based applications. In *Proceedings of the 5th Annual International Conference on Quality Software (QSIC 2005).* Los Alamitos, CA: IEEE Computer Society.

Chen, F., Ryder, B., Milanova, A., & Wannacott, D. (2004a). Testing of java Web services for robustness. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004),* (pp. 23–34).New York: ACM Press.

Chen, T. Y., Huang, D. H., Tse, T. H., & Zhou, Z. Q. (2004b). Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004),* (pp. 569–583). Madrid, Spain: Polytechnic University of Madrid.

Chen, T. Y., Tse, T. H., & Zhou, Z. Q. (2002). Semi-proving: An integrated method based on global symbolic evaluation and metamorphic testing. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2002).* (pp. 191–195). New York:ACM Press.

Chen, T. Y., Tse, T. H., & Zhou, Z. Q. (2003). Fault-based testing without the need of oracles. *Information and Software Technology, 45* (1), 1–9.

Chen, T.Y., Cheung, S.C., & Yiu, S.M. (1998). *Metamorphic testing: A new approach for generating next test cases* (Tech. Rep. HKUST-CS98-01). Hong Kong: Hong Kong University of Science and Technology, Department of Computer Science.

Chunyang Ye, S.C. Cheung & W.K. Chan (2006). Publishing and composition of atomicity-equivalent services for B2B collaboration. In *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006).* Los Alamitos, CA:IEEE Computer Society.

Colan, M. (2004). *Service-oriented architecture expands the vision of Web Services, part 1: Characteristics of service-oriented architecture*. Retrieved from http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro.html.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. & Weerawarana, S. (2002). Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing, 6* (2), 86–93.

Deng, Y., Frankl, P., & Wang, J. (2004). Testing web database applications, SECTION: Workshop on testing, analysis and verification of web ser-

vices (TAV-WEB) papers. *SIGSOFT Software Engineering Notes, 29* (5).

Dustar, S., & Haslinger, S. (2004). Testing of service-oriented architectures: A practical approach. In *Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (NODe 2004),* (pp. 97–112). Berlin, Heideberg: Springer-Verlag.

Frankl, P. G. & Weyuker, E. J. (1988). An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering, 14* (10), 1483–1498.

Kapfhammer, G. M., & Soffa, M. L. (2003). A family of test adequacy criteria for database-driven applications. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE 2003)* (pp. 98–107). New York: ACM Press.

Keckel, R. & Lohmann, M. (2005). Towards contract-based testing of Web services. *Electronic Notes in Theoretical Computer Science, 116,* 145–156.

Kreger, H. (2003). Fulfilling the web services promise. *Communications of the ACM, 46* (6), 29–34.

Looker, N., & Xu, J. (2003, October). Assessing the dependability of SOAP RPC based Web services by fault injection. In *Proceeding of IEEE International Workshop on Object-Oriented, Real-Time and Dependable Systems,* Capri Island.

Mecella, M. & Pernici, B. (2003), Designing wrapper components for e-services in integrating heterogeneous systems. *The VLDB Journal, 10* (1), 2–15.

Mukhi, N. K., Konuru, R., & Curbera, F. (2004). Cooperative middleware specialization for service oriented architectures. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW 2004),* (pp. 206–215) New York: ACM Press.

OASIS (2005). *Universal Description, Discovery and Integration* (UDDI) version 3.0.2, retrieved from http://uddi.org/pubs/uddi_v3.htm.

Offutt, J. & Xu, W. (2004). Generating test cases for web services using data perturbation, SECTION: Workshop on testing, analysis and verification of web services (TAV-WEB) papers, *SIGSOFT Software Engineering Notes, 29* (5).

Parasoft Corporation (2005). SOATest, Retrieved from http://www.parasoft.com/jsp/products/home.jsp?product=SOAP&itemId=101 .

Rajasekaran, P., Miller, J. A., Verma, K., Sheth, A. P. (2004). Enhancing Web services description and discovery to facilitate composition. In *Proceeding of The First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004),* LNCS 3387, (pp. 55-68). Berlin, Heideberg: Springer-Verlag.

Tsai, W. T., Chen, Y., Paul, R., Huang, H., Zhou, X., & Wei, X. (2005). Adaptive testing, oracle generation, and test case ranking for Web services. In *Proceedings of the 29th Annual International Computer Software and Applications conference (COMPSAC 2005),* (pp. 101–106). Los Alamitos, CA :IEEE Computer Society.

Tsai, W. T., Chen, Y., Cao, Z. Bai, X., Hung, H., & Paul, R. (2004). Testing Web services using progressive group testing. In *Proceedings of Advanced Workshop on Content Computing (AWCC 2004),* LNCS 3309, (pp. 314–322). Berlin, Heideberg:Springer-Verlag.

Tsai, W. T., Paul, R., Wang, Y., Fan, C., 7 Wang, D. (2002a). Extending WSDL to facilitate Web services testing. In *Proceedings of The 7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002),* (pp. 171–172) Los Alamitos, CA: IEEE Computer Society.

Tsai, W. T., Paul, R., Song, W., & Cao Z. (2002b). Coyote: An XML-based framework for Web services testing. In *Proceedings of The 7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002)*, (pp. 173–176). Los Alamitos, CA: IEEE Computer Society.

Tse, T. H., Yau, S. S., Chan, W. K., Lu, H., & Chen T. Y. (2004). Testing context-sensitive middleware-based software applications. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*, (pp. 458–466). Los Alamitos, CA:IEEE Computer Society.

W3C (2003). *SOAP Version 1.2 Part 1: Messaging Framework*. Retrieved from http://www.w3.org/tr/soap12-part1/.

W3C (2002). Web Services Activity, Retrieved from: http://www.w3.org/2002/ws .

W3C (2005). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Retrieved from http://www.w3.org/tr/wsdl20/ .

W3C (2004). *Extensible Markup Language (XML) 1.0 (Third Edition)*. Retrieved from http://www.w3.org/TR/2004/REC-xml-20040204/ .

W3C (2001). *XML Schema*. Retrieved from http://www.w3.org/xml/schema .

Umar, A. (1997). *Application reengineering. Building web-based applications and dealing with legacy*. Cliffs, NJ: Prentice-Hall, Englewood

Zhu, H., Hall, P. A. V., & May, J. H. R. (1997). Software unit test coverage and adequacy. *ACM Computing Survey, 29* (4), 366–427.

ENDNOTES

1 All correspondence should be addressed to Dr. W. K. Chan at Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Tel: (+852) 2358 7016. Fax: (+852) 2358 1477. Email: wkchan@cse.ust.hk .

2 XML stands for Extensible Markup Language (W3C, 2004).

3 IBM also proposes to design stateless Web services when it introduces service-oriented architecture (see http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro.html .)

4 We refer readers to Section 0 for the role of a tester in the design and implementation of metamorphic relations, and refer readers to Section 0 for the illustration to construct test cases in the metamorphic approach.

5 Although services are highly recommended to publish their functional properties for other services to judge its usefulness in their areas of concerns and to subscribe to use the service, yet the design of metamorphic relations, the implicit form of functional properties, is not within the scope of this paper. We refer interested readers to a case study (Chen et al., 2004) on the selection of effective metamorphic relations for more details.

This work was previously published in International Journal of Web Services Research, Vol. 4, Issue 2, edited by L. Zhang, pp. 61-81, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.18
Deductive Semantics of RTPA

Yingxu Wang
University of Calgary, Canada

ABSTRACT

Deductive semantics is a novel software semantic theory that deduces the semantics of a program in a given programming language from a unique abstract semantic function to the concrete semantics embodied by the changes of status of a finite set of variables constituting the semantic environment of the program. There is a lack of a generic semantic function and its unified mathematical model in conventional semantics, which may be used to explain a comprehensive set of programming statements and computing behaviors. This article presents a complete paradigm of formal semantics that explains how deductive semantics is applied to specify the semantics of real-time process algebra (RTPA) and how RTPA challenges conventional formal semantic theories. Deductive semantics can be applied to define abstract and concrete semantics of programming languages, formal notation systems, and large-scale software systems, to facilitate software comprehension and recognition, to support tool development, to enable semantics-based software testing and verification, and to explore the semantic complexity of software systems. Deductive semantics may greatly simplify the description and analysis of the semantics of complicated software systems specified in formal notations and implemented in programming languages.

INTRODUCTION

Semantics in linguistics is a domain that studies the interpretation of words and sentences, and analysis of their meanings. Semantics deals with how the meaning of a sentence in a language is obtained, hence the sentence is comprehended. Studies on semantics explore mechanisms in the understanding of languages and their meanings on the basis of syntactic structures (Chomsky, 1956,

1957, 1959, 1962, 1965, 1982; Tarski, 1944).

Software semantics in computing and computational linguistics have been recognized as one of the key areas in the development of fundamental theories for computer science and software engineering (Bjoner, 2000; Gries, 1981; Hoare, 1969; McDermid, 1991; Slonneg & Kurts, 1995; Wang, 2006b, 2007c). The semantics of a programming language is the behavioral meaning that constitute what a syntactically correct instructional statement in the language is supposed to do during run time. The development of formal semantic theories of programming is one of the pinnacles of computing and software engineering (Gunter, 1992; Meyer, 1990; Louden, 1993; Bjoner, 2000; Pagan, 1981).

Definition 1. *The semantics of a program in a given programming language is the logical consequences of an execution of the program that results in the changes of values of a finite set of variables and/or the embodiment of computing behaviors in the underpinning computing environment.*

A number of formal semantics, such as the *operational* (Marcotty & Ledgard, 1986; Ollongren, 1974; Wegner, 1972; Wikstrom, 1987), *denotational* (Bjorner and Jones, 1982; Jones, 1980; Schmidt, 1988, 1994, 1996; Scott, 1982; Scott & Strachey, 1971), *axiomatic* (Dijktra, 1975, 1976; Gries, 1981; Hoare, 1969), and *algebraic* (Goguen, Thatcher, Wagner, & Wright, 1977; Gougen & Malcolm, 1996; Guttag & Horning, 1978), have been proposed in the last three decades for defining and interpreting the meanings of programs and programming languages. The classic software semantics are oriented on a certain set of software behaviors that are limited at the level of language statements rather than that of programs and software systems. There is a lack of a generic semantic function and its unified mathematical model in conventional semantics, which may be used to explain a comprehensive set of programming statements and computing behaviors. The mathematical models of the target machines and the semantic environments in conventional semantics seem to be inadequate to deal with the semantics of complex programming requirements, and to express some important instructions, complex control structures, and the real-time environments at run time. For supporting systematical and machine enabled semantic analysis and code generation in software engineering, the *deductive semantics* is developed that provides a systematic semantic analysis methodology.

Deduction is a reasoning process that discovers new knowledge or derives a specific conclusion based on generic premises such as abstract rules or principles (Wang, 2006b, 2007a, 2007c). The nature of semantics of a given programming language is its computational meanings or embodied behaviors expressed by an instruction in the language. Because the carriers of software semantics are a finite set of variables declared in a given program, program semantics can be reduced onto the changes of values of these variables over time. In order to provide a rigorous mathematical treatment of both the abstract and concrete semantics of software, a new type of formal semantics known as the deductive semantics is presented.

Definition 2. *Deductive semantics is a formal semantics that deduces the semantics of a program in a given programming language from a generic abstract semantic function to the concrete semantics, which are embodied onto the changes of status of a finite set of variables constituting the semantic environment of computing.*

This article presents a comprehensive theory of deductive semantics of software systems. The mathematical models of deductive semantics and the fundamental properties are described. The deductive models of semantics, semantic function, and semantic environment at various composing levels of programs are introduced. Properties of software semantics and relationships between

the software behavioral space and the semantic environment are studied. New methods such as the semantic differential and semantic matrix are developed to facilitate deductive semantic analyses from a generic semantic function to a specific semantic matrix, and from semantics of statements to those of processes and programs. The establishment of the deductive semantic rules of RTPA (Wang, 2002, 2003, 2006a, 2006b, 2007a, 2007b, 2008a, 2008b) is described, where the semantics of a comprehensive set of processes is systematically modeled.

THE THEORY OF DEDUCTIVE SEMANTICS

This section presents the theory of deductive semantics (Wang, 2006b, 2007c). A generic mathematical model of deductive semantics of software is developed, and the concepts of semantic environment and semantic function are rigorously defined. Based on them, deductive semantics of programs at different composition levels are rigorously modeled. Then, common properties of software semantics are analyzed.

The Semantic Environment and Semantic Function

Definition 3. *A semantic environment* Θ *of a programming language is a logical model of a set of identifiers I and their values V bound in pairs, i.e.:*

$$\begin{aligned}\Theta &\triangleq f: I \to V, V \subseteq \mathbb{R} \\ &= \{\mathop{R}_{k=1}^{\#I}(i_k, v_k)\} \\ &= \{(i_1, v_1), (i_2, v_2), ..., (i_{\#I}, v_{\#I})\}\end{aligned} \quad (1)$$

where $\mathbb{R}$ is the set of real numbers, $i_k \in I$, $v_k \in V \subseteq \mathbb{R}$, and *#I* the number of elements in *I*.

Note the big-R notation is adopted to denote a set of recurring structures or repetitive behaviors (Wang, 2002, 2007c, 2008a). The semantic environment constituting the behaviors of software is inherently a three dimensional structure known as those of operations, memory space, and time.

Definition 4. *The behavioral space* Ω *of a program executed on a certain machine is a finite set of variables operated in a 3-D state space determined by a triple, i.e.:*

$$\Omega \triangleq (OP, T, S) \quad (2)$$

where *OP* is a finite set of *operations*, *T* is a finite set of discrete *time* points of program execution, and *S* is a finite set of *memory locations* or their logical representations by *identifiers* of variables.

According to Definitions 3 and 4, the set of *variables* of a program, *S*, plays an important role in semantic modeling and analysis, because they are the objects of software behavioral operations and the carriers of program semantics. Variables can be classified as *free* and *system* variables. The former are user defined and the latter are language provided. From a functional point of view, variables can be classified into *object representatives*, *control variables*, *result containers*, and *address locaters*. The life spans or scopes of variables can be categorized as *persistent*, *global*, *local*, and *temporal*. The persistent variables are those that their lifespan are longer than the program that generates them, such as data in a database or files in a distributed network.

A new calculus introduced in deductive semantics is the *partial differential of sets* (Wang, 2006b, 2007c), which is used to facilitate the instantiation of abstract semantics by concrete ones, as described below.

Definition 5. *Given two sets X and U,* $X \subseteq \wp U$*, a partial differential of X on U with elements x,* $x \in$

X, denoted by ∂U/∂x, is an elicitation of interested elements from U as specified in X, i.e.:

$$\frac{\partial U}{\partial x} \triangleq X \cap U, \ x \in X$$
$$= X, \ X \subseteq \text{Þ}U \tag{3}$$

where Þ*U* denotes a power set of *U*.

The partial differential of sets can be easily extended to double, triple, or more generally, multiple partial differentials as defined below.

Definition 6. *A multiple partial differential of X_1, X_2, …, and X_n on ÞU with elements $x_1 \in X_1$, $x_2 \in X_2$, ..., and $x_n \in X_n$, denoted by*

$$\frac{\partial^n}{\partial x_1 \, \partial x_2 \ldots \partial x_n} U,$$

is a Cartesian product of all partial differentials that select interested elements from U as specified in X_1, X_2, …, and X_n, respectively, i.e.:

$$\frac{\partial^n}{\partial x_1 \, \partial x_2 \ldots \partial x_n} U \triangleq X_1 \times X_2 \times \ldots \times X_n \tag{4}$$

where $X_1, X_2, \ldots, X_n \subseteq \text{þ}U$ and $\forall i \neq j$, $l \leq i, j \leq n$, $X_i \cap X_j = \varnothing$.

For example,

$$\frac{\partial^2}{\partial \text{x} \, \partial \text{y}} U = X \times Y, \ x \in X, y \in Y, \text{and } X, Y \subseteq \text{þ}U$$

and

$$\frac{\partial^3}{\partial \text{x} \, \partial \text{y} \, \partial z} U = X \times Y \times Z, x \in X, y \in Y, x \in X, \text{and } X, Y, Z \subseteq \text{þ}U.$$

On the basis of the definitions of software behavioral space and partial differential of sets, the semantic environment of software can be formally described.

Definition 7. *The semantic environment Θ of a program on a certain target machine is its run-time behavioral space Ω projected onto the Cartesian plane determined by T and S, i.e.:*

$$\Theta = \frac{\partial^2 \Omega}{\partial t \, \partial s}, \ t \in T \wedge s \in S$$
$$= \frac{\partial^2 \Omega}{\partial t \, \partial s}(OP, T, S)$$
$$= T \times S \tag{5}$$

As indicated in Definition 7, the semantic environment of a program is a dynamic space over time, because following each execution of a statement in the program, the semantic environment Θ, particularly the sets of variables *S* and their values *V*, may be changed.

In semantic analysis, the changed part of the semantic environment Θ is particularly interested, which is the embodiment of software semantics. A generic semantic function is developed below, which can be used to derive a specific and concrete semantic function for a given statement, process, or program by mathematical deduction.

Definition 8. *A semantic function of a program ℘, $f_\theta(\wp)$, is a function that maps the semantic environment Θ into a finite set of values V determining by a Cartesian product on a finite set of executing steps T and a finite set of variables S, i.e.:*

$$f_\theta(\wp) \triangleq f : T \times S \to V =$$
$$\begin{pmatrix} & \mathbf{s_1} & \mathbf{s_2} & \cdots & \mathbf{s_m} \\ \mathbf{t_0} & \perp & \perp & \cdots & \perp \\ \mathbf{t_1} & v_{11} & v_{12} & & v_{1m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{t_n} & v_{n1} & v_{n1} & \cdots & v_{nm} \end{pmatrix} \tag{6}$$

where $T = \{t_0, t_1, \ldots, t_n\}$, $S = \{s_1, s_2, \ldots, s_m\}$, and V is a finite set of values $v(t_i, s_j)$, $0 \leq i \leq n$, and $1 \leq j \leq m$.

In Equation 6, all values of $v(t_i, s_j)$ at t_0 is undefined for a program as denoted by the bottom symbol $\perp$, i.e. $v(0, s_j) = \perp$, $1 \le j \le m$. However, for a statement or a process, it is usually true that $v(0, s_j) \neq \perp$ dependent on the context of previous statement(s) or the initialization of the system.

According to Definitions 7 and 8, the semantic environment and the domain of a semantic function can be illustrated by a semantic diagram as described below (Wang, 2006b, 2007c).

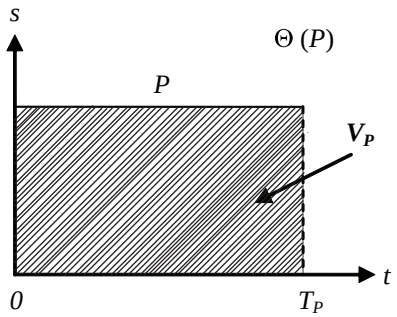
Definition 9. *A semantic diagram is a sub Cartesian-plane in the semantic environment* Θ *that forms the domain of the semantic function for a given process P with* $f_\theta(P) = f: T_P \times S_P \rightarrow V_P$.

For example, the semantic diagram of an abstract process *P*, $f_\theta(P)$, as defined in Definition 9 can be illustrated in Figure 1, where V_P is the domain of dynamic variable values of process *P* over time, i.e., $V_P = T_P \times S_P$. The semantic diagram of two sequential processes, $P \rightarrow Q$, can be referred to Figure 3.

The semantic diagram can be used to analyze complex semantic relations, and to demonstrate semantic functions and their semantic environments. Observing Figures 1 and 3, the flowing properties of process relations can be derived.

Lemma 1. *The variables of two arbitrary processes P and Q,* S_P *and* S_Q*, in the semantic environment* Θ *possess the following properties:*

Figure 1. The semantic diagram of a process



a. *The entire set of variables:*

$$S = S_P \cup S_Q \tag{7}$$

b. *Global variables:*

$$S_G \subseteq S_P \cap S_Q \tag{8}$$

c. *Local variables:*

$$S_L = S - S_G,\ S_L \subseteq S_P \oplus S_Q,$$

where $S_{Lp} = S_L \setminus S_Q$ *and* $S_{Lq} = S_L \setminus S_P$

(9)

Deductive Semantics of Programs at Different Levels of Compositions

According to the generic model and the hierarchical architecture of programs, the semantics of a program in a given programming language can be described and analyzed at various composition levels, such as those of *statement*, *process*, and *system* from the bottom-up (Wang, 2007c, 2008b).

Definition 10. *The semantics of a statement p,* $\theta(p)$*, on a given semantic environment* Θ *is a double partial differential of the semantic function* $f_\theta(p)$ *on executing steps T and the set of variables S, i.e.:*

$$\begin{aligned}
\theta(P) &\triangleq \underset{\#T(p)\ \#S(p)}{\frac{\partial^2}{\partial t\ \partial s}} f_\theta(p) \\
&= \underset{i=0}{\overset{1}{R}} \left(\underset{j=1}{\overset{\#\{s_1, s_2, \ldots, s_m\}}{R}} v_p\ t_i, s_j \right) \\
&= \underset{i=0}{\overset{1}{R}} \left(\underset{j=1}{\overset{\#\{s_1, s_2, \ldots, s_m\}}{R}} v_p\ t_i, s_j \right) \\
&= \begin{pmatrix} & \mathbf{s_1} & \mathbf{s_2} & \cdots & \mathbf{s_m} \\ \mathbf{t_0} & v_{01} & v_{02} & \cdots & v_{0m} \\ (\mathbf{t_0}, \mathbf{t_1}] & v_{11} & v_{12} & \cdots & v_{1m} \end{pmatrix}
\end{aligned} \tag{10}$$

where t denotes the discrete time immediately before and after the execution of p during (t_0, t_1), and # is the cardinal calculus that counts the number of elements in a given set, i.e. n = #T(p) and m=#S(p).

In Definition 10, the first partial differential selects all related variable $S(p)$ of the statement p from Θ. The second partial differential selects a set of discrete steps of p's execution $T(p)$ from Θ. According to Definition 10, the semantics of a statement can be reduced onto a semantic function that results in a 2-D matrix with the changes of values of all variables over time of program execution.

On the basis of Definitions 8 and 10, semantics of individual statements can be analyzed using Equation 10 in a deductive process.

Example 1. *Analyze the semantics of Statement 3, $\theta(p_3)$, in the following program entitled sum:*

```
void sum;
{
  (0) int x, y, z;
  (1) x = 8;
  (2) y = 2;
  (3) z := x + y;
}
```

According to Definition 10, the semantics of Statement p_3 is as follows:

$$\begin{aligned}\theta(p_3) &= \frac{\partial^2}{\partial t\,\partial s} f_\theta(p_3) \\ &= \mathop{R}_{i=2}^{3} (\mathop{R}_{j=1}^{\#S(p_3)} v_{p_3}\ t_i, s_j) \\ &= \mathop{R}_{i=2}^{3} (\mathop{R}_{j=1}^{\#\{x,y,z\}} v_{p_3}\ t_i, s_j) \\ &= \begin{pmatrix} & \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \mathbf{t_2} & 8 & 2 & \perp \\ \mathbf{(t_2, t_3]} & 8 & 2 & 10 \end{pmatrix}\end{aligned} \tag{11}$$

This example shows how the concrete semantics of a statement can be derived on the basis of the generic and abstract semantic function as given in Definition 10.

Definition 11. *The semantic effect of a statement p, $\theta^*(p)$, is the resulted changes of values of vari-*

Box 1.

$$\begin{aligned}\theta^*(p) &= \mathop{R}_{j=1}^{\#S(p)} (v_p(t_i, s_j) \oplus v_p(t_{i+1}, s_j)) \\ &= \mathop{R}_{j=1}^{\#S(p)} < v_p(t_i, s_j) \to v_p(t_{i+1}, s_j) \mid v_p(t_i, s_j) \neq v_p(t_{i+1}, s_j) >\end{aligned} \tag{12}$$

Box 2.

$$\begin{aligned}\theta^*(p_3) &= \mathop{R}_{j=1}^{\#S(p_3)} < v_{p_3}(t_2, s_j) \to v_{p_3}(t_3, s_j) \mid v_{p_3}(t_2, s_j) \neq v_{p_3}(t_3, s_j) > \\ &= \mathop{R}_{j=1}^{\#(x,y,z)} < v_{p_3}(t_2, s_j) \to v_{p_3}(t_3, s_j) \mid v_{p_3}(t_2, s_j) \neq v_{p_3}(t_3, s_j) > \\ &= \{< v_{p_3}(t_2, z) = \perp \ \to v_{p_3}(t_3, z) = 10 >\}\end{aligned}$$

ables by its semantic function θ(p) during the time interval immediately before and after the execution of p, Δt = (t_i, t_{i+1}), see Box 1, where → denotes a transition of values for a given variable.

Example 2. *For the same statement p_3 as given in Example 1, determine its semantic effect θ*(p_3).*

According to Equation 12, the semantic effect θ*(p_3) is seen in Box 2.

It is noteworthy in Examples 1 and 2 that deductive semantics can be used not only to describe the *abstract* and *concrete* semantics of programs, but also to elicit and highlight their semantic effects.

Considering that a program or a process is composed by individual statements with given rules of compositions, the definition and mathematical model of deductive semantics at the statement level can be extended onto the higher levels of program hierarchy.

Definition 12. *The semantics of a process P, θ(P), on a given semantic environment Θ is a double partial differential of the semantic function $f_\theta(\boldsymbol{P})$ on the sets of variables S and executing steps T, see Box 3, where $\mathbf{V}_{\mathbf{p}_k}$, 1≤ k ≤ n-1, is a set of values of local variables that belongs to processes P_k, and $\mathbf{V}_G$ is a finite set of values of global variables.*

On the basis of Definition 12, the semantics of a program at the top-level composition can be deduced to the combination of the semantics of a set of processes, each of which can be further deduced to the composition of all statements' semantics as described below.

Definition 13. *The semantics of a program ℘, θ(℘), on a given semantic environment Θ, is a combination of the semantic functions of all processes θ(P_k), 1≤ k ≤ n, i.e.:*

$$\begin{aligned}\theta(\wp) &= \mathop{R}_{k=1}^{\#K(\wp)} \frac{\partial^2}{\partial t\,\partial s} f_\theta(\wp) \\ &= \mathop{R}_{k=1}^{\#K(\wp)} \theta(P_k) \\ &= \mathop{R}_{k=1}^{\#K(\wp)} [\mathop{R}_{i=0}^{\#T(P_k)} (\mathop{R}_{j=1}^{\#S(P_k)} v_{P_k}(t_i, s_j))]\end{aligned} \tag{14}$$

where #*K*(℘) is the number of processes or components encompassed in the program.

It is noteworthy that Equation 14 will usually result in a very large matrix of semantic space,

Box 3.

$$\begin{aligned}\theta(P) &= \frac{\partial^2}{\partial t\,\partial s} f_\theta(P) \\ &= \mathop{R}_{k=1}^{n-1} \{[\frac{\partial^2}{\partial t\,\partial s} f_\theta(P_k)]\, r_{kl}\, [\frac{\partial^2}{\partial t\,\partial s} f_\theta(P_l)]\}, l = k+1 \\ &= \mathop{R}_{k=1}^{n-1} \{[\mathop{R}_{i=0}^{\#T(P_k)} \mathop{R}_{j=1}^{\#S(P_k)} v_{P_k}(t_i, s_j)]\, r_{kl}\, [\mathop{R}_{i=0}^{\#T(P_l)} \mathop{R}_{j=1}^{\#S(P_l)} v_{P_l}(t_i, s_j)]\} \\ &= \begin{pmatrix} \mathbf{V}_{\mathbf{p}_1} & & & \mathbf{V}_G \\ & \mathbf{V}_{\mathbf{p}_2} & & \mathbf{V}_G \\ & & \ddots & \vdots \\ & & \mathbf{V}_{\mathbf{p}_{n-1}} & \mathbf{V}_G \end{pmatrix}\end{aligned} \tag{13}$$

which can be quantitatively predicated as follows.

Definition 14. *The semantic space of a program $S_\Theta(\wp)$ is a product of $\#S(\wp)$ variables and $\#T(\wp)$ executing steps, i.e.:*

$$S_\Theta(\wp) = \#S(\wp) \bullet \#T(\wp) = \sum_{k=1}^{\#K(\wp)} \#S(\wp_k) \bullet \sum_{k=1}^{\#K(\wp)} \#T(\wp_k) \quad (15)$$

The semantic space of programs provides a useful measure of software complexity. Due to the tremendous size of the semantic space, both program composition and comprehension are innately a hard problem in terms of complexity and cognitive difficulty.

Properties of Software Semantics

Observing the formal definitions and mathematical models of deductive semantics developed in previous subsections, a number of common properties of software semantics may be elicited, which are useful for explaining the fundamental characteristics of software semantics.

One of the most interesting characteristics of program semantics is its invariance against different executing speeds as described in the following theorem.

Theorem 1. *The asynchronicity of program semantics states that the semantics of a relatively timed program is invariant with the changes of executing speed, as long as any absolute time constraint is met.*

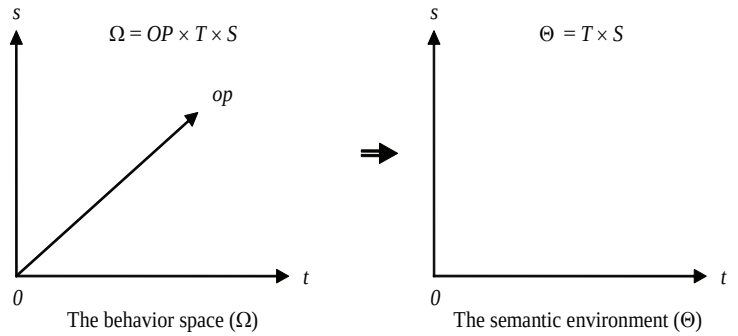
Theorem 1 asserts that, for most non real-time or relatively timed programs, different executing speeds or simulation paces will not alter the semantics of the software system. This explains why a programmer may simulate the run-time behaviors of a given program executing at a speed of up to 10^9 times faster than that of human beings. It also explains why computers with different system clock frequencies may correctly run the same program and obtain the same behavior.

Definition 15. *The behavior of a computational statement is a set of observable actions or changes of status of objects operated by the statement.*

According to Definition 4, the behavioral space of software, Ω, is three dimensional, while as given in Definition 7, the semantic environment Θ is two dimensional. Therefore, to a certain extent, semantic analysis is a projection of the 3-D software behaviors into the 2-D semantic environment Θ as shown in Figure 2.

The theory of deductive semantics can be systematically applied to formally and rigorously model and describe the semantics of the RTPA

Figure 2. Relationship between software behavior space and the semantic environment



metaprocesses and the *process relations* (operations). On the basis of the mathematical models and properties of deductive semantics, the following sections formally describe a comprehensive set of RTPA semantics, particularly the 17 metaprocesses and the 17 process relations (Wang, 2002, 2003, 2007c, 2008a, 2008b). This work extends the coverage of semantic rules of programming languages to a complete set of features that encompasses both basic computing operations and their algebraic composition rules. Because RTPA is a denotational mathematical structure based on process algebra that covers a comprehensive set of computing and programming requirements, any formal semantics that is capable to process RTPA is powerful enough to express the semantics of any programming language.

DEDUCTIVE SEMANTICS OF RTPA METAPROCESSES

Metaprocesses of RTPA are elicited from basic computational requirements. Complex processes can be composed with multiple metaprocesses. RTPA identified 17 metaprocesses, $\mathfrak{P}$, on fundamental computing operations such as assignment, system control, event/time handling, memory and I/O manipulation, i.e., $\mathfrak{P}$ = {:=, ◆, ⇒, ⇐, ⇏, ⩾, ⩽, |⩾, |⩽, @, ≙, ↑, ↓, !, ⊗, ⊠, §}. Detailed descriptions of the metaprocesses of RTPA and their syntaxes may be referred to (Wang, 2002, 2007c, 2008b), where each metaprocess is a basic operation on one or more operands such as variables, memory elements, or I/O ports. Based on Definitions 8 and 12, the deductive semantics of the set of RTPA metaprocesses can be defined in the following subsections.

The Assignment Process

Definition 16. *The semantics of the assignment process on a given semantic environment* Θ, $\theta(y\mathrm{RT} := x\mathrm{RT})$*, is a double partial differential of the semantic function* $f_\theta(y\mathrm{RT} := x\mathrm{RT})$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(y\mathrm{RT} := x\mathrm{RT}) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(y\mathrm{RT} := x\mathrm{RT}) \\ &= \mathop{R}_{i=0}^{\#T(y\mathrm{RT} := x\mathrm{RT})} \Big(\mathop{R}_{j=1}^{\#S(y\mathrm{RT} := x\mathrm{RT})} v\ t_i, s_j \Big) \\ &= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{2} v(t_i, s_j) \\ &= \begin{pmatrix} & x\mathrm{RT} & y\mathrm{RT} \\ \mathbf{t_0} & x\mathrm{RT} & \perp \\ \mathbf{(t_0, t_1]} & x\mathrm{RT} & x\mathrm{RT} \end{pmatrix}\end{aligned} \tag{16}$$

where the size of the matrix is #T • #S.

The Evaluation Process

Definition 17. *The semantics of the evaluation process on* Θ, $\theta(◆exp\mathbb{T} \to \mathbb{T})$*, is a double partial differential of the semantic function* $f_\theta(\theta(◆exp\mathbb{T} \to \mathbb{T})$ *on the sets of variables S and executing steps T in the following two forms, i.e.:*

$$\begin{aligned}\theta(◆(exp\mathrm{BL}) \to \mathrm{BL}) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(◆(exp\mathrm{BL}) \to \mathrm{BL}) \\ &= \mathop{R}_{i=0}^{\#T(◆exp\mathrm{BL}\to\mathrm{BL})} \mathop{R}_{j=1}^{\#S(◆exp\mathrm{BL}\to\mathrm{L})} v(t_i, s_j) \\ &= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{2} (v\ t_i, s_j) \\ &= \begin{pmatrix} & exp\mathrm{B} & ◆(exp\mathrm{BL})\mathrm{BL} \\ \mathbf{(t_0, t_1]} & \delta(exp\mathrm{BL})\mathrm{BL} & \perp \\ \mathbf{(t_1, t_2]} & \mathsf{T} & \mathsf{T} \\ \mathbf{(t_1, t_{2^-}]} & \mathsf{F} & \mathsf{F} \end{pmatrix}\end{aligned} \tag{17a}$$

or

$$\theta(\blacklozenge exp\mathbb{T} \to \mathbb{T}) \triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\blacklozenge exp\mathbb{T} \to \mathbb{T})$$
$$= \overset{\#T(\blacklozenge exp\mathbb{T}\to\mathbb{T})}{\underset{i=0}{R}} \left(\overset{\#S(\blacklozenge exp\mathbb{T}\to\mathbb{T})}{\underset{j=1}{R}} v\ t_i, s_j \right)$$
$$= \overset{1}{\underset{i=0}{R}} \overset{2}{\underset{j=1}{R}} v(t_i, s_j)$$
$$= \begin{pmatrix} & exp\mathbb{T} & \blacklozenge(exp\mathbb{T})\mathbb{T} \\ \mathbf{(t_0, t_1]} & \delta(exp\mathbb{T})\mathbb{T} & \bot \\ \mathbf{(t_1, t_2]} & n\mathbb{T} & n\mathbb{T} \end{pmatrix}$$

(17b)

*where ◆(exp*BL*) is the Boolean evaluation function on expBL that results in* T *or* F*. ◆(exp*$\mathbb{T}$*) is a more general cardinal or numerical evaluation function on exp*$\mathbb{T}$ *that results in* $\mathbb{T}$ *= {*N*,* Z*,* R*,* B*}, i.e., in types of nature number, integer, real number, and byte, respectively (Wang, 2002).*

The Addressing Process

Definition 18. *The semantics of the addressing process on* Θ*,* θ*(id*S ⇒ *ptr*P*), is a double partial differential of the semantic function* f_θ*(id*S ⇒ *ptr*P*) on the sets of variables S and executing steps T, i.e.:*

$$\theta(id\mathrm{S} \Rightarrow ptr\mathrm{P}) \triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\mathrm{idS} \Rightarrow \mathrm{ptrP})$$
$$= \overset{\#T(\mathrm{idS} \Rightarrow \mathrm{trP})}{\underset{i=0}{R}} \left(\overset{\#S(\mathrm{idS} \Rightarrow \mathrm{ptrP})}{\underset{j=1}{R}} v\ t_i, s_j \right)$$
$$= \overset{1}{\underset{i=0}{R}} \overset{2}{\underset{j=1}{R}} v(t_i, s_j)$$
$$= \begin{pmatrix} & \mathbf{id}\mathrm{S} & \mathbf{ptr}\mathrm{P} \\ \mathbf{t_0} & id\mathrm{S} & \bot \\ \mathbf{(t_0, t_1]} & id\mathrm{S} & \pi(id\mathrm{S})\mathrm{H} \end{pmatrix}$$

(18)

where π*(id*S*)*H *is a function that associates a declared identifier id*S *to its hexadecimal memory address located by the pointed ptr*P*.*

The Memory Allocation Process

Definition 19. *The semantics of the memory allocation process on* Θ*,* θ*(id*S ⇐ *MEM(ptr*P*)*RT*), is a double partial differential of the semantic function* f_θ*(id*S ⇐ *MEM(ptr*P*)*RT*) on the sets of variables S and executing steps T, see Box 4. Where* π*(id*S*)*H *is a mapping function that associates an identifier id*S *to a memory block starting at a hexadecimal address located by the pointed ptr*P*. The ending address of the allocated memory block, ptr*P*+size(*RT*)-1, is dependent on a machine implementation of the size of a given variable in type* RT*.*

The Memory Release Process

Definition 20. *The semantics of the memory release process on* Θ*,* θ*(id*S ⇍ *MEM(*⊥*)*RT*), is a double partial differential of the semantic function* f_θ*(id*S ⇍ *MEM(*⊥*)*RT*) on the sets of variables S and executing steps T, i.e.:*

$$\theta(id\mathrm{S} \nLeftarrow \mathrm{MEM}[\ \bot\ ]\mathrm{RT}) \triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(id\mathrm{S} \nLeftarrow \mathrm{MEM}[\ \bot\ ]\mathrm{RT})$$
$$= \overset{\#T(id\mathrm{S} \nLeftarrow \mathrm{MEM}[\bot]\mathrm{RT})}{\underset{i=0}{R}} \left(\overset{\#S(id\mathrm{S} \nLeftarrow \mathrm{MEM}[\bot]\mathrm{RT})}{\underset{j=1}{R}} v\ t_i, s_j \right)$$
$$= \overset{1}{\underset{i=0}{R}} \overset{3}{\underset{j=1}{R}} v(t_i, s_j)$$
$$= \begin{pmatrix} & \mathbf{id}\mathrm{RT} & \mathbf{ptr}\mathrm{P} & \mathbf{MEM}\mathrm{RT} \\ \mathbf{t_0} & (id\mathrm{S}) & \pi(id\mathrm{S})\mathrm{H} & \mathrm{MEM}(ptr\mathrm{P})\mathrm{RT} \\ \mathbf{(t_0, t_1]} & \bot & \bot & \bot \end{pmatrix}$$

(20)

The Read Process

Definition 21. *The semantics of the read process on* Θ*,* θ*(MEM(ptr*P*)*RT*⩾x*RT*), is a double partial differential of the semantic function* f_θ*(MEM(ptr*P*)*RT*⩾x*RT*) on the sets of variables S and executing steps T, see Box 5.*

Box 4.

$$\theta(\text{idS} \Leftarrow \text{MEM[ptrP]RT}) \triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\text{idS} \Leftarrow \text{MEM[ptrP]RT})$$
$$= \mathop{R}_{i=0}^{\#T(\text{idS} \Leftarrow \text{MEM[ptrP]RT})} \left(\mathop{R}_{j=1}^{\#S(\text{idS} \Leftarrow \text{MEM[ptrP]RT})} v\ t_i, s_j \right)$$
$$= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j)$$
$$= \begin{pmatrix} & \textbf{id}\text{S} & \textbf{ptr}\text{P} & \textbf{MEM}\text{RT} \\ \mathbf{t_0} & id\text{S} & \perp & \perp \\ \mathbf{(t_0, t_1]} & id\text{S} & \pi(id\text{S})\text{H} & \text{MEM}[ptr\text{P}]\text{RT} \end{pmatrix} \tag{19}$$

Box 5.

$$\theta(\text{MEM[ptrP]RT} \gg \text{xRT})$$
$$\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\text{MEM[ptrP]RT} \gg \text{xRT})$$
$$= \mathop{R}_{i=0}^{\#T(\text{MEM[ptrP]RT} \gg \text{xRT})} \left(\mathop{R}_{j=1}^{\#S(\text{MEM[ptrP]RT} \gg \text{xRT})} v\ t_i, s_j \right)$$
$$= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j)$$
$$= \begin{pmatrix} & \textbf{ptr}\text{P} & \textbf{MEM(ptr)}\text{P}\text{RT} & \textbf{x}\text{RT} \\ \mathbf{t_0} & ptr\text{P} & \perp & \perp \\ \mathbf{(t_0, t_1]} & ptr\text{P} & \text{MEM}[ptr\text{P}]\text{RT} & \text{MEM}[ptr\text{P}]\text{RT} \end{pmatrix} \tag{21}$$

The Write Process

Definition 22. *The semantics of the write process on* Θ, θ*(MEM(ptr*P*)*RT≪*x*RT*), is a double partial differential of the semantic function* f_θ*(MEM(ptr*P*)*RT≪*x*RT*) on the sets of variables S and executing steps T, i.e.:*

$$\theta(\text{MEM[ptrP]RT} \ll \text{xRT})$$
$$\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\text{MEM[ptrP]RT} \ll \text{xRT})$$
$$= \mathop{R}_{i=0}^{\#T(\text{MEM[ptrP]RT} \ll \text{xRT})} \left(\mathop{R}_{j=1}^{\#S(\text{MEM[ptrP]RT} \quad \text{xRT})} v\ t_i, s_j \right)$$
$$= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j)$$
$$= \begin{pmatrix} & \textbf{x}\text{RT} & \textbf{ptr}\text{P} & \textbf{MEM[ptr]}\text{RT} \\ \mathbf{t_0} & x\text{RT} & \perp & \perp \\ \mathbf{(t_0, t_1]} & x\text{RT} & ptr\text{P} & x\text{RT} \end{pmatrix} \tag{22}$$

The Input Process

Definition 23. *The semantics of the input process on* Θ, θ*(PORT(ptr*P*)*RT$|\gg x$RT*), is a double partial differential of the semantic function* f_θ*((PORT(ptr*P*)*RT$|\gg x$RT*) on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
&\theta(\text{PORT}[\text{ptr}^{\text{P}}]\text{RT} \,|\gg \text{xRT}) \\
&\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(\text{PORT}[\text{ptr}^{\text{P}}]\text{RT} \,|\gg \text{xRT}) \\
&= \mathop{R}_{i=0}^{\#T(\text{PORT}[\text{ptr}^{\text{P}}]\text{RT}|\gg \text{xRT})} \Big(\mathop{R}_{j=1}^{\#S(\text{PORT}[\text{ptr}^{\text{P}}]\text{RT}|\ \text{xRT})} v\ t_i, s_j \Big) \\
&= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j) \\
&= \begin{pmatrix} & \textbf{ptr}^{\text{P}} & \textbf{PORT[ptr}^{\text{P}}\textbf{]RT} & \textbf{x}\text{RT} \\ \mathbf{t_0} & ptr^{\text{P}} & \perp & \perp \\ \mathbf{(t_0, t_1]} & ptr^{\text{P}} & \text{PORT}[ptr^{\text{P}}]\text{RT} & \text{PORT}[ptr^{\text{P}}]\text{RT} \end{pmatrix}
\end{aligned}
\tag{23}
$$

The Output Process

Definition 24. *The semantics of the output process on* Θ, θ*(x*RT $|\ll$ *PORT(ptr*P*)*RT*), is a double partial differential of the semantic function* f_θ*(x*RT $|\ll$ *PORT(ptr*P*)*RT*) on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
&\theta(\text{xRT} \,|\ll \text{PORT}[\text{ptr}^{\text{P}}]\text{RT}) \\
&\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(\text{xRT} \,|\ll \text{PORT}[\text{ptr}^{\text{P}}]\text{RT}) \\
&= \mathop{R}_{i=0}^{\#T(\text{xRT}|\ll \text{PORT}[\text{ptr}^{\text{P}}]\text{RT})} \Big(\mathop{R}_{j=1}^{\#S(\text{xRT}|\ll \text{PORT}[\text{ptr}^{\text{P}}]\text{RT})} v\ t_i, s_j \Big) \\
&= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j) \\
&= \begin{pmatrix} & \textbf{x}\text{RT} & \textbf{ptr}^{\text{P}} & \textbf{PORT[ptr}^{\text{P}}\textbf{]RT} \\ \mathbf{t_0} & x\text{RT} & \perp & \perp \\ \mathbf{(t_0, t_1]} & x\text{RT} & ptr^{\text{P}} & x\text{RT} \end{pmatrix}
\end{aligned}
\tag{24}
$$

The Timing Process

Definition 25. *The semantics of the timing process on* Θ, θ*(@t*TM $\hat{=}$ *§t*TM*), is a double partial differential of the semantic function* f_θ*(@t*TM $\hat{=}$ *§t*TM*) on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
\theta(@\,\text{tTM} \hat{=} \S\text{t M}) &\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(@\,\text{tTM} \hat{=} \S\text{tTM}) \\
&= \mathop{R}_{i=0}^{\#T(@\,\text{tTM} \hat{=} \S\text{t M})} \Big(\mathop{R}_{j=1}^{\#S(@\,\text{tTM} \hat{=} \S\text{tTM})} v\ t_i, s_j \Big) \\
&= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{2} v(t_i, s_j) \\
&= \begin{pmatrix} & \textbf{§t}\text{TM} & \textbf{@t}\text{TM} \\ \mathbf{t_0} & \S\text{tTM} & \perp \\ \mathbf{(t_0, t_1]} & \S\text{tTM} & \S\text{tTM} \end{pmatrix}
\end{aligned}
\tag{25}
$$

where TM *represents the three timing types, i.e.,* TM = *{*yy:MM:dd, hh:mm:ss:ms, yy:MM:dd:hh:mm:ss:ms*}.*

The Duration Process

Definition 26. *The semantics of the duration process on* Θ, θ*(@t*TM$\triangleq$*§t*TM+Δ*dZ), is a double partial differential of the semantic function* f_θ*(@t*TM$\triangleq$*§t*TM+Δ*dN) on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
&\theta(@\,\text{tTM} \triangleq \S\text{t M}+\Delta\text{dZ}) \\
&\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(@\,\text{tTM} \triangleq \S\text{t M}+\Delta\text{dZ}) \\
&= \mathop{R}_{i=0}^{\#T(@\,\text{tTM} \triangleq \S\text{t M})} \Big(\mathop{R}_{j=1}^{\#S(@\,\text{tTM} \triangleq \S\text{tTM})} v\ t_i, s_j \Big) \\
&= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j) \\
&= \begin{pmatrix} & \textbf{§t}\text{TM} & \Delta\textbf{d}\text{N} & \textbf{@t}\text{TM} \\ \mathbf{t_0} & \S\text{tTM} & \Delta\text{dN} & \perp \\ \mathbf{(t_0, t_1]} & \S\text{tTM} & \Delta\text{dN} & \S\text{tTM}+\Delta d\text{N} \end{pmatrix}
\end{aligned}
\tag{26}
$$

where TM = {yy:MM:dd, hh:mm:ss:ms, yy:MM:dd: hh:mm:ss:ms}.

The Increase Process

Definition 27. *The semantics of the increase process on* Θ, $\theta(\uparrow(x\mathrm{RT}))$, *is a double partial differential of the semantic function* $f_\theta(\uparrow(x\mathrm{RT}))$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(\uparrow(\mathrm{xRT})) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(\uparrow(\mathrm{xRT}))\\ &= \mathop{R}_{i=0}^{\#T(\uparrow(\mathrm{xRT}))} \mathop{R}_{j=1}^{\#S(\uparrow(\mathrm{xRT}))} v(t_i, s_j)\\ &= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{1} (\nu\ t_i, s_j)\\ &= \begin{pmatrix} & \mathbf{x}\mathrm{RT} \\ \mathbf{t_0} & x\mathrm{RT} \\ \mathbf{(t_0, t_1]} & x\mathrm{RT}+1 \end{pmatrix}\end{aligned} \tag{27}$$

where the run-time type RT = {N, Z, B, H, P, TM}

The Decrease Process

Definition 28. *The semantics of the decrease process on* Θ, $\theta(\downarrow(x\mathrm{RT}))$, *is a double partial differential of the semantic function* $f_\theta(\downarrow(x\mathrm{RT}))$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(\downarrow(\mathrm{xRT})) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(\downarrow(\mathrm{xRT}))\\ &= \mathop{R}_{i=0}^{\#T(\downarrow(\mathrm{xRT}))} \mathop{R}_{j=1}^{\#S(\downarrow(\mathrm{xRT}))} v(t_i, s_j)\\ &= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{1} (\nu\ t_i, s_j)\\ &= \begin{pmatrix} & \mathbf{x}\mathrm{RT} \\ \mathbf{t_0} & x\mathrm{RT} \\ \mathbf{(t_0, t_1]} & x\mathrm{RT}-1 \end{pmatrix}\end{aligned} \tag{28}$$

where the run-time type RT = {N, Z, B, H, P, TM}

The Exception Detection Process

Definition 29. *The semantics of the exception detection process on* Θ, $\theta(!(@)e\mathrm{S})$, *is a double partial differential of the semantic function* $f_\theta(!(@)e\mathrm{S}))$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(!(@e\mathrm{S})) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(!(@e\mathrm{S}))\\ &= \mathop{R}_{i=0}^{\#T(!(@e\mathrm{S}))} \mathop{R}_{j=1}^{\#S(!(@e\mathrm{S}))} (\nu\ t_i, s_j)\\ &= \mathop{R}_{i=0}^{1} \mathop{R}_{j=1}^{3} v(t_i, s_j)\\ &= \begin{pmatrix} & \mathbf{@e}\mathrm{S} & \mathbf{ptr}\mathrm{P} & \mathbf{PORT(ptr}\mathrm{P}\mathbf{)}\mathrm{S} \\ \mathbf{t_0} & @e\mathrm{S} & \perp & \perp \\ \mathbf{(t_0, t_1]} & @e\mathrm{S} & ptr\mathrm{P} & @e\mathrm{S} \end{pmatrix}\end{aligned} \tag{29}$$

Equation 29 indicates that the semantics of exception detection is the output of a string @*e*S to a designated port PORT[*ptr*P]S, where the pointer *ptr*P points to a CRT or a printer. Therefore, the semantics of exception detection can be described based on the semantics of the output process as defined in Equation 24, i.e.:

$$\theta((!(@e\mathrm{S})) = \theta(@e\mathrm{S} \mid\!\prec \mathrm{PORT}[ptr\mathrm{P}]\mathrm{S}) \tag{30}$$

The Skip Process

Definition 30. *The semantics of the skip process on* Θ, $\theta(\otimes)$, *is a double partial differential of the semantic function* $f_\theta(\otimes)$ *on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
\theta(\otimes) &\triangleq \theta(P^k \curvearrowright P^{k-1}) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P^k \curvearrowright P^{k-1}) \\
&= \overset{\#T(P^k \curvearrowright P^{k-1})}{\underset{i=0}{R}} \; (\overset{\#S(P^k \curvearrowright P^{k-1})}{\underset{j=1}{R}} \; v\; t_i, s_j) \\
&= \overset{1}{\underset{i=0}{R}} \overset{2}{\underset{j=1}{R}} v(t_i, s_j) \\
&= \begin{pmatrix} & \mathbf{S_{P^{k-1}}} & \mathbf{S_{P^k}} \\ \mathbf{t_0} & S_{P^{k-1}} & S_{P^k} \\ \mathbf{(t_0, t_1]} & S_{P^{k-1}} \setminus S_{P^k} & \perp \end{pmatrix}
\end{aligned}
\tag{31}
$$

where P^k is a process P at a given embedded layer k in a program with P^0 at the uttermost layer, and $\curvearrowright$ denotes the jump process relation where its semantics will be formally defined in the next section.

According to Definition 30, the skip process $\otimes$ has no semantic effect on the current process P^k at the given embedded layer k in a program, such as a branch, loop, or function. However, it redirects the system to jump to execute an upper-layer process P^{k-1} in the embedded hierarchy. Therefore, skip is also known as *exit* or *break* in programming languages.

The Stop Process

Definition 31. *The semantics of the stop process on Θ, θ ($\oiint$), is a double partial differential of the semantic function $f_\theta(\boxtimes)$ on the sets of variables S and executing steps T, i.e.:*

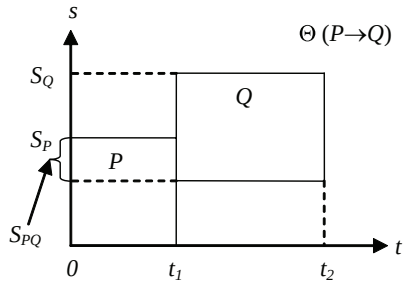
$$
\begin{aligned}
\theta(\boxtimes) &\triangleq \theta(P \curvearrowright \S) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \curvearrowright \S) \\
&= \overset{\#T(P \curvearrowright \S)}{\underset{i=0}{R}} \; (\overset{\#S(P \curvearrowright \S)}{\underset{j=1}{R}} \; v\; t_i, s_j) \\
&= \overset{1}{\underset{i=0}{R}} \overset{2}{\underset{j=1}{R}} v(t_i, s_j) \\
&= \begin{pmatrix} & \mathbf{S_\S} & \mathbf{S_P} \\ \mathbf{t_0} & S_\S & S_P \\ \mathbf{(t_0, t_1]} & S_\S \setminus S_P & \perp \end{pmatrix}
\end{aligned}
\tag{32}
$$

where the stop process $\boxtimes$ does nothing but returns the control of execution to the system.

DEDUCTIVE SEMANTICS OF RTPA PROCESS RELATIONS

The preceding section provides formal definitions of metaprocesses of RTPA for software system modeling. Via the composition of multiple metaprocesses by the 17 process relations, R = {$\rightarrow$, $\curvearrowright$, |, |…|…, R^*, R^+, R^i, $\circlearrowleft$, $\rightarrowtail$, ||, $\oiint$, |||, », $\lightning$, $\hookrightarrow_t$, $\hookrightarrow_e$, $\hookrightarrow_i$}, complex architectures and behaviors of software systems, in the most complicated case, a real-time system, can be sufficiently described (Wang, 2002, 2006a, 2007c, 2008b). On the basis of Definitions 8 and 12, the semantics of the RTPA process relations can be formally defined and analyzed as follows.

Figure 3. The semantic diagram of the sequential process relation



The Sequential Process Relation

Definition 32. *The semantics of the sequential relation of processes on Θ, θ(P→Q), is a double partial differential of the semantic function $f_\theta(P \rightarrow Q)$ on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}
\theta(P \rightarrow Q) &\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \rightarrow Q) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) \rightarrow \frac{\partial^2}{\partial t\, \partial s} f_\theta(Q) \\
&= \overset{\#T(P)}{\underset{i=0}{R}} \left(\overset{\#S(P)}{\underset{j=1}{R}} v_P\ t_i, s_j \right) \rightarrow \overset{\#T(Q)}{\underset{i=0}{R}} \left(\overset{\#S(Q)}{\underset{j=1}{R}} v_Q\ t_i, s_j \right) \\
&= \overset{\#T(\widehat{PQ})}{\underset{i=0}{R}} \left(\overset{\#S(P \cup Q)}{\underset{j=1}{R}} v\ t_i, s_j \right) \\
&= \begin{pmatrix} & \mathbf{s_P} & \mathbf{s_Q} & \mathbf{s_{PQ}} \\ \mathbf{t_0} & \perp & \perp & \perp \\ \mathbf{(t_0, t_1]} & V_{1P} & - & V_{1PQ} \\ \mathbf{(t_1, t_2]} & - & V_{2Q} & V_{2PQ} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{V_P} & & \mathbf{V_{PQ}} \\ & \mathbf{V_Q} & \mathbf{V_{PQ}} \end{pmatrix}
\end{aligned} \tag{33}$$

where $P^\frown Q$ indicates a concatenation of these two processes over time, and in the simplified notation of the matrix, $V_P = v(t_P, s_P)$, $0 \le t_P \le n_P$, $1 \le s_P \le$

Box 6.

$$\begin{aligned}
\theta(P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_5) &= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_5) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P_0) \rightarrow \frac{\partial^2}{\partial t\, \partial s} f_\theta(P_1) \rightarrow \ldots \rightarrow \frac{\partial^2}{\partial t\, \partial s} f_\theta(P_5) \\
&= \overset{\#T(P_0)}{\underset{i=0}{R}} \left(\overset{\#S(P_0)}{\underset{j=1}{R}} v_{P_0}\ t_i, s_j \right) \rightarrow \overset{\#T(P_1)}{\underset{i=0}{R}} \left(\overset{\#S(P_1)}{\underset{j=1}{R}} v_{P_1}\ t_i, s_j \right) \rightarrow \ldots \rightarrow \overset{\#T(P_5)}{\underset{i=0}{R}} \overset{\#S(P_5)}{\underset{j=1}{R}} v_{P_5}(t_i, s_j) \\
&= \overset{\#T(\widehat{P_0 P_1 \ldots P_5})}{\underset{i=0}{R}} \left(\underset{j=1}{\overset{\#S(P_0 \cup P_1 \cup \ldots \cup P_5)}{R}} v\ t_i, s_j \right) \\
&= \overset{5}{\underset{i=0}{R}} \overset{4}{\underset{j=1}{R}} v(t_i, s_j) \\
&= \begin{pmatrix} & \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{PORT[CRTP]N} \\ \mathbf{t_0} & \perp & \perp & \perp & \perp \\ \mathbf{(t_0, t_1]} & 2 & \perp & \perp & \perp \\ \mathbf{(t_1, t_2]} & 2 & 8 & \perp & \perp \\ \mathbf{(t_2, t_3]} & 2 & 8 & 10 & \perp \\ \mathbf{(t_3, t_4]} & 2 & 8 & 20 & \perp \\ \mathbf{(t_4, t_5]} & 2 & 8 & 20 & 20 \end{pmatrix}
\end{aligned} \tag{34}$$

m_P; $V_Q = v(t_Q, s_Q)$, $0 \le t_Q \le n_Q$, $1 \le s_Q \le m_Q$; *and* $V_{PQ} = v(t_{PQ}, s_{PQ})$, $0 \le t_{PQ} \le n_{PQ}$, $1 \le s_{PQ} \le m_{PQ}$.

In Equation 33, the first partial differential selects a set of related variables in the sequential processes P and Q, $S(P \cup Q)$. The second partial differential selects a set of time moments $T(P \frown Q)$. The semantic diagram of the sequential process relation as defined in Equation 33 is illustrated in Figure 3 on Θ.

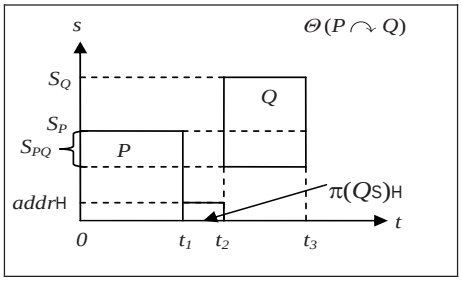
The following example shows the physical meaning of Equation 33 and how the abstract syntaxes and their implied meanings are embodied onto the objects (variables) and their dynamic values in order to obtain the concrete semantics in deductive semantics.

Example 3. *Analyze the semantics of the sequential processes P_0 through P_5 in the following program:*

```
void sequential_sum;
  {
     int x, y, z;          // P0
     x = 2;                // P1
     y = 8;                // P2
     z := x + y;           // P3
     z := x + y + z;       // P4
     print z;              // P5
  }
```

According to Definition 32, the semantics of the above program can be analyzed as seen in Box 6.

Figure 4. The semantic diagram of the jump process relation



Where PORT[*CRT*P]N denotes a system monitor of type N located by the pointer *CRT*P.

The Jump Process Relation

Definition 33. *The semantics of the jump relations of processes on Θ, θ(P ↷ Q), is a double partial differential of the semantic function $f_\theta(P \curvearrowright Q)$ on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
&\theta(P \curvearrowright Q) \triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \curvearrowright Q) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) (\curvearrowright \frac{\partial^2}{\partial t\, \partial s} f_\theta\ Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j) \curvearrowright \mathop{R}_{i=0}^{\#T(Q)} (\mathop{R}_{j=1}^{\#S(Q)} v_Q\ t_i, s_j) \\
&= \mathop{R}_{i=0}^{\#T(\widehat{PQ})} (\mathop{R}_{j=1}^{\#S(P \cup Q)} v\ t_i, s_j) \\
&= \begin{pmatrix}
 & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} & \mathbf{addr}\mathsf{H} \\
\mathbf{[t_0, t_1]} & V_{1P} & \perp & V_{1PQ} & \perp \\
\mathbf{(t_1, t_2]} & - & - & - & \pi(QS)\mathsf{H} \\
\mathbf{(t_2, t_3]} & - & V_{3Q} & V_{3PQ} &
\end{pmatrix}
\end{aligned}
\tag{35}
$$

*where π(QS)H is a system addressing function of the system that directs the program control flow to execute the new process Q, which physically located in a different memory address at addr*H *= π(QS)H.*

The semantic diagram of the jump process relation as defined in Equation 35 is illustrated in Figure 4 on $\Theta(P \curvearrowright Q)$.

The jump process relation is an important process relation that forms a fundamental part of many other processes and constructs. For instances, the jump process relation has been applied in expressing the semantics of the *skip* and *stop* processes in the preceding section.

Box 7.

$$\theta(\text{exp}_{\text{RT}} \rightarrow P \mid \sim \rightarrow Q) \triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(\text{exp}_{\text{RT}} \rightarrow P \mid \sim \rightarrow Q)$$

$$= \quad \text{exp}_{\text{BL}} \rightarrow \frac{\partial^2}{\partial t\, \partial s} f_\theta(P)$$

$$\mid \quad \rightarrow \frac{\partial^2}{\partial t\, \partial s} f_\theta(Q)$$

$$= \quad \text{exp}_{\text{BL}} \rightarrow \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j)$$

$$\mid \quad \rightarrow \mathop{R}_{i=0}^{\#T(Q)} (\mathop{R}_{j=1}^{\#S(Q)} v_Q\ t_i, s_j)$$

$$= \begin{pmatrix} & \textbf{exp}_{\text{BL}} & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} \\ \mathbf{(t_0, t_1]} & \delta(\text{exp}_{\text{BL}}) & \perp & \perp & \perp \\ \mathbf{(t_1, t_2]} & \mathsf{T} & V_{2P} & - & V_{2PQ} \\ \mathbf{(t_1, t_{2'}]} & \mathsf{F} & - & V_{3Q} & V_{3PQ} \end{pmatrix}$$

(36)

Figure 5. The semantic diagram of the branch process relation

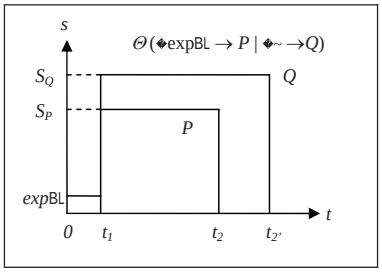
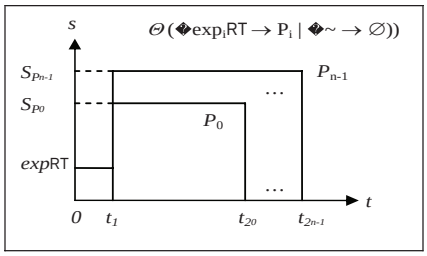


Figure 6. The semantic diagram of the switch process relation



The Branch Process Relation

Definition 34. *The semantics of the branch relation of processes on* Θ, $\theta(\text{exp}_{\text{BL}} = \mathsf{T} \rightarrow P \mid \sim \rightarrow Q)$, *abbreviated by* $\theta(P|Q)$, *is a double partial differential of the semantic function* $f_\theta(P|Q)$ *on the sets of variables S and executing steps T, see Box 7, where* $\delta(\text{exp}_{\text{BL}})$ *is the evaluation function on the value of* exp_{BL}, $\delta(\text{exp}_{\text{BL}}) \in \{\mathsf{T}, \mathsf{F}\}$.

The semantic diagram of the branch process relation as defined in Equation 34 is illustrated in Figure 5 on $\Theta(\text{exp}_{\text{BL}} \rightarrow P \mid \neg\, \text{exp}_{\text{BL}} \rightarrow Q)$.

The Switch Process Relation

Definition 35. *The semantics of the switch relations of processes on* Θ, $\theta(\text{exp}_i\text{RT} \rightarrow P_i \mid \sim \rightarrow \varnothing)$, *abbreviated by* $\theta(P_i \mid \varnothing)$, *is a double partial differential of the semantic function* $f_\theta(P_i \mid \varnothing)$ *on the sets of variables S and executing steps T, i.e.:*

$$
\begin{aligned}
&\theta(\text{◆}exp_i\mathsf{RT} \rightarrow \mathrm{P_i} \mid \text{◆} \quad \rightarrow \otimes) \\
&\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\text{◆}exp_i\mathsf{RT} \rightarrow \mathrm{P_i} \mid \text{◆} \quad \rightarrow \otimes) \\
&= \ \text{◆}exp\mathsf{RT} = 0 \rightarrow \frac{\partial^2}{\partial t\ \partial s} f_\theta(P_0) \\
&\quad \mid ... \\
&\quad \mid \text{◆}exp\mathsf{RT} = n-1 \rightarrow \frac{\partial^2}{\partial t\ \partial s} f_\theta(P_{n-1}) \\
&\quad \mid \text{◆}exp\mathsf{RT} = n \rightarrow \frac{\partial^2}{\partial t\ \partial s} f_\theta(\otimes) \\
&= \ \text{◆}exp\mathsf{RT} = 0 \rightarrow \overset{\#T(P_0)}{\underset{i=0}{R}} \ \overset{\#S(P_0)}{\underset{j=1}{R}} v_{P_0}(t_i, s_j) \\
&\quad \mid ... \\
&\quad \mid \text{◆}exp\mathsf{RT} = n-1 \rightarrow \overset{\#T(P_{n-1})}{\underset{i=0}{R}} \ \overset{\#S(P_{n-1})}{\underset{j=1}{R}} v_{P_{n-1}}(t_i, s_j) \\
&\quad \mid \text{◆}exp\mathsf{RT} = n \rightarrow \varnothing \\
&= \begin{pmatrix}
 & \mathbf{exp}\mathsf{RT} & \mathbf{S_{P_0}} & \cdots & \mathbf{S_{P_{n-1}}} & \mathbf{S_G} \\
\mathbf{[t_0, t_1]} & \delta(\mathrm{expRT}) & \perp & \cdots & \perp & \perp \\
\mathbf{(t_1, t_{2_0}]} & 0 & V_{2_0 P} & \cdots & - & V_G \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\mathbf{(t_1, t_{2_{n-1}}]} & n-1 & - & \cdots & V_{2_{n-1} P} & V_G \\
\mathbf{(t_1, t_{2_n}]} & n & - & \cdots & - & V_G
\end{pmatrix}
\end{aligned}
\tag{37}
$$

where V_G is a set of global variables shared by P_0, P_1, and P_{n-1}.

The semantic diagram of the switch process relation as defined in Equation 37 is illustrated in Figure 6 on Θ(◆exp_iRT $\rightarrow$ $\mathrm{P_i}$ | ◆ ~ $\rightarrow \varnothing$).

The While-Loop Process Relation

Definition 36. *The semantics of the while-loop relations of processes on* Θ*,*

$$\theta(\overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(P)),$$

is a double partial differential of the semantic function

$$f_\theta(\overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(P))$$

on the sets of variables S and executing steps T, i.e.:

$$
\begin{aligned}
\theta(\overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(P)) &\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(\overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(P)) \\
&= \overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(\frac{\partial^2}{\partial t\ \partial s} f_\theta(P)) \\
&= \overset{\mathsf{F}}{\underset{\mathrm{expBL}=\mathsf{T}}{R}}{}^{*}(\overset{\#T(P)}{\underset{i=0}{R}} \ \overset{\#S(P)}{\underset{j=1}{R}} v_P(t_i, s_j)) \\
&= \begin{pmatrix}
 & \mathbf{exp}\mathsf{BL} & \mathbf{S_P} \\
\mathbf{[t_0, t_1]} & \delta(\mathrm{expBL}) & \perp \\
\mathbf{(t_1, t_2]} & \mathsf{T} & V_P \\
\mathbf{(t_1, t_{2'}]} & \mathsf{F} & \otimes \\
\vdots & \vdots & \vdots \\
\mathbf{(t_3, t_4]} & \delta(\mathrm{expBL}) & - \\
\mathbf{(t_4, t_5]} & \mathsf{T} & V_P \\
\mathbf{(t_4, t_{5'}]} & \mathsf{F} & \otimes
\end{pmatrix}
\end{aligned}
\tag{38}
$$

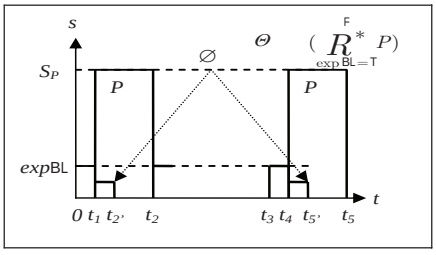
*where $\varnothing$ denotes exit, and δ(exp*BL*) is the evaluation function on the Boolean expression, δ(exp*BL*) $\in$ {T, F}.*

The semantic diagram of the while-loop process relation as defined in Equation 38 is illustrated in Figure 7 on Θ.

The Repeat-Loop Process Relation

Definition 37. *The semantics of the repeat-loop relations of processes on* Θ*,*

Figure 7. The semantic diagram of the while-loop process relation



$\theta(\overset{F}{R}{}^{+}_{\text{expBL=T}}(P))$,

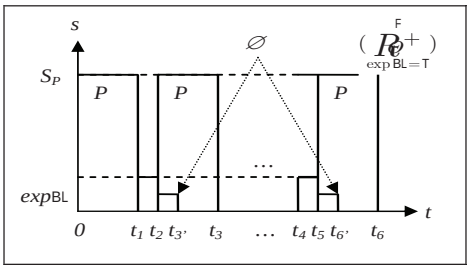
is a double partial differential of the semantic function

$f_\theta(\overset{F}{R}{}^{+}_{\text{expBL=T}}(P))$

on the sets of variables S and executing steps T, i.e.:

$$\theta(\overset{F}{R}{}^{+}_{\text{expBL=T}}(P)) \triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(\overset{F}{R}{}^{+}_{\text{expBL=T}}(P))$$
$$= \overset{F}{R}{}^{+}_{\text{expBL=T}}(\frac{\partial^2}{\partial t\,\partial s} f_\theta(P))$$
$$= P \to \overset{F}{R}{}^{*}_{\text{expBL=T}}(\overset{\#T(P)}{\underset{i=0}{R}}\ \overset{\#S(P)}{\underset{j=1}{R}} v_P(t_i, s_j))$$
$$= \begin{pmatrix} & \textbf{exp}\,\mathsf{BL} & \mathbf{S_P} \\ \mathbf{[t_0, t_1]} & \perp & V_P \\ \mathbf{(t_1, t_2]} & \delta(\text{expBL}) & - \\ \mathbf{(t_2, t_3]} & \mathsf{T} & V_P \\ \mathbf{(t_2, t_{3'}]} & \mathsf{F} & \otimes \\ \vdots & \vdots & \vdots \\ \mathbf{(t_4, t_5]} & \delta(\text{expBL}) & - \\ \mathbf{(t_5, t_6]} & \mathsf{T} & V_P \\ \mathbf{(t_5, t_{6'}]} & \mathsf{F} & \otimes \end{pmatrix} \quad (39)$$

Figure 8. The semantic diagram of the repeat-loop process relation



The semantic diagram of the repeat-loop process relation as defined in Equation 39 is illustrated in Figure 8 on Θ .

The For-Loop Process Relation

Definition 38. *The semantics of the for-loop relations of processes on Θ,*

$\theta(\overset{n}{R}{}_{\text{iN=1}}P(i))$,

is a double partial differential of the semantic function

$f_\theta(\overset{n}{R}{}_{\text{iN=1}}P(i))$

on the sets of variables S and executing steps T, i.e.:

$$\theta(\overset{n}{R}{}_{\text{iN=1}}\text{P(i)}) \triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(\overset{n}{R}{}_{\text{iN=1}}\text{P(i)})$$
$$= \overset{n}{R}{}_{\text{kN= 1}}(\frac{\partial^2}{\partial t\,\partial s} f_\theta(P_i))$$
$$= \overset{n}{R}{}_{\text{kN=1}}(\overset{\#T(P_k)}{\underset{i=0}{R}}\ \overset{\#S(P_k)}{\underset{j=1}{R}} v_{P_k}(t_i, s_j))$$
$$= \begin{pmatrix} & \mathbf{k}\mathsf{N} & \mathbf{S_P} \\ \mathbf{[t_0, t_1]} & 1 & \perp \\ \mathbf{(t_1, t_2]} & 1 & V_P \\ \vdots & \vdots & \vdots \\ \mathbf{(t_{n-2}, t_{n-1}]} & n & - \\ \mathbf{(t_{n-1}, t_n]} & n & V_P \end{pmatrix} \quad (40)$$

Figure 9. The semantic diagram of the for-loop process relation

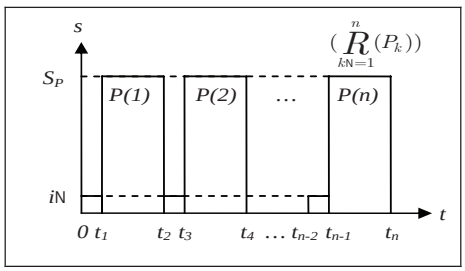


Figure 10. The semantic diagram of the function call process relation

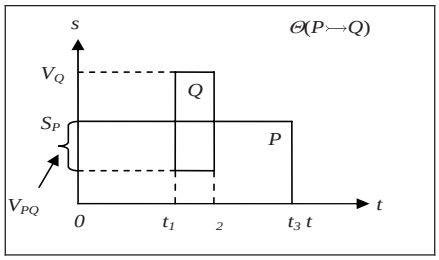
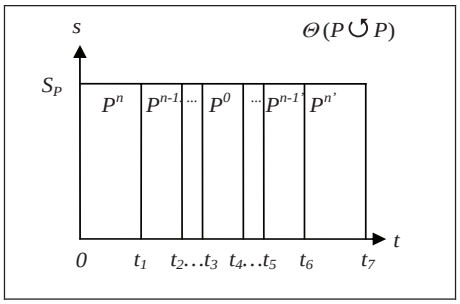


Figure 11. The semantic diagram of the recursive process relation



The semantic diagram of the for-loop process relation as defined in Equation 40 is illustrated in Figure 9 on Θ .

The Function Call Process Relation

Definition 39. *The semantics of the function call relations of processes on Θ, θ(P ⟼ Q), is a double partial differential of the semantic function f_θ(P ⟼ Q) on the sets of variables S and executing steps T, i.e.:*

$$\theta(P \rightarrowtail Q) \triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \rightarrowtail Q)$$

$$= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) \rightarrowtail \frac{\partial^2}{\partial t\, \partial s} f_\theta(Q)$$

$$= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j) \rightarrowtail \mathop{R}_{i=0}^{\#T(Q)} (\mathop{R}_{j=1}^{\#S(Q)} v_Q\ t_i, s_j)$$

$$= \mathop{R}_{i=0}^{\#T([t_0,t_1)^\frown(t_1,t_2)^\frown(t_2,t_3])} (\mathop{R}_{j=1}^{\#S(P\cup Q)} v\ t_i, s_j)$$

$$= \begin{pmatrix} & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} \\ \mathbf{t_0} & \perp & \perp & \perp \\ \mathbf{(t_0, t_1)} & V_{1P} & - & V_{1PQ} \\ \mathbf{(t_1, t_2)} & - & v_{2Q} & V_{2PQ} \\ \mathbf{(t_2, t_3)} & V_{3P} & - & V_{3PQ} \end{pmatrix} \tag{41}$$

The semantic diagram of the procedure call process relation as defined in Equation 41 is illustrated in Figure 10 on Θ(P ⟼ Q).

The Recursive Process Relation

Definition 40. *The semantics of the recursive relations of processes on Θ, θ(P↻P), is a double partial differential of the semantic function f_θ(P↻P) on the sets of variables S and executing steps T, i.e.:*

$$\theta(P \circlearrowright P) \triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \circlearrowright P)$$

$$= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) \circlearrowright \frac{\partial^2}{\partial t\, \partial s} f_\theta(P)$$

$$= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v\ t_i, s_j) \circlearrowright \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v\ t_i, s_j)$$

$$= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v\ t_i, s_j)$$

$$= \begin{pmatrix} & \mathbf{S_P} \\ \mathbf{[t_0, t_1]} & V_{P^n} \\ \mathbf{(t_1, t_2]} & V_{P^{n-1}} \\ \vdots & \vdots \\ \mathbf{(t_3, t_4]} & V_{P^0} \\ \vdots & \vdots \\ \mathbf{(t_5, t_6]} & V_{P'^{n-1}} \\ \mathbf{(t_6, t_7]} & V_{P'^n} \end{pmatrix} \tag{42}$$

The semantic diagram of the recursive process relation as defined in Equation 42 is illustrated in Figure 11 on Θ(P↻P).

Figure 12. The semantic diagram of the parallel process relation

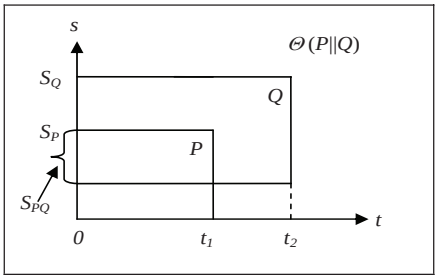
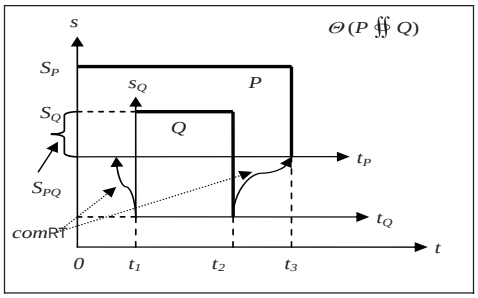


Figure 13. The semantic diagram of the concurrent process relation



The Parallel Process Relation

Definition 41. *The semantics of the parallel relations of processes on Θ, θ(P||Q), is a double partial differential of the semantic function f_θ(P||Q) on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}
\theta(P \parallel Q) &\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \parallel Q) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) \parallel \frac{\partial^2}{\partial t\, \partial s} f_\theta(Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j) \parallel \mathop{R}_{i=0}^{\#T(Q)} (\mathop{R}_{j=1}^{\#S(Q)} v_Q\ t_i, s_j) \\
&= \mathop{R}_{i=0}^{\max(\#T(P),\#T(Q))} (\mathop{R}_{j=1}^{\#S(P\cup Q)} v\ t, s) \\
&= \begin{pmatrix} & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} \\ \mathbf{t_0} & V_{0P} & V_{0Q} & V_{0PQ} \\ \mathbf{(t_0, t_1]} & V_{1P} & V_{1Q} & V_{1PQ} \\ \mathbf{(t_1, t_2]} & - & V_{2Q} & V_{2PQ} \end{pmatrix}
\end{aligned} \tag{43}$$

where t_2 = max(#T(P), (#T(Q)) is the synchronization point between two parallel processes.

The semantic diagram of the parallel process relation as defined in Equation 43 is illustrated in Figure 12 on Θ(P||Q).

It is noteworthy that parallel processes *P* and *Q* are interlocked. That is, they should start and end at the same time. In case $t_1 \neq t_2$, the process completed earlier, should wait for the completion of the other. The second condition between parallel processes is that the shared resources, in particular variables, memory space, ports, and devices should be protected. That is, when a process operates on a shared resource, it is locked to the other process until the operation is completed. A variety of interlocking and synchronization techniques, such as *semaphores*, *mutual exclusions*, and *critical regions*, have been proposed in real-time system techniques (McDermid, 1991).

The Concurrent Process Relation

Definition 42. *The semantics of the concurrent relations of processes on Θ, θ(P ⫰ Q), is a double partial differential of the semantic function f_θ(P ⫰ Q) on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}
\theta(P \fint Q) &\triangleq \frac{\partial^2}{\partial t\, \partial s} f_\theta(P \fint Q) \\
&= \frac{\partial^2}{\partial t\, \partial s} f_\theta(P) \fint \frac{\partial^2}{\partial t\, \partial s} f_\theta\ Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} (\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j) \fint \mathop{R}_{i=0}^{\#T(Q)} \mathop{R}_{j=1}^{\#S(Q)} v_Q(t_i, s_j) \\
&= \mathop{R}_{i=0}^{\max(\#T(P),\#T(Q))} (\mathop{R}_{j=1}^{\#S(P\cup Q)} v\ t_i, s_j) \\
&= \begin{pmatrix} & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} & \mathbf{com}\mathrm{RT} \\ \mathbf{t_0} & V_{0P} & V_{0P} & V_{0P} & V_{0com} \\ \mathbf{(t_0, t_1]} & V_{1P} & - & V_{1PQ} & V_{1com} \\ \mathbf{(t_1, t_2]} & V_{2P} & V_{2Q} & V_{2PQ} & V_{2com} \\ \mathbf{(t_2, t_3]} & V_{3P} & - & V_{3PQ} & V_{3com} \end{pmatrix}
\end{aligned} \tag{44}$$

where comRT is a set of interprocess communication variables that are used to synchronize P and Q executing on different machines based on independent system clocks.

The semantic diagram of the concurrent process relation as defined in Equation 44 is illustrated in Figure 13 on $\Theta(P \oiint Q)$.

The Interleave Process Relation

Definition 43. *The semantics of the interleave relations of processes on* Θ, $\theta(P \,|||\, Q)$, *is a double partial differential of the semantic function* $f_\theta(P \,|||\, Q)$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}
\theta(P \,|||\, Q) &\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(P \,|||\, Q) \\
&= \frac{\partial^2}{\partial t\ \partial s} f_\theta(P) \,|||\, \frac{\partial^2}{\partial t\ \partial s} f_\theta(Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} \Big(\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j \Big) \,|||\, \mathop{R}_{i=0}^{\#T(Q)} \mathop{R}_{j=1}^{\#S(Q)} v_Q(t_i, s_j) \\
&= \mathop{R}_{i=0}^{\#T([t_0,t_1] \frown (t_1,t_2) \frown (t_2,t_3) \frown (t_3,t_4] \frown (t_4,t_5])} \Big(\mathop{R}_{j=1}^{\#S(P \cup Q)} v\ t_i, s_j \Big) \\
&= \begin{pmatrix}
 & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} \\
\mathbf{t_0} & V_{0P} & V_{0Q} & V_{0PQ} \\
\mathbf{(t_0, t_1]} & V_{1P'} & - & V_{1PQ} \\
\mathbf{(t_1, t_2]} & - & V_{2Q'} & V_{2PQ} \\
\mathbf{(t_2, t_3]} & V_{3P''} & - & V_{3PQ} \\
\mathbf{(t_3, t_4]} & - & V_{4Q''} & V_{4PQ} \\
\mathbf{(t_4, t_5]} & V_{5P'''} & - & V_{5PQ}
\end{pmatrix}
\end{aligned} \tag{45}$$

The semantic diagram of the interleave process relation as defined in Equation 45 is illustrated in Figure 14 on $\Theta(P \,|||\, Q)$.

The Pipeline Process Relation

Definition 44. *The semantics of the pipeline relations of processes on* Θ, $\theta(P >> Q)$, *is a double partial differential of the semantic function* $f_\theta(P >> Q)$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}
\theta(P \gg Q) &\triangleq \frac{\partial^2}{\partial t\ \partial s} f_\theta(P \gg Q) \\
&= \frac{\partial^2}{\partial t\ \partial s} f_\theta(P) \gg \frac{\partial^2}{\partial t\ \partial s} f_\theta(Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} \mathop{R}_{j=1}^{\#S(P)} v_P(t_i, s_j) \gg \mathop{R}_{i=0}^{\#T(Q)} \mathop{R}_{j=1}^{\#S(Q)} v_Q(t_i, s_j) \\
&= \mathop{R}_{i=0}^{\#T(\widehat{PQ})} \Big(\mathop{R}_{j=1}^{\#S(P \cup Q)} v\ t_i, s_j \Big) \\
&= \begin{pmatrix}
 & \mathbf{S_P} & \mathbf{S_{Po} = S_{Qi}} & \mathbf{S_Q} \\
\mathbf{t_0} & V_{0P} & V_{0PQ} & V_{0Q} \\
\mathbf{(t_0, t_1]} & V_{1P} & V_{1PQ} & - \\
\mathbf{(t_1, t_2]} & - & V_{2PQ} & V_{2Q}
\end{pmatrix}
\end{aligned} \tag{46}$$

where $\mathbf{S_{P_o}}$ *and* $\mathbf{S_{Q_i}}$ *denote a set of n one-to-one connections between the outputs of P and inputs of Q, respectively, as follows:*

$$\mathop{R}_{k=0}^{n-1} (P_o(i) = Q_i(i)) \tag{47}$$

The semantic diagram of the pipeline process relation as defined in Equation 46 is illustrated in Figure 15 on $\Theta(P >> Q)$.

The Interrupt Process Relation

Definition 45. *The semantics of the interrupt relations of processes,* $\theta(P \leftthreetimes Q)$, *on a given semantic environment* Θ *is a double partial differential*

Figure 14. The semantic diagram of the interleave process relation

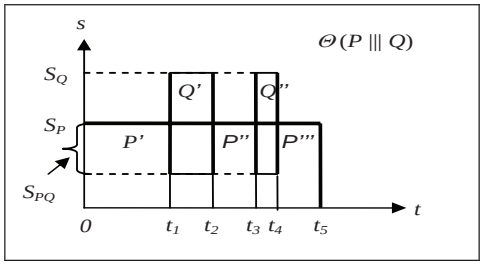


Figure 15. The semantic diagram of the pipeline process relation

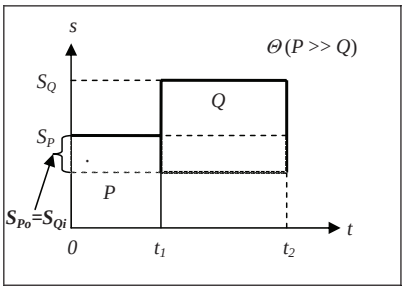
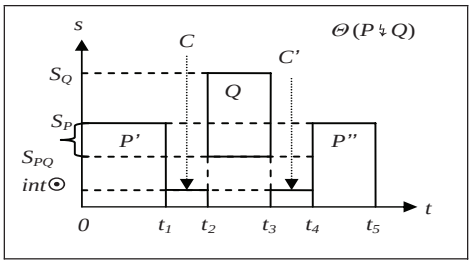


Figure 16. The semantic diagram of the interrupt process relation



of the semantic function $f_\theta(P \lightning Q)$ on the sets of variables S and executing steps T, i.e.:

$$\begin{aligned}
\theta(P \lightning Q) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(P \lightning Q) \\
&= \frac{\partial^2}{\partial t\,\partial s} f_\theta(P) \lightning \frac{\partial^2}{\partial t\,\partial s} f_\theta(Q) \\
&= \mathop{R}_{i=0}^{\#T(P)} \left(\mathop{R}_{j=1}^{\#S(P)} v_P\ t_i, s_j \right) \lightning \mathop{R}_{i=0}^{\#T(Q)} \left(\mathop{R}_{j=1}^{\#S(Q)} v_Q\ t_i, s_j \right) \\
&= \mathop{R}_{i=0}^{\#T(P' \frown Q \frown P'')} \left(\mathop{R}_{j=1}^{\#S(P \cup Q)} v\ t_i, s_j \right) \\
&= \begin{pmatrix}
 & \mathbf{S_P} & \mathbf{S_Q} & \mathbf{S_{PQ}} & \mathbf{int}\odot \\
[\mathbf{t_0}, \mathbf{t_1}] & V_{1P'} & \perp & V_{1PQ} & \perp \\
(\mathbf{t_1}, \mathbf{t_2}] & - & - & V_{2PQ} & V_{2\mathbf{int}\odot} \\
(\mathbf{t_2}, \mathbf{t_3}] & - & V_{3Q} & V_{3PQ} & - \\
(\mathbf{t_3}, \mathbf{t_4}] & - & - & V_{4PQ} & V_{4\mathbf{int}'\odot} \\
(\mathbf{t_4}, \mathbf{t_5}] & V_{5P''} & - & V_{5PQ} & -
\end{pmatrix}
\end{aligned} \tag{48}$$

The semantic diagram of the interrupt process relation as defined in Equation 48 is illustrated in Figure 16 on $\Theta(P \lightning Q)$, where $C(\text{int}'\odot)$ and $C'(\text{int}'\odot)$ are the interrupt and interrupt-return points, respectively.

The deductive semantics of the three system dispatch process relations will be presented in the next section.

DEDUCTIVE SEMANTICS OF SYSTEM-LEVEL PROCESSES OF RTPA

The deductive semantics of systems at the top level of programs can be reduced onto a dispatch mechanism of a finite set of processes based on the mechanisms known as time, event, and interrupt. This section first describes the deductive semantics of the system process. Then, the three system dispatching processes will be formally modeled.

The System Process

Definition 46. *The semantics of the system process § on Θ, θ(§), is an abstract logical model of the executing platform with a set of parallel dispatched*

processes based on internal system clock, external events, and system interrupts, i.e.:

$$\begin{aligned}\theta(\S) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(\S) \\ &= \frac{\partial^2}{\partial t\,\partial s} f_\theta\{ \mathop{R}_{i=0}^{n_e-1}(@e_i \mathsf{S}\hookrightarrow P_i) \\ &\quad \| (\mathop{R}_{j=0}^{n_t-1} @t_j \mathsf{TM}\hookrightarrow P_j) \\ &\quad \| (\mathop{R}_{k=0}^{n_{int}-1} @int_k \mathsf{S}\hookrightarrow P_k) \\ &\quad \} \\ &= \mathop{R}_{SysShuntDown \mathsf{BL}=\mathsf{F}}^{\mathsf{T}} \{ (\mathop{R}_{i=0}^{n_e-1} @e_i \mathsf{S}\hookrightarrow P_i) \\ &\quad \| (\mathop{R}_{j=0}^{n_t-1} @t_j \mathsf{TM}\hookrightarrow P_j) \\ &\quad \| (\mathop{R}_{k=0}^{n_{int}-1} @int_k \mathsf{S}\hookrightarrow P_k) \\ &\quad \}\end{aligned} \tag{49}$$

where the semantics of the parallel relations has been given in Definition 41, and those of the system dispatch processes will be described in the following subsections.

The Time-Driven Dispatching Process Relation

Definition 47. *The semantics of the time-driven dispatching relations of processes on* Θ, $\theta(@t_k\mathsf{TM} \hookrightarrow_t P_k)$*, is a double partial differential of the semantic function* $f_\theta(@t_k\mathsf{TM} \hookrightarrow_t P_k)$ *on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(@t_k\mathsf{TM} \hookrightarrow_k P_k) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(@t_k\mathsf{TM} \hookrightarrow_k P_k) \\ &= \mathop{R}_{k=1}^{n}(@t_k\mathsf{TM} \to \frac{\partial^2}{\partial t\,\partial s} f_\theta(P_k)) \\ &= \mathop{R}_{k=1}^{n}(@t_k\mathsf{TM} \to \mathop{R}_{i=0}^{\#T(P_k)} \mathop{R}_{j=1}^{\#S(P_k)} v_{P_k}(t_i, s_j)) \\ &= \blacklozenge @t_1\mathsf{TM} \to \mathop{R}_{i=0}^{\#T(P_1)} (\mathop{R}_{j=1}^{\#S(P_1)} v_{P_1}\ t_i, s_j) \\ &\quad | \ ... \\ &\quad | \ \blacklozenge @t_n\mathsf{TM} \to \mathop{R}_{i=0}^{\#T(P_n)} \mathop{R}_{j=1}^{\#S(P_n)} v_{P_n}(t_i, s_j) \\ &= \begin{pmatrix} & @t_k\mathsf{TM} & \mathbf{S}_{\mathbf{P_1}} & ... & \mathbf{S}_{\mathbf{P_n}} \\ \mathbf{[t_0, t_1]} & \delta(@t_k\mathsf{TM}) & \perp & \cdots & \perp \\ \mathbf{(t_1, t_2]} & @t_1 & V_{P_1} & \cdots & - \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{(t_1, t_n]} & @t_n & - & \cdots & V_{P_n} \end{pmatrix}\end{aligned} \tag{50}$$

where $\blacklozenge(@t_k\mathsf{TM}) = \blacklozenge(@t_k\mathsf{N})$ *is the evaluation function as defined in Equation 17b.*

The semantic diagram of the time-driven dispatching process relation as defined in Equation 50 is illustrated in Figure 17 on Θ.

The Event-Driven Dispatching Process Relation

Definition 48. *The semantics of the event-driven dispatching relations of processes on* Θ, $\theta(@$

Figure 17. The semantic diagram of time-driven dispatch relation

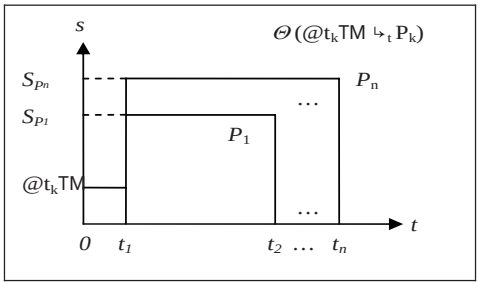
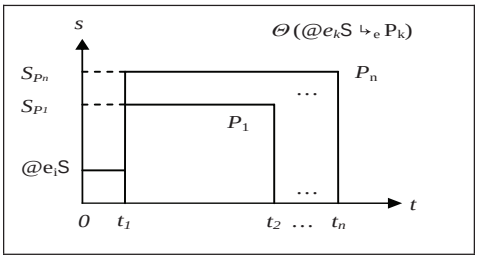


Figure 18. The semantic diagram of the event-driven dispatch relation



e_kS $\hookrightarrow_e P_k$), is a double partial differential of the semantic function f_θ(@e_kS $\hookrightarrow_e P_k$) on the sets of variables S and executing steps T, i.e.:

$$\begin{aligned}\theta(@e_k\mathrm{S}\hookrightarrow_e \mathrm{P}_k) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(@e_k\mathrm{S}\hookrightarrow_e \mathrm{P}_k)\\ &= \mathop{R}_{k=1}^{n}(@e_k\mathrm{S}\rightarrow \frac{\partial^2}{\partial t(P_k)\partial s(P_k)} f_\theta(P_k))\\ &= \mathop{R}_{k=1}^{n}(@e_k\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_k)}\mathop{R}_{j=1}^{\#S(P_k)} v_{P_k}(t_i,s_j))\\ &= \blacklozenge @e_1\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_1)}\mathop{R}_{j=1}^{\#S(P_1)} v_{P_1}(t_i,s_j)\\ &\quad |\ \ldots\\ &\quad |\ \blacklozenge @e_n\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_n)}\mathop{R}_{j=1}^{\#S(P_n)} v_{P_n}(t_i,s_j)\\ &= \begin{pmatrix} & \mathbf{@e_k}\mathrm{S} & \mathbf{S_{P_1}} & \ldots & \mathbf{S_{P_n}}\\ \mathbf{[t_0,t_1]} & \delta(@e_k\mathrm{S}) & \perp & \cdots & \perp\\ \mathbf{(t_1,t_2]} & @e_1 & V_{P_1} & \cdots & -\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{(t_1,t_n]} & @e_n & - & \cdots & V_{P_n}\end{pmatrix}\end{aligned} \tag{51}$$

The semantic diagram of the event-driven process relation as defined in Equation 51 is illustrated in Figure 18 on Θ.

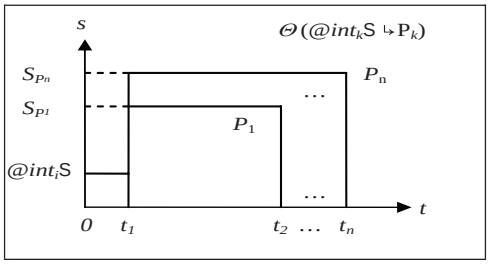
The Interrupt-Driven Dispatching Process Relation

Definition 49. *The semantics of the interrupt-driven dispatching relations of processes on* Θ, *θ(@int_kS $\hookrightarrow_i P_i$), is a double partial differential of the semantic function f_θ(@int_kS $\hookrightarrow_i P_i$) on the sets of variables S and executing steps T, i.e.:*

$$\begin{aligned}\theta(@int_k\mathrm{S}\hookrightarrow_i \mathrm{P}_k) &\triangleq \frac{\partial^2}{\partial t\,\partial s} f_\theta(@int_k\mathrm{S}\hookrightarrow_i \mathrm{P}_k)\\ &= \mathop{R}_{k=1}^{n}(@int_k\mathrm{S}\rightarrow \frac{\partial^2}{\partial t(P_k)\partial s(P_k)} f_\theta(P_k))\\ &= \mathop{R}_{k=1}^{n}(@int_k\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_k)}\mathop{R}_{j=1}^{\#S(P_k)} v_{P_k}(t_i,s_j))\\ &= @int_1\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_1)}(\mathop{R}_{j=1}^{\#S(P_1)} v_{P_1}\ t_i,s_j)\\ &\quad |\ \ldots\\ &\quad |\ @int_n\mathrm{S}\rightarrow \mathop{R}_{i=0}^{\#T(P_n)}\mathop{R}_{j=1}^{\#S(P_n)} v_{P_n}(t_i,s_j)\\ &= \begin{pmatrix} & \mathbf{@int_k}\mathrm{S} & \mathbf{S_{P_1}} & \ldots & \mathbf{S_{P_n}}\\ \mathbf{[t_0,t_1]} & \delta(@int_k\mathrm{S}) & \perp & \cdots & \perp\\ \mathbf{(t_1,t_2]} & @e_1 & V_{P_1} & \cdots & -\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{(t_1,t_n]} & @e_n & - & \cdots & V_{P_n}\end{pmatrix}\end{aligned} \tag{52}$$

The semantic diagram of the interrupt-driven process relation as defined in Equation 52 is illustrated in Figure 19 on Θ.

Figure 19. The semantic diagram of the interrupt-driven dispatch relation



CONCLUSION

Semantics plays an important role in cognitive informatics, computational linguistics, computing, and software engineering theories. Deductive semantics is a formal software semantics that deduces the semantics of a program in a given programming language from a generic abstract semantic function to the concrete semantics, which are embodied by the changes of statuses of a finite set of variables constituting the semantic environment of computing. Based on the mathematical models and architectural properties of programs at different composing levels, deductive models of software semantics, semantic environment, and semantic matrix have been formally defined. Properties of software semantics and relations between the software behavioral space and semantic environment have been discussed.

Case studies on the deductive semantic rules of RTPA have been presented, which serve not only as a comprehensive paradigm, but also the verification of the expressive and analytic capacity of deductive semantics.

A rigorous treatment of deductive semantics of RTPA has been presented in this article, which enables a new approach towards deductive reasoning of software semantics at all composing levels of the program hierarchy (i.e., statements, processes, and programs) from the bottom up. Deductive semantics has greatly simplified the description and analysis of the semantics of complicated software systems implemented in programming languages or specified in formal notations. Deductive semantics can be used to define both abstract and concrete semantics of large-scale software systems, facilitate software comprehension and recognition, support tool development, enable semantics-based software testing and verification, and explore the semantic complexity of software systems.

ACKNOWLEDGMENT

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The author would like to thank anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Bjorner, D. (2000, November). Pinnacles of software engineering: 25 years of formal methods. In Y. Wang & D. Patel (Eds.), *Annals of software engineering: An international journal,10,* 11-66.

Bjorner, D., & Jones, C. B. (1982). *Formal specification and software development.* Englewood Cliffs, NJ: Prentice Hall.

Chomsky, N. (1956). Three models for the description of languages. *I.R.E. Transactions on Information Theory, 2*(3), 113-124.

Chomsky, N. (1957). *Syntactic structures.* The Hague, The Netherlands: Mouton.

Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding.* Cambridge, MA: MIT Press

Dijkstra, E. W. (1975). Guarded commands, nondeterminacy, and the formal derivation of programs. *Communications of the ACM, 18*(8), 453-457.

Dijkstra, E. W. (1976). *A discipline of programming.* Englewood Cliffs, NJ: Prentice Hall.

Goguen, J. A., & Malcolm, G. (1996). *Algebraic semantics of imperative programming.* Cambridge, MA: MIT Press.

Goguen, J.A., Thatcher, J. W., Wagner, E. G., & Wright, J. B. (1977). Initial algebra semantics and continuous algebras. *Journal of the ACM, 24*(1), 68-59.

Gries, D. (1981). *The science of programming.* Berlin, Germany: Spinger Verlag

Gunter, C. A. (1992). Semantics of programming languages: Structures and techniques. In M. Garey & A. Meyer (Eds.), *Foundations of computing.* Cambridge, MA: MIT Press.

Guttag, J. V., & Horning, J. J. (1978). The algebraic specification of abstract data types. *Acta Informatica, 10,* 27-52.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM, 12*(10), 576-580.

Jones, C. B. (1980). *Software development: A rigorous approach.* London: Prentice Hall International.

Louden, K. C. (1993). *Programming languages: Principles and practice*. Boston: PWS-Kent.

Marcotty, M., & Ledgard. H. (1986). *Programming language landscape* (2nd ed.). Chicago: SRA.

McDermid, J. A. (Ed.). (1991). *Software engineer's reference book*. Oxford, England: Butterworth-Heinemann.

Meyer, B. (1990). *Introduction to the theory of programming languages*. Englewood Cliffs, NJ: Prentice Hall

Ollongren, A. (1974). *Definition of programming languages by interpreting automata*. New York: Academic Press.

Pagan, F. G. (1981). *Semantics of programming languages: A panoramic primer.* Englewood Cliffs, NJ: Prentice Hall.

Schmidt, D. (1988). *Denotational semantics: A methodology for language development*. Dubuque, IA: Brown.

Schmidt, D. (1996, March). Programming language semantics. *ACM Computing Surveys*, *28*(1).

Schmidt, D. A. (1994). *The structure of typed programming languages.* Cambridge, MA: MIT Press.

Scott, D. (1982). Domains for denotational semantics. In *Automata, languages and programming IX* (pp. 577-613). Berlin, Germany: Springer Verlag.

Scott, D. S., & Strachey, C. (1971). *Towards a mathematical semantics for computer languages.* (Programming Research Group Tech. Rep. PRG-1-6). Oxford University.

Slonneg, K., & Kurts, B. (1995). *Formal syntax and semantics of programming languages.* Addison-Wesley.

Tarski, A. (1944). The semantic conception of truth. *Philosophic Phenomenological Research, 4*, 13-47.

Wang, Y. (2002). The real-time process algebra (RTPA). *Annals of Software Engineering: A International Journal, 14*, 235-274.

Wang, Y. (2003). Using Process algebra to describe human and software system behaviors. *Brain and Mind*, *4*(2), 199-213.

Wang, Y. (2006a). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation (Invited plenary talk). In *Proceedings of the First International Conference on Rough Set and Knowledge Technology (RSKT'06)* (LNAI 4062, pp. 69-78). Chongqing, China: Springer.

Wang, Y. (2006b). On the informatics laws and deductive semantics of software, *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 161-171.

Wang, Y. (2007a). The Cognitive Processes of Formal Inferences. *The International Journal of Cognitive Informatics and Natural Intelligence*, *1*(4), 75-86.

Wang, Y. (2007b). Keynote speech on theoretical foundations of software engineering and denotational mathematics. In *Proceedings of the Fifth Asian Workshop on Foundations of Software* (pp. 99-102). Xiamen, China:

Wang, Y. (2007c). *Software engineering foundations:A software science perspective.* Auerbach Publications, NY., July.

Wang, Y. (2008a). On the big-R notation for describing iterative and recursive behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(1),17-28.

Wang, Y. (2008b, April). RTPA: A denotational mathematics for manipulating intelligent and

computational behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence*, 2(2), 44-62.

Wegner, P. (1972). The Vienna definition language. *ACM Computing Surveys, 4*(1), 5-63.

Wikstrom, A. (1987). *Functional programming using standard ML*. Englewood Cliffs, NJ: Prentice Hall.

This work was previously published in International Journal of Cognitive Informatics and Natural Intelligence, Vol. 2, Issue 2, edited by Y. Wang, pp. 95-121, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.19
RTPA:
A Denotational Mathematics for Manipulating Intelligent and Computational Behaviors

Yingxu Wang
University of Calgary, Canada

ABSTRACT

Real-time process algebra (RTPA) is a denotational mathematical structure for denoting and manipulating system behavioral processes. RTPA is designed as a coherent algebraic system for intelligent and software system modeling, specification, refinement, and implementation. RTPA encompasses 17 metaprocesses and 17 relational process operations. RTPA can be used to describe both logical and physical models of software and intelligent systems. Logic views of system architectures and their physical platforms can be described using the same set of notations. When a system architecture is formally modeled, the static and dynamic behaviors performed on the architectural model can be specified by a three-level refinement scheme at the system, class, and object levels in a top-down approach. RTPA has been successfully applied in real-world system modeling and code generation for software systems, human cognitive processes, and intelligent systems.

INTRODUCTION

The modeling and refinement of software and intelligent systems require new forms of denotational mathematics that possess enough expressive power for rigorously describing system architectures and behaviors. Real-time process algebra (RTPA) is a new denotational mathematical structure for software system modeling, specification, refinement, and implementation for both real-time and nonreal-time systems (Wang, 2002b, 2007d; Wang & King, 2000). An important finding in formal methods is that a software system can be perceived and described as the *composition* of a set of *processes*, which are constructed on the basis of algebraic operations. Theories on process algebras can be traced back

to Hoare's *Communicating Sequential Processes* (CSP) (Hoare, 1978, 1985) and Milner's *Calculus of Communicating Systems* (CCS) (Milner, 1980). The *process metaphor* of software systems has evolved from concurrent processes to real-time process systems in the area of operating system research and formal methods (Boucher & Gerth, 1987; Hoare, 1978, 1985; Milner, 1980, 1989; Reed & Roscoe, 1986; Schneider, 1991).

Definition 1. *A process is an abstract model of a unit of meaningful system behaviors that represents a transition procedure of the system from one state to another by changing values of the sets of inputs, outputs, and/or internal variables.*

It is recognized that generic computing problems are 3-D problems, known as those of the behavior, space, and time dimensions, which require a denotational mathematical means for addressing the requirements in all dimensions, particularly the time dimension (Wang, 2002b, 2003b, 2007d). However, conventional process models are hybrid (Hoare, 1985), which do not distinguish the concepts of fundamental metaprocesses and process operations between them. The CSP notation models a major part of elementary software behaviors that may be used in system specification and description. However, it lacks many useful processes that are perceived essential in system modeling, such as addressing, memory manipulation, timing, and system dispatch. CSP models all input and output (I/O) as abstract channel operations that are not expressive enough to denote complex system interactions, particularly for those of real-time systems.

A number of timed extensions and variations of process algebra have been proposed (Baeten & Bergstra, 1991; Boucher & Gerth, 1987; Cerone, 2000; Corsetti, Montanari, & Ratto, 1991; Dierks, 2000; Fecher, 2001; Gerber, Gunter, & Lee, 1992; Jeffrey, 1992; Klusener, 1992; Nicollin & Sifakis, 1991; Reed & Roscoe, 1986; Schneider, 1991; Vereijken, 1995). It is found that the existing work on process algebra and their timed variations can be extended and refined to a new form of denotational mathematics, RTPA, based on a set of algebraic process operations and laws (Wang, 2002b, 2003b, 2006a, 2006c, 2007d, 2008a). RTPA can be used to formally and precisely describe and specify architectures and behaviors of software systems on the basis of algebraic process notations and rules.

Definition 2. *A process P in RTPA is a composed component of n metastatements s_i and s_j, $1 \le i < n$, $j = i + 1$, according to certain composing relations r_{ij}, i.e.:*

$$P = (...(((s_1)\ r_{12}\ s_2)\ r_{23}\ s_3)\ ...\ r_{n-1,n}\ s_n) \tag{1}$$

where $r_{ij} \in \mathfrak{R}$, which is a set of relational process operators of RTPA that will be formally defined in Lemma 2.

Definition 2 indicates that the mathematical model of a process is a cumulative relational structure among computing operations. The simplest process is a single computational statement. Further discussion will be provided in the section on the unified mathematical model of programs.

Definition 3. *RTPA is a denotational mathematical structure for algebraically denoting and manipulating system behavioral processes and their attributes by a triple, i.e.:*

$$RTPA \triangleq (\mathfrak{T}, \mathfrak{P}, \mathfrak{R}) \tag{2}$$

where $\mathfrak{T}$ is a set of 17 primitive types for modeling system architectures and data objects, $\mathfrak{P}$ is a set of 17 metaprocesses for modeling fundamental system behaviors, and $\mathfrak{R}$ is a set of 17 relational process operations for constructing complex system behaviors.

RTPA provides a coherent notation system and a formal engineering methodology for modeling both software and intelligent systems. RTPA can be used to describe both *logical* and *physical* models of systems, where logical views of the architecture of a software system and its operational platform can be described using the same set of notations. When the system architecture is formally modeled, the static and dynamic behaviors that perform on the system architectural model can be specified by a three-level refinement scheme at the system, class, and object levels in a top-down approach.

This article presents the RTPA structure and methodology for modeling software and intelligent system behaviors. The type system, process notations, process relations, and process composing rules of RTPA are described. The type system and formal type rules of RTPA are formally described, and their usage in data object and system architectural modeling are demonstrated. The fundamental processes and their denotational functionality in modeling both human and system behaviors are presented. A set of algebraic process operations is introduced for constructing complex processes and systems. Then, applications of RTPA in computing and intelligent system modeling and manipulation are explored with case studies. A unified mathematic model of software and programs is derived based on the RTPA theories. The system specification and refinement methodology of RTPA and case studies on real-world problems are provided, which demonstrate the descriptive power of RTAP as a precise and neat algebraic system for software engineering.

THE TYPE SYSTEM OF RTPA

Computational operations of systems can be classified into the categories of *data object*, *behavior*, and *resource* modeling and manipulations. Based on this view, programs are perceived as the coordination of data objects and behaviors in computing. Data object modeling is a process to creatively extract and abstractly represent a real-world problem by computing objects based on the constraints of given computing resources. Using types to model the natural world can be traced back to the mathematical thought of Bertrand Russell (Russell, 1961; Schilpp, 1946) and Godel (van Heijenoort, 1997). A type is a category of variables that share a common property, such as kinds of data, domain, and allowable operations. Types are an important logical property shared by data objects in programming. Although data in their most primitive form is a string of bits, types are found expressively convenient for data representation at the logical level in computing and software engineering.

The Type System for Data Objects Modeling in RTPA

A type is a set in which all member data objects share a common logical property or attribute. The maximum range of values that a variable can assume is a type, and a type is associated with a set of predefined or allowable operations. A type can be classified as *primitive* and *derived* (complex). The former is the most elemental of types that cannot further be divided into simpler ones; the latter is a compound form of multiple primitive types based on given type rules. Most primitive types are provided by programming languages while most user-defined types are derived ones.

Definition 4. *A type system is a set of predefined templates and manipulation rules for modeling data objects and system architectures.*

RTPA types, syntaxes, and properties are given in Table 1, where the 17 primitive types in computing and human cognitive process modeling have been elicited (Cardelli & Wegner, 1985;Martin-Lof, 1975; Mitchell, 1990; Wang, 2002a, 2002b, 2003b, 2006a, 2007d, 2007e). In Table 1, the first

11 primitive types are for mathematical and logical manipulation of data objects, and the remaining 6 are for system architectural modeling.

Lemma 1. *The primary types of computational objects state that the RTPA type system $\mathfrak{T}$ encompasses 17 primitive types elicited from fundamental computing needs, i.e.:*

$$\mathfrak{T} \triangleq \{\mathsf{N}, \mathsf{Z}, \mathsf{R}, \mathsf{S}, \mathsf{BL}, \mathsf{B}, \mathsf{H}, \mathsf{P}, \mathsf{TI}, \mathsf{D}, \mathsf{DT}, \mathsf{RT}, \mathsf{ST}, @e\mathsf{S}, @t\mathsf{TM}, @int\odot, \circledS s\mathsf{BL}\} \quad (3)$$

RTPA adopts the *type suffix convention* in which every variable *x* declared in a type T, *x* : T, by a bold type label attached to the variable in all invocations in the form *x*T, where T is any valid primitive as defined in Table 1 or a derived type based on the table. The advance of the type suffix convention is the improvement of readability. Using the type suffixes, system analysts may easily identify if all variables in a statement or expression are equivalent or compatible without referring to earlier declarations, which may be scattered in a system model across hundreds of pages in a large-scale software. The type suffix convention also greatly simplifies type checking requirements during parsing the RTPA specifications by machines (Tan, Wang, & Noglah, 2004, 2006).

An essence of type theory is that types can be classified into the domains of *mathematical* (logical) D_m, *language defined* D_l, and *user defined* D_u, as shown in Table 1 (2007d).

Theorem 1. *The domain constraints of data objects states that the following relationship between the domains of any data object in computing is always held, i.e.:*

$$D_u \subseteq D_l \subseteq D_m \quad (4)$$

It is noteworthy that although a generic computing behavior is constrained by D_m, an executable program is constrained by D_l, and, at most time, it is further restricted by the user defined domain D_u, where $D_u \subseteq D_l$. According to Theorem 1, the following corollary can be derived.

Corollary 1. *The precedence of domain determinations and type inferences in computing and software engineering is always as follows:*

$$D_u \Rightarrow D_l \Rightarrow D_m \quad (5)$$

Advanced Types of RTPA

The most common and powerful derived type in computing is a *record*, also known as a *construct*, because its flexibility to accommodate different data fields in different primary or composed types. System architectures can be modeled on the basis of structured records. There are also a number of special advanced types introduced in RTPA, such as the *system* type, dynamic *run-time* type, and *event*, *interrupt*, and *status* types (Wang, 2002b, 2007d).

Definition 5. *The run-time type* RT *is a non-deterministic type at compile-time that can be dynamically bound during run-time with one of the predefined primitive types.*

The run-time type RT provides programmers a powerful means to express and handle highly flexible and nondeterministic computing objects in data modeling. Some languages such as Java and IDL (OMG, 2002) label the dynamic type RT as the *anytype*, for which a specific type may be bound until run time.

Definition 6. *An event is an advanced type in computing that captures the occurring of a predefined external or internal change of status, such as an action of users, an external change of environment, and an internal change of the value of a specific variable.*

Table 1. RTPA primitive types and their domains

No.	Type	Syntax	D_m	D_l	Equivalence
1	Natural number	N	[0, +∞]	[0, 65535]	Arithmetic, mathematic, assignment
2	Integer	Z	[-∞, +∞]	[-32768, +32767]	
3	Real	R	[-∞, +∞]	[-2147483648, 2147483647]	
4	String	S	[0, +∞]	[0, 255]	String and character operations
5	Boolean	BL	[T, F]	[T, F]	Logical, assignment
6	Byte	B	[0, 256]	[0, 256]	Arithmetic, assignment, addressing
7	Hexadecimal	H	[0, +∞]	[0, max]	
8	Pointer	P	[0, +∞]	[0, max]	
9	Time	TI = hh:mm:ss:ms	hh: [0, 23] mm: [0, 59] ss: [0, 59] ms: [0, 999]	hh: [0, 23] mm: [0, 59] ss: [0, 59] ms: [0, 999]	Timing, duration, arithmetic (A generic abbreviation: TI={TI, D, DT})
10	Date	D = yy:MM:dd	yy: [0, 99] MM: [1, 12] dd: [1, 31}	yy: [0, 99] MM: [1, 12] dd: [1, 31}	
11	Date/Time	DT = yyyy:MM:dd: hh:mm:ss:ms	yyyy: [0, 9999] MM:[1, 12] dd: [1, 31] hh: [0, 23] mm: [0, 59] ss: [0, 59] ms: [0, 999]	Yyyy: [0, 9999] MM:[1, 12] dd: [1, 31] hh: [0, 23] mm: [0, 59] ss: [0, 59] ms: [0, 999]	
12	Run-time determinable type	RT	–	–	Operations suitable at run-time
13	System architectural type	ST	–	–	Assignment (field reference by '.')
14	Random event	@*e*S	[0, +∞]	[0, 255]	String operations
15	Time event	@*t*TM	[0ms, 9999 yyyy]	[0ms, 9999 yyyy]	Logical
16	Interrupt event	@*int*⊙	[0, 1023]	[0, 1023]	Logical
17	Status	ⓢ*s*BL	[T, F]	[T, F]	Logical

The event types of RTPA can be classified into those of *operation* (@*e*S), *time* (@*t*TM), and *interrupt* (@*int*⊙), as shown in Table 1, where @ is the *event prefix*, and S, TM, and ⊙ the corresponding type suffixes, respectively.

A special set of complex types known as the system type ST is widely used for modeling system architectures in RTPA, particularly real-time, embedded, and distributed systems architectures. All the system types are nontrivial data objects in computing, rather than simple data or logical objects, which play a very important role in the whole lifecycle of complex system development including design, modeling, specification, refinement, comprehension, implementation, and maintenance of such systems.

Definition 7. *A system type* ST *is a system architectural type that models the architectural components of the system and their relations.*

A generic ST type is the *Component Logical Model* (CLM), as defined in RTPA (Wang, 2002b, 2007d). CLMs are powerful modeling means in system architectural modeling.

Definition 8. *CLMs are a record-structured abstract model of a system architectural component that represents a hardware interface, an internal logical model, and/or a common control structure of a system.*

CLMs can be used for unifying user defined complex types in system modeling. A formal treatment of CLMs will be provided in Definition 12.

It is recognized that any form of intelligence is memory based (Wang, 2002a, 2203a, 2006b, 2007c, 2007d, 2007e). All data objects, no matter language generated or user created, should be implemented as physical data objects and be bound to specific memory locations. Therefore, memory models play an important role in system modeling.

Definition 9. *The generic system memory model, MEM*ST*, can be described as a special system type* ST *with a finite linear space, i.e.:*

$$\text{MEMST} \triangleq [addr_1\text{H} \ldots addr_2\text{H}]\text{RT} \tag{6}$$

where addr$_1$H *and addr*$_2$H *are the start and end addresses of the memory space, and* RT *is the type of each of the memory elements, which is usually in Byte* B *in computing.*

Another special system type is the I/O port type for modeling hardware architectures and their interfaces.

Definition 10. *The generic system I/O port model, PORT*ST*, can be described as a system type* ST *with a finite linear space, i.e.:*

$$\text{PORTST} \triangleq [ptr_1\text{H} \ldots ptr_2\text{H}]\text{RT} \tag{7}$$

where ptr$_1$H *and ptr*$_2$H *are the start and end addresses of the port space, and* RT *is the type of each of the port I/O interfaces, which is usually in Byte* B *in computing.*

Formal Type Rules of RTPA

A type system specifies the data objects modeling and composing rules of a programming language as that of a grammar system which specifies the program behavior modeling and composing rules of the language. The basic *properties* of type systems are decidable, transparent, and enforceable (Cardelli & Wegner, 1985; Martin-Lof, 1975; Mitchell, 1990). Type systems should be *decidable* by a type checking system, which ensures that types of variables are both well-declared and referred. Type systems should be *transparent* that helps for diagnoses of reasons for inconsistency between variables or variables and their declarations. Type systems should be *enforceable* in order to check type inconsistence as much as possible.

A *formal type system* is a collection of all type rules for a given programming language or formal notation system. A type rule is a mathematical relation and the constraints on a given type. Type rules are defined on the basis of a type environment.

Definition 11. *The type environment* Θ_t *of RTPA is a collection of all primitive types in the formal notation system, i.e.:*

$$\Theta_t \triangleq \mathfrak{T} = \{\text{N, Z, R, S, BL, B, H, P, TI, D, DT, RT, ST, } @e\text{S, } @t\text{TM, } @int\odot, \circledS s\text{BL}\} \tag{8}$$

where $\mathbb{T}$ is the set of primary types as defined in Table 1.

Complex and derived types of RTPA can be described by composing type rules based on those of the primitive types.

As given in Definition 8, a CLM is a generic system architectural type for modeling and manipulating data objects and system architectures (Wang, 2002b, 2007d).

Definition 12. *The type rule of a CLM type, CLM, is a complex system type* ST *in RTPA derived from* Θ_t*, i.e.:*

$$\frac{\Theta_t \vdash \mathsf{ST}}{\Theta_t \vdash CLM : \mathsf{ST}} \tag{9}$$

The declaration of a variable, *ClmID*, with a given CLM type can be denoted by using the following type rule as given in Equation 10, where the ClmIDST is defined by the string type label ClmIDS with an n-field structure, each of them specifies a metavariable ID_i in type T_i, and its constraints denoted by Constraint($ID_i T_i$), which are a set of expressions.

A *process* in RTPA is a basic behavioral unit for modeling software system operations onto the data objects. A process can be a metaprocess or a complex process composed with multiple metaprocesses by relational process operators. Because processes are so frequently used in system modeling, a derived type in RTPA known as the process type can be introduced as a special system type.

Definition 13. *The type rule of a process type, PROC, is a complex system type* ST *in RTPA derived from* Θ_t*, i.e.:*

$$\frac{\Theta_t \vdash \mathsf{ST}}{\Theta_t \vdash PROC : \mathsf{ST}} \tag{11}$$

The declaration of a variable, *ProcID*, with the *PROC* type can be denoted by using the following type rule as given in Equation 11, where the ProcIDST is defined by the string type label ProcIDS with a set of n inputs and a set of m outputs in a specific type, as well as a set of q I/O constructs or CLMs in a specific ST type.

METAPROCESSES OF RTPA

On the basis of the process metaphor, this section elicits the most general and fundamental system behaviors in computing and intelligent systems. Computational operations in conventional process algebra, such as CSP (Hoare, 1985), Timed-CSP (Boucher & Gerth, 1987; Reed & Roscoe, 1986; Schneider, 1991), and other proposals are treated as a set of processes at the same level. This approach results in an exhaustive listing of processes. Whenever a new operation is identified or required in computing, the existing process system must be extended.

RTPA adopts the foundationalism in order to find the most primitive computational processes

Box 1.

$$\frac{\Theta_t \vdash \mathsf{ST}, \Theta_t \vdash \mathsf{T}, \Theta_t \vdash ClmID{:}\mathsf{ST}, \Theta_t \vdash ID{:}\mathsf{T}}{\Theta_t \vdash ClmID\mathsf{ST} \triangleq ClmID\mathsf{S} :: \{ \mathop{\mathsf{R}}_{i=1}^{n} < ID_i : T_i \mid \text{Constraint}(ID_i T_i) >; \}} \tag{10}$$

Box 2.

$$\frac{\Theta_t \vdash ProcID{:}\mathsf{ST}}{\Theta_t \vdash ProcID\mathsf{ST} \triangleq ProcID\mathsf{S}}$$

$$(\mathrm{I}::<\mathop{\mathrm{R}}_{i=1}^{n} ID_i\mathsf{T}_i>;\ \mathrm{O}::<\mathop{\mathrm{R}}_{j=1}^{m} ID_j\mathsf{T}_j>;\ \mathrm{CLM}::<\mathop{\mathrm{R}}_{k=1}^{q} ClmID_k\mathsf{ST}_k>) \quad (12)$$

known as the *metaprocesses*. In this approach, complex processes are treated as derived processes from these metaprocesses, based on a set of algebraic process composition rules known as the *process relations.*

Definition 14. *A metaprocess in RTPA is a primitive computational operation that cannot be broken down to further individual actions or behaviors.*

A metaprocess is an elementary process that serves as a basic building block for modeling software behaviors. *Complex processes* can be composed from metaprocesses using *process relations*. In RTPA, a set of 17 metaprocesses has been elicited as shown in Table 2, from essential and primary computational operations commonly identified in existing formal methods and modern programming languages (Aho, Sethi, & Ullman, 1985; Higman, 1977; Hoare et al., 1987; Louden, 1993; Wilson & Clark, 1988; Woodcock & Davies, 1996). Mathematical notations and syntaxes of the metaprocesses are formally described in Table 2, while formal semantics of the metaprocesses of RTPA may be found in Wang (2006c, 2008a).

Lemma 2. *The RTPA metaprocess system* $\mathfrak{P}$ *encompasses 17 fundamental computational operations elicited from the most basic computing needs, i.e.:*

$$\mathfrak{P} = \{:=, \blacklozenge, \Rightarrow, \Leftarrow, \nLeftarrow, \gtrdot, \lessdot, |\gtrdot, |\lessdot, @, \triangleq, \uparrow, \downarrow, !, \otimes, \boxtimes, \S\} \quad (13)$$

As shown in Lemma 2 and Table 2, each metaprocess is a basic operation on one or more operands such as variables, memory elements, or I/O ports. Structures of the operands and their allowable operations are constrained by their types, as described in previous sections.

It is noteworthy that not all generally important and fundamental computational operations, as shown in Table 2, had been explicitly identified in conventional formal methods (e.g., the evaluation, addressing, memory allocation/release, timing/duration, and the system processes). However, all these are found necessary and essential in modeling system architectures and behaviors (Wang, 2007d).

ALGEBRAIC PROCESS OPERATIONS IN RTPA

The metaprocesses of RTPA developed in the preceding section identified a set of fundamental elements for modeling the most basic behaviors of computing and intelligent systems. It is interesting to realize that there is only a small set of 17 metaprocesses in system modeling. However, via the combination of a number of the metaprocesses by certain algebraic operations, any architecture and behavior of real-time or nonreal-time systems

Table 2. RTPA Metaprocesses

No.	Meta Process	Notation	Syntax
1	Assignment	:=	$y\mathbb{T} := x\mathbb{T}$
2	Evaluation	$\blacklozenge$	$\blacklozenge_{\mathbb{T}} exp\mathbb{T} \rightarrow \mathbb{T}$
3	Addressing	$\Rightarrow$	$id\mathbb{T} \Rightarrow \text{MEM}[ptr\mathbb{P}]\ \mathbb{T}$
4	Memory allocation	$\Leftarrow$	$id\mathbb{T} \Leftarrow \text{MEM}[ptr\mathbb{P}]\ \mathbb{T}$
5	Memory release	$\nLeftarrow$	$id\mathbb{T} \nLeftarrow \text{MEM}[\perp]\mathbb{T}$
6	Read	$\gtrdot$	$\text{MEM}[ptr\mathbb{P}]\mathbb{T} \gtrdot x\mathbb{T}$
7	Write	$\lessdot$	$x\mathbb{T} \lessdot \text{MEM}[ptr\mathbb{P}]\mathbb{T}$
8	Input	$\mid\gtrdot$	$\text{PORT}[ptr\mathbb{P}]\mathbb{T} \mid\gtrdot x\mathbb{T}$
9	Output	$\mid\lessdot$	$x\mathbb{T} \mid\lessdot \text{PORT}[ptr\mathbb{P}]\mathbb{T}$
10	Timing	$\underline{@}$	$@t\mathbb{TM} \underline{@} \S t\mathbb{TM}$ $\mathbb{TM} =$ yy:MM:dd | hh:mm:ss:ms | yy:MM:dd:hh:mm:ss:ms
11	Duration	$\triangleq$	$@t_n\mathbb{TM} \triangleq \S t_n\mathbb{TM} + \Delta n\mathbb{TM}$
12	Increase	$\uparrow$	$\uparrow(n\mathbb{T})$
13	Decrease	$\downarrow$	$\downarrow(n\mathbb{T})$
14	Exception detection	!	! (@*e*S)
15	Skip	$\otimes$	$\otimes$
16	Stop	$\boxtimes$	$\boxtimes$
17	System	§	§(*SysID*S$\mathbb{T}$)

can be sufficiently described (Wang, 2002b, 2003b, 2006a, 2007d).

Definition 15. *A process relation in RTPA is an algebraic operation and a compositional rule between two or more metaprocesses in order to construct a complex process.*

A set of 17 fundamental process relational operations has been elicited from fundamental algebraic and relational operations in computing in order to build and compose complex processes in the context of real-time software systems. Syntaxes and usages of the 17 RTPA process relations are formally described in Table 3. Deductive semantics of these process relations may be found in Wang (2006c, 2008a).

Lemma 3. *The software composing rules state that the RTPA process relation system $\mathfrak{R}$ encompasses 17 fundamental algebraic and relational operations elicited from basic computing needs, i.e.:*

$$\mathfrak{R} = \{\rightarrow, \curvearrowright, |, |\ldots|\ldots, R^*, R^+, R^i, \circlearrowleft, \rightarrowtail, \|, \oiint, |||, », \nleftrightarrow, \hookrightarrow_t, \hookrightarrow_e, \hookrightarrow_i\} \quad (14)$$

As modeled in Lemma 3 and Table 3, the first seven process relations—i.e., *sequential* (#1), *jump*

Table 3. RTPA process relations and algebraic operations

No.	Process Relation	Notation	Syntax
1	Sequence	$\rightarrow$	$P \rightarrow Q$
2	Jump	$\curvearrowright$	$P \curvearrowright Q$
3	Branch	|	$\begin{array}{l} \text{exp} \\ \mid \sim \rightarrow Q \end{array}$
4	Switch	$\begin{array}{c} \mid \\ \cdots \\ \mid \end{array}$	$\begin{array}{l} \text{exp} \\ \mathrm{i} \rightarrow P_i \\ \mid \sim \rightarrow \varnothing \\ \text{where } \mathbb{T} \in \{\mathrm{N}, \mathrm{Z}, \mathrm{B}, \mathrm{S}\} \end{array}$
5	While-loop	R^*	$\overset{\mathrm{F}}{\underset{\text{exp}=\mathrm{T}}{R}} P$
6	Repeat-loop	R^+	$P \rightarrow \overset{\mathrm{F}}{\underset{\text{exp}=\mathrm{T}}{R}} P$
7	For-loop	R^i	$\overset{n\mathrm{N}}{\underset{i\mathrm{N}=1}{R}} P(i\mathrm{N})$
8	Recursion	$\circlearrowleft$	$\overset{0}{\underset{i\mathrm{N}=n\mathrm{N}}{R}} P^{i\mathrm{N}} \circlearrowleft P^{i\mathrm{N}-1}$
9	Function call	$\rightarrowtail$	$P \rightarrowtail F$
10	Parallel	$\|\|$	$P \mid\mid Q$
11	Concurrence	$\fatslash\!\!\fatslash$	$P \fatslash\!\!\fatslash Q$
12	Interleave	$\|\|\|$	$P \mid\mid\mid Q$
13	Pipeline	»	$P \gg Q$
14	Interrupt	$\lightning$	$P \lightning Q$
15	Time-driven dispatch	$\hookrightarrow_t$	$@t_i \mathrm{TM} \hookrightarrow_t P_i$
16	Event-driven dispatch	$\hookrightarrow_e$	$@e_i \mathrm{S} \hookrightarrow_e P_i$
17	Interrupt-driven dispatch	$\hookrightarrow_i$	$@int_j \circledcirc \hookrightarrow_i P_j$

(#2), *branch* (#3), *switch* (#4), and *iterations* (#5 through #7)—may be identified as the Basic Control Structures (BCSs) of system behaviors (Aho et al., 1985; Hoare et al., 1987; Wilson & Clark, 1988). To represent the modern programming structural concepts, CSP (Hoare, 1985) identified the following seven additional process relations such as *recursion* (#8), *function call* (#9), *parallel* (#10), *concurrency* (#11), *interleave* (#12), *pipeline* (#13), and *interrupt* (#14). However, these process relations or operations were treated as the same of the metaprocesses in existing formal methods. That is, the conventional notation systems are not an algebraic production system rather than an exhaustive instruction system, which do not distinguish the basic computational operations and their composing rules.

RTPA (Wang, 2002b) extends the BCS's and process relations to *time-driven dispatch* (#15), *event-driven dispatch* (#16), *and interrupt-driven*

dispatch (#17) in order to model the top-level system behaviors, particularly those of real-time systems. The 17 process relations (BCSs) are regarded as the foundation of programming and system behavioral design, because any complex process can be combinatory implemented by the algebraic process composing operations onto the set of the 17 metaprocesses. In Table 3, the big-R used in process relations #5 through #8 is a special calculus recently created for denoting iterative and recursive behaviors of software systems (Wang, 2008b).

Theorem 2. *The express power of algebraic modeling states that the total number of the possible computational behaviors (operations) $\mathcal{N}$ is a set of combinations between two arbitrary metaprocesses $\mathbb{P}_1, \mathbb{P}_2 \in \mathfrak{P}$ composed by each of the process relations $\mathbb{R} \in \mathfrak{R}$ in RTPA, i.e.:*

$$\begin{aligned}\mathcal{N} &= \#\mathfrak{R} \bullet \mathbf{C}^2_{\#\mathfrak{P}} \\ &= 17 \bullet \frac{17!}{2!(17\text{-}2)!} \\ &= 17 \bullet 136 \\ &= 2{,}312\end{aligned} \tag{15}$$

Theorem 2 demonstrates the expressive power of the algebraic structure of RTPA towards computational behavior modeling and programming. It is noteworthy that an ordinary high-level programming language may introduce about 150 to 300 individual instructions. However, the expressive power of RTPA is much higher than those of programming languages and other exhaustive formal notation systems, although it just adopts a small set of 17 metaprocesses and 17 process relations.

MANIPULATION OF COMPUTATIONAL BEHAVIORS BY RTPA

As presented in previous sections, RTPA provides a neat and powerful denotational mathematics structure, which is capable to be used as a generic notation system for system architecture and behavior modeling and specifications. This section describes the usage and methodology of RTPA for software system modeling and refinement. Its applications in formally modeling intelligent systems and human cognitive processes will be presented in the next section.

The Universal Mathematical Model of Programs Based on RTPA

Program modeling is on coordination of computational behaviors with given data objects. On the basis of RTPA, a generic program model can be described by a formal treatment of statements, processes, and complex processes from the bottom up in the program hierarchy.

Definition 16. *A process P is the basic unit of an applied computational behavior that is composed by a set of statements s_i, $1 \le i \le n\text{-}1$, with left-associated cumulative relations, i.e.:*

$$\begin{aligned}P &= \mathop{R}_{i=1}^{n-1} (s_i\ r_{ij}\ s_j),\ j = i+1 \\ &= (...(((s_1)\ r_{12}\ s_2)\ r_{23}\ s_3)\ ...\ r_{n-1,n}\ s_n)\end{aligned} \tag{16}$$

where $s_i \in P$ and $r_{ij} \in R$.

With the formal process model as defined above, the universal mathematical model of programs can be derived below.

Definition 17. *A program $\wp$ is a composition of a finite set of m processes according to the time-,*

event-, and interrupt-based process dispatching rules of RTPA, i.e.:

$$\wp = \mathop{R}_{k=1}^{m}(@e_k \hookrightarrow P_k) \tag{17}$$

Equations 16 and 17 indicate that a program is an *embedded relational algebraic* entity, where a statement *s* in a program is an instantiation of a metainstruction of a programming language that executes a basic unit of coherent function and leads to a predictable behavior.

Theorem 3. *The Embedded Relational Model (ERM) states that a software system or a program $\wp$ is a set of complex embedded relational processes, in which all previous processes of a given process form the context of the current process, i.e.:*

$$\begin{aligned}\wp &= \mathop{R}_{k=1}^{m}(@e_k \hookrightarrow P_k) \\ &= \mathop{R}_{k=1}^{m}[@e_k \hookrightarrow \mathop{R}_{i=1}^{n-1}(p_i(k) \rightarrow_{ij} k)\, p_j(k))],\ j = i+1\end{aligned} \tag{18}$$

The ERM model presented in Theorem 3 reveals that a program is a finite and nonempty set of embedded binary relations between a current statement and *all previous ones* that formed the *semantic context* or environment of computing. Theorem 3 provides a unified software model, which is a formalization of the well accepted but informal process metaphor for software systems in computing.

ADT Modeling and Specification in RTPA

Abstract data types (ADTs) are perfect software architectures at component level that will be used to explain the modeling methodology of computing architectures and behaviors in RTPA. An ADT is a logical model of complex and/or user defined data type with a set of predefined operations on it. A queue as a typical ADT is presented in this subsection to demonstrate how the RTPA notation system is used to model and specify the architecture, static behaviors, and dynamic behaviors of software systems.

There are a number of approaches to the specification of ADTs, which can be classified into the logical and algebraic approaches. The logic approach is good at specifying the static properties of ADT operations, usually in forms of preconditions and post-conditions of operations of ADTs; while the algebraic approach is good at describing dynamic and run-time behaviors of ADTs.

A queue in RTPA is modeled as an algebraic entity, which has predefined operations on a set of data objects in given types. Unlike the conventional approaches to ADT specifications that treat ADTs as static data types, ADTs in RTPA are treated as dynamic finite state machines, which have both architectures and behaviors, in order to model both structural and operational components in system design.

Architectural Modeling in RTPA

At the top level, an RTPA specification of the queue, *Queue*ST, has three parallel facets, which are architecture, static behaviors, and dynamic behaviors, as given below.

$$\begin{aligned}\text{QueueST} \triangleq\ & \text{QueueST.Architecture} \\ & \| \text{QueueST.StaticBehaviors} \\ & \| \text{QueueST.DynamicBehaviors}\end{aligned} \tag{19}$$

Then, QueueST can be further refined by detailed specifications according to the RTPA methodology (Wang, 2007d).

The key modeling methodology for system architectures in RTPA is CLMs, as described in the subsection of advanced types. Any system architecture and data object, including their control structures, internal logic model, and hardware

interface model can be rigorously described and specified by using a set of CLMs.

Example 1. *The architecture of Queue*ST *modeled in RTPA is given in Figure 1, where both the architectural CLM and an access model are provided for the Queue.*

In Figure 1, the *access model* of QueueST is a logic model for supporting external invocation of the QueueST in operations, such as enqueue and service. The other parts of the model are designed for internal manipulation of the QueueST, such as creation, memory allocation, and memory release.

Static Behavior Modeling in RTPA

System static behaviors in RTPA are valid operations of systems that can be determined at compile-time, which describe the configuration of processes of the component and their relations. The set of static behaviors of QueueST can be modeled by a set of behavioral processes encompassing *create, release, enqueue, serve, clear, empty test,* and *full test.*

Example 2. *The detailed specification of one of the Queue's static behaviors, Queue*ST.*serve, is given in Figure 2.*

Contrasting the static behavior model of QueueST.serve in RTPA as shown in Figure 2 and in conventional propositional logic (Stubbs & Webre, 1985), the advances of RTAP method and notations may be well demonstrated. Among them, the most important advantage is that an RTPA model may be seamlessly refined into code

Figure 1. The architectural model of the Queue in RTPA

```
Queue ST.Architecture ≙  CLM : ST
                         || AccessModel : ST
                         || Events : S
                         || Status : BL

QueueST.Architecture.CLM ≙ QueueIDS :
        ( <Size : N | SizeN ≥ 0>,
          <Element : RT>,
          <CurrentPos : P | 0 ≤ CurrentPosP ≤ SizeN-1>
        )

Queue ST.Architecture.AccessModel ≙ QueueIDS(CurrentPosP)RT
```

Figure 2. The static behavioral model of the Queue in RTPA

```
QueueST.Serve (<I ::  QueueInstS>;
               <O :: ⓢQueueID.ServedBL, ElementRT>)  ≙
{
  QueueIDS := QueueInstS
  → ( ◆ ⓢQueueExistBL = T  ∧ CurrentPosP  > 0
        → (QueueID(1))RT ≻Element RT
        → QueueID(i))RT ≻QueueID(i-1)RT
        → ↓ (QueueID.CurrentPosP)
        → ⓢQueueID.ServedBL := T
    | ◆ ~
        → ⓢQueueID.ServedBL := F
        → ! (@'QueueIDExistBL = F  ∨  QueueEmptyBL = T')
    )
}
```

in a programming language in the succeeding phase of software engineering.

Dynamic Behavior Modeling in RTPA

System dynamic behaviors in RTPA are process relations determined at run time. According to the RTPA system modeling and refinement scheme, models of system static behaviors are process models of the system. To put the component processes into a live and interacting system, the dynamic behaviors of the system in terms of process deployment and dispatch are yet to be specified.

Example 3. *The dynamic behaviors of Queue*ST *are specified in RTPA as shown in Figure 3, where the process dispatch mechanisms of the Queue specifies detailed dynamic process relations at run time by a set of event-driven relations.*

Figures 1 through 3 model an ADT, *Queue*ST, in a coherent system from three facets. With the RTPA specification and refinement methodology and the expressive power of the RTPA notation system, the features of ADTs as both static data types and dynamic finite machines can be specified formally and precisely.

MANIPULATION OF INTELLIGENT BEHAVIORS BY RTPA

RTPA may be used not only for modeling and description of computing behaviors but also for modeling and denoting the cognitive processes of the brain in cognitive informatics (Wang, 2002a, 2003a, 2003b, 2006a, 2006b, 2007a, 2007e, 2007f; Wang & Wang, 2006; Wang et al., 2006). A formal treatment of memorization as a cognitive process is presented in this section by a rigorous RTPA model based on the cognitive model of human memorization (Wang, 2007a).

The Cognitive Process of Memorization

Memorization as a cognitive process can be described by two phases: the *establishment* phase and the *reconstruction* phase. The former represents the target information in the form of an object-attribute-relation (OAR) model (Wang, 2007c) and creates a memory in long-term memory (LTM). The letter retrieves the memorized information and reconstructs it in the form of a concept in the short-term memory (STM). Therefore, memorization can be perceived as the transformation of information and knowledge between STM and LTM, where the forward transformation from STM to LTM is for memory establishment, and

Figure 3. The dynamic behavioral model of the Queue in RTPA

QueueST**.DynamicBehaviors** $\triangleq$ { § $\rightarrow$
(@CreateQueueS ↳ Queue.Create (<**I::** QueueInstS, ElementInstRT, SizeInstN>;
<**O::** ⓢQueueID.AllocatedBL, ⓢQueueID.ExistBL>)
| @ReleaseQueueS ↳ Queue.Release (<**I::** QueueInstS>; <**O::** ⓢQueueID.ReleasedBL>)
| @EnqueueS ↳ Queue.Enqueue (<**I::** QueueInstS, ElementInstRT>; <**O::** ⓢQueueID.EnqueuedBL>)
| @ServeS ↳ Queue.Serve (<**I::** QueueInstS>; <**O::** ⓢQueueID.ServedBL, ElementRT>)
| @ClearS ↳ Queue.Clear (<**I::** QueueInstS>; <**O::** ⓢQueueID.ClearedBL>)
| @QueueEmptyS ↳ Queue.EmptyTest (<**I::** QueueInstS>; <**O::** ⓢQueueID.FullBL>)
| @QueueFullS ↳ Queue.FullTest (<**I::** QueueInstS>; <**O::** ⓢQueueID.FullBL>)
) $\rightarrow$ §
}

the backward transformation from LTM to STM is for memory reconstruction.

Algorithm 1. *The cognitive process of memorization can be carried out by the following steps:*

(0) Begin.
(1) Encoding: This step generates a representation of a given concept by transferring it into a sub-OAR model;
(2) Retention: This step updates the entire OAR in LTM with the sub-OAR for memorization by creating new synaptic connections between the sub-OAR and the entire OAR;
(3) Rehearsal test: This step checks if the memorization result in LTM needs to be rehearsed. If yes, it continues to practice Steps (4) and (5); otherwise, it jumps to Step (7);
(4) Retrieval: This step retrieves the memorized object in the form of sub-OAR by searching the entire OAR with clues of the initial concept;
(5) Decoding: This step transfers the retrieved sub-OAR from LTM into a concept and represents it in STM;
(6) Repetitive memory test: This step tests if the memorization process was succeeded or not by comparing the recovered concept with the original concept. If need, repetitive memorization will be called.
(7) End.

It is noteworthy that the input of memorization is a structured concept formulated by learning or other cognitive processes (Wang, 2007d, 2007f; Wang et al., 2006).

Formal Description of the Memorization Process in RTPA

The cognitive process of memorization described in Algorithm 1 can be formally modeled using RTPA as given in Figure 4. According to the LRMB model (Wang et al., 2006) and the OAR model (Wang, 2007c) of internal knowledge representation in the brain, the input of the memorization process is a structured concept c(OS, AS, RS)ST, which will be transformed to update the entire OAR model of knowledge in LTM in order to create a permanent memory. Therefore, the output of memorization is the updated OAR'ST in LTM.

In the RTPA memorization process, as shown in Figure 4, the *encoding* subprocess is modeled as a function that maps the given concept *c*ST into a sub-OAR, sOARST. The *retention* subprocess composes the sOARST with the entire OARST in LTM that maintains the whole knowledge of an individual. In order to check the memorization quality, rehearsals are usually needed. In a rehearsal, the *retrieval* subproecss searches a related sOARST in LTM by giving clues of previously memorized objects and attributes in cST. Then, the *decoding* subprocess transfers the sOARST into a recovered concept c'ST. In the repetitive memory test subprocess, the reconstructed c'ST will be compared with the original input of cST in order to determine if further memorization is recursively needed.

According to the 24-hour law of memorization as stated in Wang and Wang (2006), the memorization process may be completed with a period at least 24 hours by several cycles of repetitions. Although almost all steps in the process shown in Figure 4 are conscious, the key step of *retention* is subconscious or nonintentionally controllable. Based on the LRMB model (Wang et al., 2006), the memorization process is closely related to learning (Wang, 2007f). In other words, memorization is a back-end process of learning, which retains learning results in LTM and retrieves them when rehearsals or applications are needed. The retrieve process is search-based by contents or sOARST matching.

The cognitive process of memorization formally modeled in RTPA provides a rigorous description of one of the important and complicated mental processes of the brain. This case study explains the second usage of RTPA in intelligent

Figure 4. Formal description of the memorization process in RTPA

The Memorization Process

Memorization (**I::** c(OS, AS, RS)ST; **O::** OAR'ST) ≜
{I. Encoding
c(OS, AS, RS)ST ⇒ sOARST
// Concept representation

II. Retention

→ OAR'ST := OARST ⊎ sOARST
// Update OARST in LTM

III. Rehearsal

→ ◆ RehearsalBL = T
(IV. Retrieval
↣ Search (**I::** OARST;
O:: sOARST | (OS, AS, RS)ST ⊆ OARST))
// Retrieval sOARST in LTM

V. Decoding
→ (sOARST → c'(OS, AS, RS)ST)
// Concept reconstruction
)

VI. Repeat

→ ◆ (c'(OS, AS, RS)ST) ~ c(OS, AS, RS)ST)
→ ⊗ // Memorization succeed
| ◆ ~ // Retry
→ Memorization (**I::** c(OS, AS, RS)ST; **O::** OAR'ST)
}

system modeling and human behavioral manipulation. Further models and applications of RTPA in intelligent system modeling may be referred to (Wang, 2007d; Wang & Ngolah, 2002, 2003). The applications of RTPA in modeling cognitive processes of the brain and natural intelligence may be found in Wang (2003b, 2007a, 2007f).

CONCLUSION

RTPA has been developed as a denotational mathematical means, which can be used as algebra-based, expressive, easy-to-comprehend, and language-independent notation system, and a practical specification and refinement method for software and intelligent system modeling. RTPA is capable to support top-down software system design and implementation by algebraic modeling and seamless refinement methodologies. The RTPA methodology covers the entire system lifecycle from high-level design to code generation in a coherent algebraic notation system.

This article has demonstrated that RTPA is not only useful as a generic notation and methodology for computing and software system modeling but is also good at modeling human cognitive processes and intelligent systems. A number of case studies on large-scale software system modeling and specifications have been carried out, such as the Telephone Switching System (TSS) (Wang, 2003b), the Lift Dispatching System (LDS) (Wang & Ngolah, 2002), and the Automated Teller Machine (ATM) (Wang & Zhang, 2003). The application results have encouragingly demonstrated that RTPA is a powerful and practical algebraic

system for both academics and practitioners in software and intelligent system engineering.

A set of support tools for RTPA have been developed (Ngolah, Wang, & Tan, 2005b, 2006; Tan & Wang, 2006; Tan, Wang, & Ngolah, 2004a, 2004b, 2005, 2006), which encompasses the RTPA parser, type checker, and code generator in C++ and Java. The RTPA code generator enables system specifications in RTPA to be automatically translated into fully executable code. The RTPA tools will support system architects, analysts, and practitioners for developing consistent and correct specifications and architectural models of large-scale software and intelligent systems, and the automatic generation of code based on formal models and rigorous specifications in the denotational mathematical notations.

ACKNOWLEDGMENT

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The author would like to thank anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Aho, A. V., Sethi, R. & Ullman, J. D. (1985). *Compilers: Principles, techniques, and tools.* New York: Addison-Wesley.

Baeten, J. C. M., Bergstra, J. A. (1991). Real time process algebra. *Formal Aspects of Computing, 3,* 142-188.

Boucher, A., & Gerth, R. (1987). A timed model for extended communicating sequential processes. In *Proceedings of ICALP'87* (LNCS 267). Springer.

Cardelli, L., & Wegner, P. (1985). On understanding types, data abstraction and polymorphism. *ACM Computing Surveys, 17*(4), 471-522.

Cerone, A. (2000). *Process algebra versus axiomatic specification of a real-time protocol* (LNCS 1816, pp. 57-67). Berlin, Germany: Springer.

Corsetti, E., Montanari, A., & Ratto, E. (1991). Dealing with different time granularities in formal specifications of real-time systems. *The Journal of Real-Time Systems, 3*(2), 191-215.

Dierks, H. (2000). A process algebra for real-time programs (LNCS 1783, pp. 66/76). Berlin, Germany: Springer.

Fecher, H. (2001). A real-time process algebra with open intervals and maximal progress, *Nordic Journal of Computing, 8*(3), 346-360.

Gerber, R., Gunter, E. L., & Lee, I. (1992). Implementing a real-time process algebra In M. Archer, J. J. Joyce, K. N. Levitt, & P. J. Windley (Eds.), *Proceedings of the International Workshop on the Theorem Proving System and its Applications* (pp. 144-145). Los Alamitos, CA: IEEE Computer Society Press.

Higman, B. (1977). *A comparative study of programming languages* (2nd ed.). MacDonald.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM, 21*(8), 666-677.

Hoare, C. A. R. (1985). *Communicating sequential processes.* London: Prentice Hall International.

Hoare, C. A. R., Hayes, I. J., He, J., Morgan, C. C., Roscoe, A. W., Sanders, J. W., et al. (1987). Laws of programming, *Communications of the ACM, 30*(8), 672-686.

Jeffrey, A. (1992). Translating timed process algebra into prioritized process algebra. In J. Vytopil (Ed.), *Proceedings of the Second International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems* (LNCS 571, pp. 493-506). Nijmegen, The Netherlands: Springer-Verlag.

Klusener, A. S. (1992). Abstraction in real time process algebra. In J. W. de Bakker, C. Huizing, W. P. de Roever, & G. Rozenberg (Eds.), *Proceedings of Real-Time: Theory in Practice* (LNCS, pp. 325-352). Berlin, Germany: Springer.

Louden K. C. (1993). *Programming languages: Principles and practice*. Boston.: PWS-Kent.

Martin-Lof, P. (1975). An intuitionistic theory of types: Predicative part. In H. Rose & J. C. Shepherdson (Eds.), *Logic Colloquium 1973*. North-Holland.

Milner, R. (1980). *A calculus of communicating systems* (LNCS 92). Springer-Verlag.

Milner, R. (1989). *Communication and concurrency*. Englewood Cliffs, NJ: Prentice Hall.

Mitchell, J. C. (1990). Type systems for programming languages. In J. van Leeuwen (Ed.), *Handbook of theoretical computer science* (pp.365-458). North-Holland.

Nicollin, X., & Sifakis, J. (1991). An overview and synthesis on timed process algebras. In *Proceedings of the Third International Computer Aided Verification Conference* (pp. 376-398).

OMG. (2002, July). *IDL syntax and semantics*. 1-74.

Reed, G. M., & Roscoe, A. W. (1986). A timed model for communicating sequential processes. In *Proceedings of ICALP'86* (LNCS 226). Berlin, Germany: Springer-Verlag.

Russell, B. (1961). *Basic writings of Bertrand Russell*. London: George Allen & Unwin Ltd.

Schneider, S. A. (1991). *An operational semantics for timed CSP* (Programming Research Group Tech. Rep. TR-1-91). Oxford University.

Schilpp, P. A. (1946). The philosophy of Bertrand Russell. *American Mathematical Monthly, 53*(4), 7210.

Stubbs, D. F., & Webre, W. R. (1985). *Data structures with abstract data types and Pascal*. Monterey, CA: Brooks/Cole.

Tan, X., Wang, Y., & Ngolah, C. F. (2004). A novel type checker for software system specifications in RTPA. In *Proceedings of the 17th Canadian Conference on Electrical and Computer Engineering (CCECE'04)*.(pp. 1549-1552). Niagara Falls, Ontario, Canada: IEEE CS Press.

Tan, X., Wang, Y., & Ngolah, C. F. (2006). Design and implementation of an automatic RTPA code generator. In *Proceedings of the 19th Canadian Conference on Electrical and Computer Engineering (CCECE'06)* (pp. 1605-1608). Ottawa, Ontario, Canada: IEEE CS Press.

van Heijenoort, J. (1997). *From Frege to Godel, a source book in mathematical logic 1879-1931*. Cambridge, MA: Harvard University Press.

Vereijken, J. J. (1995). A process algebra for hybrid systems. In A. Bouajjani & O. Maler (Eds.), In *Proceedings of the Second European Workshop on Real-Time and Hybrid Systems*. Grenoble: France.

Wang, Y. (2002a). On cognitive informatics (Keynote speech). In *Proceedings of the First IEEE International Conference on Cognitive Informatics (ICCI'02)* (pp. 34-42). Calgary, Canada: IEEE CS Press.

Wang, Y. (2002b). The real-time process algebra (RTPA). *Annals of Software Engineering: An International Journal, 14*, 235-274.

Wang, Y. (2003a). On cognitive informatics. *Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy, 4*(3), 151-167.

Wang, Y. (2003b). Using process algebra to describe human and software system behaviors. *Brain and Mind, 4*(2), 199-213.

Wang, Y. (2006a). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation (Invited plenary talk). In *Proceedings of the First International Conference on Rough Set and Knowledge Technology (RSKT'06)* (LNAI 4062, pp. 69-78). Chongqing, China: Springer.

Wang, Y. (2006b). Cognitive informatics—Towards the future generation computers that think and feel (Keynote speech). In *Proceedings of the Fifth IEEE International Conference on Cognitive Informatics (ICCI'06)* (pp. 3-7). Beijing, China: IEEE CS Press.

Wang, Y. (2006c). On the informatics laws and deductive semantics of software. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 161-171.

Wang, Y. (2007a). Formal description of the cognitive process of memorization. In *Proceedings of the Sixth International Conference on Cognitive Informatics (ICCI'07)* (pp. 284-293). Lake Tahoe, CA: IEEE CS Press.

Wang, Y. (2007b). Keynote speech, on theoretical foundations of software engineering and denotational mathematics,. In *Proceedings of the Fifth Asian Workshop on Foundations of Software* (pp. 99-102). Xiamen, China.

Wang, Y. (2007c). The OAR Model of neural informatics for internal knowledge representation in the brain. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(3), 64-75.

Wang, Y. (2007d). *Software engineering foundations: A software science perspective.* In *CRC series in software engineering: Vol. 2.* CRC Press.

Wang, Y. (2007e). The theoretical framework of cognitive informatics. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(1), 1-27.

Wang, Y. (2007f). The Theoretical framework and cognitive process of learning. In *Proceedings of the Sixth International Conference on Cognitive Informatics (ICCI'07)* (pp. 470-479). Lake Tahoe, CA: IEEE CS Press.

Wang, Y. (2008a). Deductive semantics of RTPA. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 95-121.

Wang, Y. (2008b). On the big-R notation for describing iterative and recursive behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(1), 17-28.

Wang, Y., & King, G. (2000). Software engineering processes: Principles and applications In *CRC Series in Software Engineering: Vol. I.*. CRC Press.

Wang, Y., & Noglah, C. F. (2002). Formal specification of a real-time lift dispatching system. In *Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'02)* (pp.669-674). Winnipeg, Manitoba, Canada.

Wang, Y., & Noglah, C. F. (2003). Formal description of real-time operating systems using RTPA. In *Proceedings of the 2003 Canadian Conference on Electrical and Computer Engineering (CCECE'03)* (pp.1247-1250). Montreal, Canada: IEEE CS Press

Wang, Y., & Wang, Y. (2006). Cognitive informatics models of the brain. *IEEE Transactions on Systems, Man, and Cybernetics* (C), *36*(2),203-207.

Wang, Y., Wang, Y., Patel, S., & Patel, D. (2006). A layered reference model of the brain (LRMB). *IEEE Transactions on Systems, Man, and Cybernetics* (C), *36*(2),124-133.

Wang, Y., & Zhang, Y. (2003). Formal description of an ATM system by RTPA. In *Proceedings of the 16th Canadian Conference on Electrical and Computer Engineering* (CCECE'03) (pp. 1255-1258). Montreal, Canada: IEEE CS Press.

Wilson, L. B., & Clark, R. G. (1988). *Comparative programming language*. Wokingham, England: Addison-Wesley.

Woodcock, J., & Davies, J. (1996). *Using Z: Specification, refinement, and proof*. London: Prentice Hall International.

This work was previously published in the International Journal of Cognitive Informatics and Natural Intelligence, Vol. 2, Issue 2, edited by Y. Wang, pp. 44-62, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.20
Measuring the Efficiency of Free and Open Source Software Projects Using Data Envelopment Analysis

Stefan Koch
Vienna University of Economics and Business Administration, Austria

ABSTRACT

In this chapter, we propose for the first time a method to compare the efficiency of free and open source projects, based on the data envelopment analysis (DEA) methodology. DEA offers several advantages in this context, as it is a non-parametric optimization method without any need for the user to define any relations between different factors or a production function, can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Using a data set of 43 large F/OS projects retrieved from SourceForge.net, we demonstrate the application of DEA, and show that DEA indeed is usable for comparing the efficiency of projects. We will also show additional analyses based on the results, exploring whether the inequality in work distribution within the projects, the licensing schem,e or the intended audience have an effect on their efficiency. As this is a first attempt at using this method for F/OS projects, several future research directions are possible. These include additional work on determining input and output factors, comparisons within application areas, and comparison to commercial or mixed-mode development projects.

INTRODUCTION

In the last years, free and open source software (also sometimes termed libre software) has gathered increasing interest, both from the business and academic world. As some projects in different application domains like most notably the operating system Linux together with the suite of GNU utilities, the office suites GNOME and KDE, Apache, sendmail, bind, and several programming languages have achieved huge success in their respective markets, both the adoption by

commercial companies, and also the development of new business models by corporations both small and large like Netscape or IBM have increased.

Currently, any comparison of free and open source (F/OS) software projects is very difficult. There is increased discussion on how the success of F/OS projects can be defined (Stewart, 2004; Stewart and Ammeter, 2002; Crowston et al., 2004; Crowston et al., 2003), using for example search engine results as proxies (Weiss, 2005). In addition, the process applied in these projects can differ significantly.

In this paper, we propose to compare F/OS projects according to their efficiency in transforming inputs into outputs. For any production process, this efficiency and productivity is a key indicator in comparison to other processes. For F/OS projects, two levels of analysis are of interest: The F/OS process in general is different to commercial software development processes, and the process variance between F/OS projects is also high. In both cases, a main difference, and a main argument for adopting a process or elements from it, is the efficiency. Neither a commercial enterprise, nor an F/OS project would knowingly and willingly waste scarce resources by using an inefficient development process. This necessitates to compute and compare the efficiency of F/OS projects to gain an understanding of the results any process decision has on the outputs, which could lead to identifying best practices, and thus to increasing the overall efficiency and thus output of all projects.

To this end, we propose to apply the method of Data Envelopment Analysis (DEA), which is a non-parametric optimization method for efficiency comparisons without any need for the user to define any relations between different factors or a production function. In addition, DEA can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Efficiency and productivity in software development is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code (Park, 1992) or, preferably due to independence from programming language, function points (Albrecht and Gaffney, 1983). While this approach can be problematic in an environment of commercial software development as well due to missing components especially of the output, for example also Kitchenham and Mendes (2004) agree that productivity measures need to be based on multiple size measures, there are additional problems in the context of F/OS development which point towards DEA as an appropriate method.

In F/OS projects, normally the effort invested is unknown, and therefore might need to be estimated (Koch, 2004; Koch, 2005), and is also more diverse than in commercial projects, as it includes core team member, committers, bug reporters and several other groups with varying intensity of participation. Besides that, also the outputs can be more diverse. In the general case, the inputs of an F/OS project can encompass a set of metrics, especially concerned with the participants. So, in the simplest case, the number of programmers and other participants can be used. The output of a project can be measured using several software metrics like most easily the number of LOC, files, checkins to the source code control system, postings, bug reports, characteristics of development speed (e.g. coefficients of a software evolution equation estimated) or even metrics for product attributes like McCabe's cyclomatic complexity (McCabe, 1976) or object-oriented metrics, e.g. the Chidamber-Kemerer suite (Chidamber and Kemerer, 1994). This range of metrics both for inputs and outputs, and their different scales necessitates application of an appropriate method, which DEA can be.

The main result of applying DEA for a set of projects is an efficiency score for each project. This score can serve different purposes as mentioned above: First, single projects can be compared accordingly, but also groups of projects, for example those following similar process models,

located in different application domains or simply of different scale can be compared to determine whether any of these characteristics lead to higher efficiency. Lastly, this method could also be used for a comparison with traditional, commercial or even hybrid projects, which in addition to volunteer resources also feature monetary input from an organization. In this way, important questions concerning software development in general and especially in F/OS projects can be answered: Which methods allow for the most efficient application of the available manpower? Is F/OS software development a new, more efficient way of producing software? Are projects on a larger scale more efficient than those on smaller scale? Is a high concentration of work on a small number of heads efficient?

In this paper, we will demonstrate a first application of DEA to compare the efficiency of a set of F/OS projects. After a short introduction to F/OS, we will describe the method of DEA in general. Based on this, the application will be detailed, starting with a description of the methodology of data retrieval and the respective data set to be used. We will then discuss how to set up the DEA model in this case, taking into account the available data, and give the results. In a section on future research, possible enhancements to this work will be given, in addition to applications of the results obtained.

FREE AND OPEN SOURCE SOFTWARE

For a quick definition of free and open source software, both terms needs to be analyzed (Laurent, 2004; Dixon, 2003; Rosen, 2004). The term open source as used by the Open Source Initiative (OSI) is defined using the Open Source Definition (Perens, 1999), which lists a number of rights which a license has to grant in order to constitute an open source license. These include most notably free redistribution, inclusion of source code, to allow for derived works which can be redistributed under the same license, and integrity of author's source code. The Free Software Foundation (FSF) advocates the term free software (Stallman, 2002). A software is defined as free if the user has the freedom to run the program, for any purpose, to study how the program works, and adapt it to his needs, to redistribute copies and to improve the program, and release these improvements to the public. Access to the source code is a necessary precondition. In this definition, open source and free software are largely interchangeable. The GNU project itself prefers copylefted software, which is free software whose distribution terms do not let redistributors add any additional restrictions when they redistribute or modify the software. This means that every copy of the software, even if it has been modified, must be free software, a prescription embodied in the most well-known and important license, the GNU General Public License (GPL).

Not only is F/OS software unique in its licenses and legal implications, but also in its development process. The main ideas of this development model are described in the seminal work of Raymond (1999), 'The Cathedral and the Bazaar', in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development. In this, a large number of developer-turned users come together without monetary compensation (Raymond, 1999; Hertel et al., 2003) to cooperate under a model of rigorous peer-review and take advantage of parallel debugging that leads to innovation and rapid advancement in developing and evolving software products. In order to allow for this to happen and to minimize duplicated work, the source code of the software needs to be accessible which necessitates suitable licenses, and new versions need to be released often. Today, agile methods like eXtreme Programming or the strict release processes in place in several

open source projects (Holck & Jorgensen, 2004) give evidence to mixed forms of development. Currently, empirical research on similarities and dissimilarities between F/OS development and other development models is still proceeding (Mockus et al., 2002; Koch, 2004).

DATA ENVELOPMENT ANALYSIS

Production and Efficiency

The production of a good or service can be formalized using a production function, which gives the maximum possible output for a given amount of input (Varian, 2005). Several mathematical forms for production functions have been used in literature, including a linear form or a Cobb-Douglas production function, necessitating an appropriate number of parameters for defining the function in a concrete case. The notion of returns to scale is intimately linked to this concept: If the inputs are scaled by a constant factor, e.g. twice as much input is used, constant returns to scale would mean that the output is scaled by the same amount. If increasing returns to scale are present, the output would increase more than that, less with decreasing returns to scale (Varian, 2005). Of course, the returns to scale might be different at different levels of production, leading to an ideal size of production (or firm). The term of economies of scale refers to the decreased per unit cost as output increases. Both the production function to be used (Hu, 1997), and the returns to scale (Banker & Kemerer, 1989; Banker & Slaughter, 1997) have long been a topic of discussion in software development.

For comparing the efficiency of different firms (or units), the most basic approach is to compute the ratio of an output to an input. Those firms (or units) exhibiting the highest ration are able to produce the most output given an input, thus are most efficient. For the area of software development, efficiency or productivity is most often denoted as lines-of-code, or preferably function points (Albrecht and Gaffney, 1983) due to their technology neutrality, per person-month of effort. Problems with this approach are that it is computationally not usable for multi-input, multi-output situations, and that it assumes constant returns to scale.

Main Concepts of DEA

The principle of the border production function was introduced by Farell (1957) for measuring the technical efficiency and enhanced by Charnes, Cooper and Rhodes (1978a) into the first Data Envelopment Analysis model (the CCR model). The term DEA is used for the first time in an evaluation of school programs for the support of handicapped children. The object of analysis the DEA considers is very generally termed Decision Making Unit (DMU). This term includes relatively flexibly each unit which is responsible for the transformation of inputs into outputs. As different applications show, this definition covers therefore hospitals, supermarkets, schools, bank branches and others.

The basic principle of DEA can be understood as a generalization of the normal efficiency evaluation as described above by means of the relationship from an output to an input into the general case of a multi-output, multi-input system without a any given conversion rates or same units for all factors. For the area of software development, this can be understood as adding for example pages of documentation to lines-of-code and function points as an output factor. In contrast to other approaches, which require the parametric specification of a production function, DEA measures production behavior directly and uses this data for the evaluation of all DMUs. The DEA derives a production function from mean relations between inputs and outputs (whereby it is only assumed that the relation is monotonous and concave), by determining the outside cover of all production relations, while for example a regression analysis

estimates a straight line through the centre of all production relations. The DEA identifies "best practicing" DMUs, which lie on the production border. Thus any outliers and measuring and / or data errors may exert a strong influence on the results, and therefore special attention is to be given to sensitivity analyses. In order to determine the production function, the efficient DMUs are linked in sections with one another. A DMU is understood as being efficient if none of the outputs can be increased, without either or several of the inputs increasing or other outputs being reduced, as well as none of the inputs can be reduced, without reducing either one or more outputs or increasing other inputs.

Computation

For each DMU an individual weighting procedure is used over all inputs and outputs. These form a weighted positive linear combination, whereby the weights are specified in such a way that they maximize the production relationship of the examined unit, in order to let these become as efficient as possible. The efficiency of an examined unit is limited with 1. That means it that no a-priori weightings are made by the user, and that the weights between the DMUs can be different. However, the definition of the relevant inputs and outputs is necessary. For each evaluation object the DEA supplies a solution vector of weighting factors and a DEA efficiency score. If this score is equal to 1, then the DMU is DEA efficient. In this context, DEA efficiency means that within the selected model variant no weighting vector could be found which would have led to a higher efficiency value. DEA efficient are thus all those DMUs which are not clearly DEA inefficient compared with the others. Any inefficiency can therefore not be ruled out completely. For inefficient DMUs weighting factors were found in the context of the selected model variant which resulted in a higher efficiency value in the case of at least another DMU.

For each inefficient DMU the DEA returns a set of efficient DMUs which exhibit a similar input/output structure and lie on the production border near to the inefficient DMU (this is also termed reference set or DEA benchmark). Using this information, an idea in which direction an increase in efficiency is possible can be gained. Because the relative efficiency measure is based on the distances from actually existing DMUs and is thus easily comprehensible, DEA is suitable as an analysis tool better than other methods.

Models of DEA

The first model of the DEA was introduced by Charnes, Cooper and Rhodes (1978b) and is designated with the initial letters of their surnames as CCR model. They pose four assumptions for the production possibility set, which are convexity (each linear combination of two DMUs results in a production possibility within the valid range), possibility for inefficient production (all production possibilities which are more inefficient than the known DMUs are permissible), constant returns to scale and minimum extrapolation (the production possibility range covers all observed DMUs). The CCR model measures the efficiency of each DMU by examining the relationship of the outputs to the inputs, optimally weighted for the DMU. This optimization happens under the constraint that equivalent conditions for no other DMU result in a value exceeding 1. By this maximization each DMU receives the optimum weighting given the constraint.

The different basic models of the DEA can be divided on the basis of two criteria: This is on the one hand the orientation of the model, on the other hand the underlying assumption regarding the returns to scale of the production process. With input-oriented models the reduction of the input vector maximally possible with the given manufacturing technology is determined, whereas with output-oriented models the maximally possible proportional increase of the output vector is

determined. Input orientation is generally present, if an enterprise function (for example manufacturing) is to be analyzed which minimizes the resources consumption, but can only affect the output in a limited way. The returns to scale can be assumed either as being constant as in the CCR model described above, or variable. With constant returns to scale size-induced productivity differences are considered into the efficiency evaluation, with variable returns to scale the differences are neutralized by the model. The most common example of a model with variable returns to scale is an advancement to the CCR model by Banker, Charnes and Cooper (1984), the BCC model. This model includes an additional measuring variable in the fundamental equation to capture rising, falling or constant returns to scale.

Applications

DEA models are generally used where scarce resources are to be used goal-oriented. The first DEA models were developed to measure the efficiency of non-profit units, for whose inputs and outputs no clear market prices exist and also otherwise no clear evaluation relations are present. The entire public sector therefore represents a typical area of application. Emphases of the first investigations were therefore health service, in particular hospitals, as well as school advancement programs (Charnes et al., 1978a). In further consequence the DEA was also applied to the evaluation of the efficiency of private business units as for instance branches of banks or supermarkets, especially if the efficiency is not only to be measured in profit, but also by parameters as for instance environmental condition, downtimes or customer satisfaction.

In the area of software development, DEA was so far only rarely applied. Banker and Kemerer (1989) use this approach in order to prove the existence of both rising and falling returns to scale. First arise in small, the others in larger software development projects. Based on published collections of project data records the authors compute in each case the point of the maximum productivity (most productive scale size) within the set of projects, starting from which the returns to scale begin to fall. Banker and Slaughter (1997) use the DEA in the area of maintenance and enhancement projects. It can be proven that rising returns to scale are present, which would have made a cost reduction of around 36 per cent possible when utilized. This was prevented however by organizational constraints, for example high punishments for exceeding the completion date. Mayrhauser et al. (2000) also report applying DEA on a data set consisting of 46 software projects from the NASA-SEL database to analyze objective variables and their impact on efficiency. They suggest combining this with the results of applying principal component analysis (PCA) in an analysis of subjective variables. An investigation of ERP projects was done by Myrtveit and Stensrud (1999). They used 30 SAP R/3-projects of a consulting firm for the application of the DEA. Kitchenham (2002) gives an in-depth discussion on the application of DEA in software development.

APPLICATION OF DEA FOR FREE AND OPEN SOURCE SOFTWARE PROJECTS

Methodology

For performing the proposed efficiency comparison of F/OS software projects, the information contained in software development repositories will be used. These repositories contain a plethora of information on the underlying software and the associated development processes (Cook et al., 1998; Atkins et al., 1999). Studying software systems and development processes using these sources of data offers several advantages (Cook et al., 1998): This approach is very cost-effective, as no additional instrumentation is necessary, and

it does not influence the software process under consideration. In addition, longitudinal data is available, allowing for analyses considering the whole project history.

Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems, bug reporting systems, or mailing lists. Many of these have already been used as information sources for closed source software development projects. For example, Cook et al. (1998) present a case study to illustrate their proposed methodology of analyzing in-place software processes. They describe an update process for large telecommunications software, analyzing several instances of this process using event data from customer request database, source code control, modification request tracking database and inspection information database. Atkins et al. (1999) use data from a version control system in order to quantify the impact of a software tool, a version-sensitive editor, on developer effort.

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels, as the participants in any project are not collocated. Therefore only a small amount of information can not be captured by repository analyses because it is transmitted inter-personally. As a side effect, the repositories in use must be available openly and publicly, in order to enable as many persons as possible to access them and to participate in the project. Therefore open source software development repositories form an optimal data source for studying the associated type of software development.

Given this situation, repository data have already been used in research on open source software development. This includes in-depth analyses of small numbers of successful projects like Apache and Mozilla (Mockus et al., 2002), GNOME (Koch and Schneider, 2002) using mostly information provided by version-control-systems, but sometimes in combination with other repository data like from mailing list archives. Large-scale quantitative investigations spanning several projects going into software development issues are not yet as common, and have mostly been limited to using aggregated data provided by software project repositories (Crowston and Scozzi, 2002; Hunt and Johnson, 2002; Krishnamurthy, 2002), meta-information included in Linux Software Map entries (Dempsey et al., 2002), or data retrieved directly from the source code itself (Ghosh and Prakash, 2000).

Data Collection and Data Set

For this efficiency comparison, a data set covering several projects was needed. Therefore, we used a subset of a data set already available from prior research derived from SourceForge.net, the software development and hosting site. The mission of SourceForge.net is 'to enrich the open source community by providing a centralized place for open source developers to control and manage open source software development'. To fulfill this mission goal, a variety of services is offered to hosted projects, including tools for managing support, mailing lists and discussion forums, web server space, shell services and compile farm, and source code control. While SourceForge.net publishes several statistics, e.g. on activity in their hosted projects, this information was not detailed enough for the proposed analysis. For example, Crowston and Scozzi (2002) used the available data for validating a theory for competency rallying, which suggests factors important for the success of a project. Hunt and Johnson (2002) have analyzed the number of downloads of projects occurring, and Krishnamurthy (2002) used the available data of the 100 most active mature projects for an analysis.

The data collection method utilized is described in detail in Hahsler and Koch (2005). It is based on automatically extracting data from the web pages and especially the source code control system, in the form of CVS, and subsequently

parsing the results and storing the relevant results, e.g. size, date and programmer of each checkin, in a database. Most of this work was done using Perl scripts. This resulted in a number of 8,621 projects for which all relevant information could be retrieved. More detailed analyses of these data can be found in Koch (2004). For this research, we chose to use the subset of largest projects, in order to have a limited set exhibiting similar characteristics. We therefore defined a threshold of at least five developers and 500,000 lines-of-code, which resulted in 43 projects. Table 1 gives descriptive statistics for these projects. While most of the variables are self-descriptive, a few notes are in order: The total number of programmers gives all different persons who have over the complete lifetime contributed code. In contrast, a programmer is defined to be active in a month if he showed activity during this time interval. The mean number of programmers per month is based on this definition, as are the cumulated active programmers. This last number can be seen as an effort indicator, denoting the sum of the active programmers for each month over the total lifetime. It is also given in person-years, although an F/OS programmer active in a given month will not necessarily have worked 40 hours per week. For example, Hertel et al. (2003) report that in the developer group of Linux kernel contributors participating in their survey, about 18.4 hours per week are spent on open source development by each person. This number could be used to convert the effort to "real" person-years, but, as only F/OS projects are to be compared here, this is not done. The status is an indicator assigned by the project administrator within SourceForge.net and aims at reflecting the phase of a project in the development lifecycle. It has seven possible values, reaching from planning, pre-alpha, alpha, beta to production/stable and mature, and to inactive. The last variable is used to depict the inequality of work distribution within the development team (Robles et al., 2004), with values nearer to 1 showing higher inequality.

Table 1. Descriptive statistics of data set (n=43)

	Min.	Max.	Mean	Std. Dev.
Size (LOC)	533321,00	7669643,00	2014731,09	2124643,99
Total of LOC added	541148,00	10773710,00	2478442,26	2447076,77
Total of LOC deleted	7827,00	3104067,00	463711,16	571624,21
Number of Files	904,00	23594,00	6381,79	5683,65
Start date	1990/08/09	2001/04/17	1999/01/16	n/a
Checkins	1897,00	133759,00	27485,40	26986,30
Total number of programmers	5,00	88,00	18,35	15,62
Mean number of programmers per month	1,40	13,00	4,39	2,86
Cumulated active programmers (person-years)	1,00	55,40	9,78	9,79
Number of months	5,00	133,00	27,44	22,12
Status	,00	6,00	4,21	1,55
Inequality (based on checkins)	,05	,87	,33	,24

DEA Model Definition

For computing DEA models, different software products are available, some of which are freeware. In this case, the program accompanying the book by Cooper, Seiford and Tone (2000) was used, which can solve different DEA models, input or output-oriented, as well as with constant or variable returns to scale.

The first choices to be taken concern the definition of input and output factors, as well as the model to be applied. Banker and Kemerer (1989) have demonstrated the existence of both increasing and decreasing returns to scale in software projects, also Myrtveit and Stensrud (1999) recommend to use a model with variable returns to scale. Using a data set of maintenance and extension projects Banker and Slaughter (1997) have found increasing returns to scale. Kitchenham (2002) gives an overview of research result and reasons for differences on economies and diseconomies of scale in software development. Regarding the orientation of the model, an output-orientation might seem more appropriate. Given a certain input which can be acquired, i.e. participants attracted, the output is to be maximized. According to this reasoning, the BCC-O model is applied.

Regarding the definition which factors are to be used as inputs and outputs, it is to be considered that with an increase in the number of factors more DMUs, i.e. projects, are estimated to be efficient, in particular if the database is small in relation. Also the availability of factors in the data set limits the possibilities. In this case, we selected to use the total number of programmers and their effort as inputs, and size, LOC added and deleted, files, checkins and status as output factors. Naturally, this selection is based on the available data, and could be changed. This point will be discussed in more detail within the section on future research.

Results

The results of a DEA analysis give for each DMU, in this case F/OS project, an efficiency measure as defined above, i.e. of value 1 if no possibilities for improvements in transforming inputs into outputs have been detected given the data. As the DEA model set up in this paper is output-oriented, any other value below 1 means that a proportional increase of outputs seems possible given the inputs of a project. A DEA efficiency of 0,75 would translate into a possible output increase of 25% for this project. In addition, a reference set is reported, which contains DMUs with similar input-output relations and lie on the production border near to the inefficient DMU. Table 2 gives an extract of the DEA results on project level from the data set, showing two different projects, one of which is deemed DEA efficient. As can be seen, project number 27, mysql, is DEA efficient, while number 3, berlin, has a lower efficiency score of

Table 2. Results from DEA on project level (extract)

No.	DMU	Score	Reference set					
...								
3	berlin	0,518	gkernel	0,211	mysql	0,597	python	0,192
...								
27	mysql	1,000	mysql	0,999				
...								

Table 3. Ranking of projects according to DEA efficiency

Rank	DMU	Score
1	aaf, crossfire, gkernel, lpr, musickit, mysql, nfs, python, u4x, winex	1,000
11	ceps, mesa3d, tcl	0,999
14	linuxsh	0,995
15	linux-apus	0,963
16	Xfce	0,954
17	linux-mac68k	0,903
18	jboss	0,896
19	firebird	0,878
20	cvsgui	0,866
21	enlightenment	0,860
22	quake	0,852
23	sdcc	0,852
24	zangband	0,847
25	opensta	0,840
26	lesstif	0,837
27	ghostscript	0,835
28	qpe	0,833
28	htdig	0,833
30	wf-gdk	0,829
31	dri	0,823
32	crystal	0,769
33	lsb	0,742
34	acs-pg	0,701
35	mahogany	0,691
36	squid	0,670
37	omseek	0,671
38	clocc	0,667
39	linux-vr	0,574

0,518 which translates into a respective possible increase in outputs. The reference set reported contains the efficient projects mysql, gkernel and python, which also have weights assigned relating to their relative importance in showing the inefficiency of berlin. In this case, mysql is the most important project, which means that it is most similar in input-output relation, and orienting at this project would show the most promising path for efficiency increase.

From all DEA efficiency scores, an efficiency ranking over all projects can be derived, which is given in Table 3 for the data set used here. As can be seen, several projects are deemed DEA efficient, therefore have a value of 1 and are ranked at the top. This list gives two major informations

Table 4. Overview of DEA results for data set

No. of DMUs	43
Average	0,828
Std. Dev.	0,196
Maximum	1,000
Minimum	0,085
Number of DEA-efficient DMUs	10
Frequency in Reference Set	
Peer set	**Frequency to other DMUs**
aaf	4
crossfire	8
gkernel	24
lpr	3
musickit	3
mysql	28
nfs	2
python	18
u4x	1
winex	0

per project: Their rank is a measure of the standing within the total group, while the efficiency score gives an indication of the amount of output increase possible.

For an overview of the results, see also Table 4. In this table, statistics on the efficiency scores in the total population are given. Overall, 10 different projects have been classified as DEA efficient (which is also visible from Table 3), the mean efficiency score with 0,828 seems relatively high. For each efficient project, the number of times it appears in the reference sets of non-efficient projects is also given. This can be used as an indicator of the relative importance of this project in determining efficiency scores, and would also be important for performing sensitivity analyses.

These results, besides demonstrating that differences in efficiency between F/OS projects do indeed exist, can then be used in several ways: First, for each single project, both an easy-to-interpret examination of the current status is given, and an indication for possible improvements in form of the reference set of similar but efficient projects is offered. Secondly, and more important, the results allow for additional analyses beyond single projects, but spanning groups of projects. Using measures of centrality like the mean or median efficiency score, groups of projects can be compared. The groups to be considered can be based on different characteristics available, including software process, application domains, scale or even license. Using any of the criteria, the population can be divided in two or more groups, and the distribution of efficiency scores can be compared statistically, using the hypothesis that the dividing variable has an effect on efficiency. If any statistically significant difference in efficiency shows up, the hypothesis is validated, meaning that indeed projects showing a certain

characteristic tend to be more (or less) efficient.

In this paper, first an example is detailed based on comparing different software process designs, using the inequality of contributions within projects as dividing factor. We will therefore compute a correlation coefficient between the inequality measure as given above, and the efficiency score of projects. In this case, the underlying hypothesis, that a relationship between group working style as depicted by the resulting inequality and efficiency exists, can not be confirmed: There is no significant correlation between both measures, which shows that a higher inequality does not have any effect on efficiency.

As a second example, the hypothesis that a very stringent copyleft-licensing scheme increases the productivity of the participants is explored. We therefore divide the set of projects into two groups, one consisting of projects licensed under GNU GPL only, resulting in 18 projects out of 43, the other with all remaining projects. Then, a non-parametric Mann-Whitney U-test is employed to test the distribution of the efficiency scores between the groups. In this case, there is no significant difference, so the hypothesis that a copyleft-licensing scheme has an impact on efficiency has to be discarded.

As a last example, we base the grouping of projects on the intended audience, with one group targeting developers and system administrators only, resulting in 20 projects, and others, to explore whether this characteristic has any impact on efficiency. Also in this case, there is no significant difference in efficiency between the two groups, again showing that the intended audience does not lead to a change in efficiency.

FUTURE RESEARCH

As this has been a very first attempt of applying DEA to the context of F/OS projects, numerous directions for future research exist. The first area of research identified with the data set in this paper is the definition of relevant input and output factors. We propose to add at least two additional factors, the first one being contributors other than programmers as an input factor, the second one being a mix of different product metrics to capture aspects of the software system produced. This could include concepts like McCabe's cyclomatic complexity (McCabe, 1976) or object-oriented metrics, e.g. the Chidamber-Kemerer suite (Chidamber and Kemerer, 1994). Also other outputs could be measured and included, for example postings to mailing lists, website metrics, and even popularity measures (Weiss, 2005).

In this paper, we have compared a set of relatively large F/OS projects. It would also be interesting to include smaller projects, in which case the issue of variable or constant returns to scale would increase in importance. A comparison might also be useful to be done within a certain application area only, for example frameworks for web service implementation, to eliminate additional influences.

Of maybe even more interest than those considerations given above on setting up a DEA might be analyses based on the respective results. After having computed efficiency scores for a set of projects, other characteristics of these can be used in order to explore reasons for any differences in efficiency. Possible characteristics would include mostly aspects of the software development process employed, similar to this paper, which used the inequality within the development team as explanatory variable. Respective comparisons could also be extended to include projects following very different approaches, like commercial projects employing for example agile methodologies, or projects with mixed participation from volunteers and paid contributors. To allow for these comparisons, additional characteristics of the projects considered need to be collected.

CONCLUSION

In this paper, we have proposed for the first time a method to compare the efficiency of F/OS projects. The method used is the DEA, which is well-established in other fields and offers several advantages in this context as well: It is a non-parametric optimization method without any need for the user to define any relations between different factors or a production function, can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Using a data set of 43 large F/OS project retrieved from SourceForge.net, we have demonstrated the application of DEA. Results show that DEA indeed is usable for comparing the efficiency of projects in transforming inputs into outputs, that differences in efficiency exist, and that single projects can be judged and ranked accordingly. We have also detailed additional analyses based on these results, which stated that neither the inequality in work distribution within the projects, nor the licensing scheme, nor the intended audience has an effect on the efficiency of a project. As this is a first attempt at using this method for F/OS projects, several future research directions are possible. These include additional work on determining input and output factors, and additional analyses based on the results using other project characteristics. These could include comparisons within application areas, different project scales, and also comparisons to commercial or mixed-mode development projects.

REFERENCES

Albrecht, A.J., & Gaffney, J.E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6), 639-648.

Atkins, S., Ball, T., Graves, T., & Mockus, A. (1999). Using Version Control Data to Evaluate the Impact of Software Tools. In *Proceedings of the 21st International Conference on Software Engineering* (pp. 324-333), Los Angeles, CA.

Banker, R.D., Charnes, A., & Cooper, W. (1984). Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis. *Management Science*, 30, 1078-1092.

Banker, R.D., & Kemerer, C. (1989). Scale Economies in New Software Development. *IEEE Transactions on Software Engineering*, 15(10), 416-429.

Banker, R.D., & Slaughter, S.A. (1997). A Field Study of Scale Economies in Software Maintenance. *Management Science*, 43(12), 1709-1725.

Charnes, A., Cooper, W., & Rhodes, E. (1978a). *A Data Envelopment Analysis Approach to Evaluation of the Program Follow Through Experiments in U.S. Public School Education* (Management Science Research Report No. 432). Pittsburgh, PA: Carnegie-Mellon University.

Charnes, A., Cooper, W., & Rhodes, E. (1978b). Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 2, 429-444.

Chidamber, S.R., & Kemerer, C.F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6), 476-493.

Cook, J.E., Votta, L.G., & Wolf, A.L. (1998). Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering*, 24(8), 650-663.

Cooper, W., Seiford, L., & Tone, K. (2000). *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*, Boston, MA: Kluwer Academic Publishers.

Crowston, K., Annabi, H., & Howison, J. (2003). Defining Open Source Software Project Success. In *Proceedings of ICIS 2003*, Seattle, WA, 14-17 December

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Towards A Portfolio of FLOSS Project Success Measures. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland, May 25.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings - Software Engineering*, 149(1), 3-17.

Dempsey, B.J., Weiss, D., Jones, P., & Greenberg, J. (2002). Who is an open source software developer? *Communications of the ACM*, 45(2), 67-72.

Dixon, R. (2003). *Open Source Software Law.* Norwood, Massachusetts: Artech House.

Farell, M.J. (1957). The Measurement of Productive Efficiency. *Journal of the Royal Statistical Society*, Series A 120(3), 250-290.

Ghosh, R., & Prakash, V.V. (2000). The Orbiten Free Software Survey. *First Monday*, 5(7).

Hahsler, M., & Koch, S. (2005). Discussion of a Large-Scale Open Source Data Collection Methodology. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, Hawaii.

Hertel, G., Niedner, S., & Hermann, S. (2003). Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7), 1159-1177.

Holck, J., & Jorgensen, N. (2004). Do Not Check in on Red: Control Meets Anarchy in Two Open Source Projects. In Koch, S. (ed.), *Free/Open Source Software Development*, Hershey, Pennsylvania: Idea Group Publishing.

Hu, Q. (1997). Evaluating Alternative Software Production Functions. *IEEE Transactions on Software Engineering*, 23(6), 379-387.

Hunt, F., & Johnson, P. (2002). On the pareto distribution of Sourceforge projects. In *Proceedings of the Open Source Software Development Workshop* (pp. 122-129), Newcastle, UK.

Kitchenham, B. (2002). The question of scale economies in software - why cannot researchers agree? *Information & Software Technology*, 44(1), 13-24.

Kitchenham, B., & Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. *IEEE Transactions on Software Engineering*, 30(12), 1023-1035.

Koch, S. (2004). Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2), 77-88.

Koch, S. (2005). *Effort Modeling and Programmer Participation in Open Source Software Projects* (Arbeitspapiere zum Tätigkeitsfeld Informationsverarbeitung, Informationswirtschaft und Prozessmanagement, Nr. 03/2005). Department für Informationsverarbeitung und Prozessmanagement, Wirtschaftsuniversität Wien, Vienna, Austria.

Koch, S., & Schneider, G. (2002). Effort, Cooperation and Coordination in an Open Source Software Project: GNOME. *Information Systems Journal*, 12(1), 27-42.

Krishnamurthy, S. (2002). Cave or community? an empirical investigation of 100 mature open source projects. *First Monday*, 7(6).

Laurent, L.S. (2004). *Understanding Open Source and Free Software Licensing.* Cambridge, Massachusetts: O'Reilly & Associates.

Mayrhauser, A., Wohlin, C., & Ohlsson, M. (2000). Assessing and Understanding Efficiency and Success of Software Production. *Empirical Software Engineering*, 5(2), 125-154.

McCabe, T.J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4), 308-320.

Mockus, A., Fielding, R., & Herbsleb, J. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), 309-346.

Myrtveit, I., & Stensrud, E. (1999). Benchmarking COTS Projects Using Data Envelopment Analysis. In *Proceedings of 6th International Software-Metrics-Symposium* (pp. 269-278), Boca-Raton.

Park, R.E. (1992). *Software size measurement: A framework for counting source statements* (Technical Report CMU/SEI-92-TR-20). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Perens, B. (1999). The Open Source Definition. In DiBona, C. et al. (eds.), *Open Sources: Voices from the Open Source Revolution*, Cambridge, Massachusetts: O'Reilly & Associates.

Raymond, E.S. (1999). *The Cathedral and the Bazaar*. Cambridge, Massachusetts: O'Reilly & Associates.

Robles, G., Koch, S., & Gonzalez-Barahona, J.M. (2004). Remote analysis and measurement of libre software systems by means of the CVSanalY tool. In *ICSE 2004 - Proceedings of the Second International Workshop on Remote Analysis and Measurement of Software Systems* (pp. 51-55), Edinburgh, Scotland.

Rosen, L. (2004). *Open Source Licensing: Software Freedom and Intellectual Property Law*. Englewood Cliffs, New Jersey: Prentice Hall PTR.

Stallman, R.M. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallma*n. Boston, Massachusetts: GNU Press.

Stewart, K.J. (2004). OSS Project Success: From Internal Dynamics to External Impact. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland, May 25.

Stewart, K.J., & Ammeter, T.A. (2003). An Exploratory Study of Factors Affecting the Popularity and Vitality of Open Source Projects. In *Proceedings of the 23rd International Conference on Information Systems*, Barcelona, Spain.

Varian, H.R. (2005). *Intermediate Microeconomics: A Modern Approach***.** Seventh Edition, New York, N.Y.: W. W. Norton & Company.

Weiss, D. (2005). Measuring Success of Open Source Projects Using Web Search Engines. In *Proceeding of the 1st International Conference on Open Source Systems* (pp. 93-99), Genoa, Italy.

This work was previously published in Emerging Free and Open Source Software Practices, edited by S. Sowe, I. Stamelos, and I. Samoladas, pp. 25-46, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.21
Examining Open Source Software Licenses through the Creative Commons Licensing Model

Kwei-Jay Lin
University of California, USA

Yi-Hsuan Lin
Creative Commons Taiwan Project, Taiwan

Tung-Mei Ko
OSSF Project, Taiwan

ABSTRACT

In this chapter, the authors present a novel perspective by using the Creative Commons (CC) licensing model to compare 10 commonly used OSS licenses. The authors also propose a license compatibility table to show that whether it is possible to combine OSS with CC-licensed open content in a creative work. By using the CC licensing concept to interpret OSS licenses, the authors hope that users can get a deeper understanding on the ideas and issues behind many of the OSS licenses. In addition, the authors hope that by means of this table, users can make a better decision on the license selection while combining open source with CC-licensed works.

INTRODUCTION

With the rapid growth of the open source software (OSS) community in the past decade, many users now are convinced that OSS is a practical and attractive alternative to proprietary software. Since almost all OSS licenses allow worldwide, royalty-free usage and encourage users to copy, modify,

and enhance original codes, OSS has attracted many users and programmers. Some other benefits include significantly lower development and deployment cost, and software quality improvement due to open inspections and discussions.

To meet the needs of various authors and users, different software licenses have been defined. The diversity and complexity of these licenses, on the other hand, create confusions for many potential OSS authors and users. It has been a constant community effort through articles, reviews, and books to discuss and to elaborate on the subtle differences among these licenses.

For non-software publications, such as Web sites, graphics, music, film, photography, literature, courseware, and so on, that normally fall under the current copyright law, some authors may want to open up part of their rights to the public with a spirit similar to those of OSS licenses. To allow for such possibilities, Creative Commons (CC) was founded in 2001 to define the licenses beyond the traditional "all rights reserved" copyright definition. CC licenses, motivated in part by the GNU General Public License (GPL) of the Free Software Foundation (FSF), provide a similar function to OSS licenses for non-software creative works.

Both OSS and CC licensing models are about promoting the ideas of free access. Therefore, it is not a rare case to combine open software released under OSS licenses with CC-licensed creative material. Nevertheless, there are differences between these two models. For users who combine these two types of materials to create a new resulting work, some questions are of deep concern. For example, whether a specific OSS license is compatible with CC licenses? Which license should the resulting work apply to? Unfortunately, so far there is hardly any study discussing these issues in depth.

As participants of the open source movement in Taiwan, we have witnessed the flourishing innovation and creativity of OSS activities in Taiwan. However, the license selection issue has continued to be an obstacle for many potential local contributors. Part of the charters of the Open Foundry project in Taiwan (called OSSF, http://www.openfoundry.org) is to help people easily capture a basic understanding of the licenses that govern OSS, related documentations and open content.

In this chapter, we present a novel perspective by using the CC licensing model to compare 10 commonly-used OSS licenses. Specifically, we have defined a license compatibility table that shows whether it is possible to combine OSS with CC-licensed open content in a creative work. The idea of comparing the two types of licenses is partly inspired by Rosen (2004). In Chapter 10 (pp. 244-251) of his book, Rosen takes four commonly used OSS licenses as examples and discusses the compatibility of these licenses. Similarly, our study may help people understand if they can re-license a resulting work under a specific CC license. The reason for our study on the compatibility table is from the observation that many new OSS contributors are primarily interested in getting their software known and accepted by the community, and circulated as widely as possible. They do not want to interfere with licensees' use of the software nor constraining the licensing of derivative works. Their goal is to create works that people may share and enjoy, much like open content. Therefore, by using the CC licensing concept (such as *attribution* and *share alike*) to interpret OSS licenses, people may get a deeper understanding on the ideas and issues behind many of the OSS licenses, and make a better decision on the license selection.

The rest of the chapter is organized as follows. The following section reviews the basic elements of OSS licenses and CC licenses. Subsequently, the comparison of the two licenses classes is presented. Next, we discuss two new license concepts, then the chapter is concluded in the last section.

BACKGROUND: FROM GPL TO ATTRIBUTION

OSS Licenses and FDL

There are many types of OSS licenses. According to the statistics from FSF and the Open Source Initiative (OSI), there are over 60 OSS licenses. In general, these licenses have three common characteristics (Free Software Foundation, 2005a; The Open Source Initiative, 2006a):

1. No royalties
2. No geographical restrictions on distribution
3. No specific licensees

Among them, we have chosen 10 more commonly-used OSS licenses (including GNU General Public License, GNU Library/Lesser General Public License, BSD license, MIT license, Apache Software License 1.1, zlib/libpng License, Artistic License, Common Public License, Qt Public License, and Mozilla Public License) plus the GNU Free Documentation License (FDL) for discussion in this paper. These licenses, excluding the FDL, have all been approved by the OSI and conform to the Open Source Definition (OSD) (The Open Source Initiative, 2006b).

The most well-known OSS license is GPL, which was drafted by Richard M. Stallman, the founder of the FSF and the Project GNU. The GPL is developed on the basis of the *copyleft* mechanism. According to the *copyleft* mechanism, a licensee has to adopt the same license as that of the licensor for his (or her) program. Using the *copyleft* mechanism, source code can always remain open and royalty free. The GNU Library/Lesser General Public License 2.1 (LGPL) is the other OSS license implementing the *copyleft* mechanism (Free Software Foundation, 2005b). The LGPL is designed specifically for library code, and is less strict than the GPL.

On the other hand, the *copyleft* mechanism does not limit any right arising from *fair use*. Thus, when an author uses GPL-licensed or LGPL-licensed codes as examples in a book or as references, he (or she) may not have to apply the GPL or LGPL to the book as long as the application falls within the scope of *fair use*. Under this circumstance, the author can choose a license at his (or her) will, for example, any traditional proprietary copyright license or any CC license for the book. Similarly, in accordance with the *fair use* doctrine, when the author attaches a whole copy of the GPL-licensed or LGPL-licensed codes with the book while published or distributed, the license adoption of the book will not be restricted to the GPL or LGPL.

Compared with the GPL, the BSD license, another popular OSS license, does not impose any restriction on the licensee in terms of future license selection. In other words, the licensee is allowed to use any license (even make it proprietary) for his (or her) program and is also allowed to collect royalties. The Apache Software License 1.1 (Apache 1.1), zlib/libpng License (zlib/libpng), and MIT License have similar characteristics as that of the BSD license.

The other four licenses discussed in this chapter are the Mozill Public License 1.1 (MPL), Common Public License 1.0 (CPL), Qt Public License 1.0 (QPL), and the Artistic License (Artistic). Basically, the MPL, CPL, and QPL are all designed for the commercial use of OSS, thus their regulations about licensees' rights and obligations are very similar. The MPL employs a partial *copyleft* mechanism in that the licensee can only use the MPL for his (or her) program in principle (Mozilla.org, 2006). However, the licensee is allowed to adopt another license for certain parts of the program. The CPL adopts the *copyleft* mechanism and is the first license to regulate commercial distribution of OSS with separate terms. Artistic has its own legal logic, which is different from the other nine OSS licenses.

Same as the GPL, the FDL is drafted by Stallman and also adopts the *copyleft* mechanism. However, the FDL is normally used for textbooks or teaching materials written for some equipment or software (Free Software Foundation, 2005b). Wikipedia, a famous online encyclopedia, adopted the FDL for its text content, is a noted example (Wikipedia, 2006).

The Inception of Creative Commons

A group of professionals from various fields, including intellectual property and cyberlaw experts James Boyle, Michael Carroll, and Lawrence Lessig, and MIT science professor Hal Abelson, founded Creative Commons in 2001 (Lessig, 2005). CC advocates the "some rights reserved" concept in contrast to the default "all rights reserved" in current copyright laws. CC also takes ideas in part from the FSF and produces a series of copyright licenses to help creators declare to the world what freedom they want their works to carry. These freedoms are composed by four elements: *Attribution* (denoted as "by" or ), *noncommercial* ("nc" or ), *no derivatives* ("nd" or ) and *share alike* ("sa" or ). When CC licenses v. 1 were first released in December 2002, these four elements were all optional. Later on, CC found that 98% of the adopters have chosen "attribution" as a requisite; thus CC sets "attribution" as a default in v. 2, and offers six licenses (Lessig, 2005).

CC Licenses

CC licenses are designed to bridge creators and users in that users have no need to ask for creators' prior permission to use the works as long as they follow the rules the creators set. For example, if a work is released under the "by-nc" CC license (i.e., attribution and noncommercial), a user can freely make use of the work under the condition that the user uses this work for noncommercial purposes only and must always credit the original creator. The six CC licenses are defined as follows:

1. **Attribution:** It means that a user can freely use the work, provided that he (or she) credits the creator.
2. **Attribution-share alike:** It means that a user can freely use the work, provided that he (or she) credits the creator and also licenses any derivative under the same license as that of the original work.
3. **Attribution-no derivatives:** It means that a user can only make use of verbatim copies of the work and have to credit the creator.
4. **Attribution-noncommercial:** It means that a user can only use the work for noncommercial purpose, and have to credit the creator.
5. **Attribution-noncommercial-no derivatives:** It means that a user can only make use of verbatim copies of the work, for noncommercial purposes only, and have to credit the creator.
6. **Attribution-noncommercial-sharealike:** It means that a user can only use the work for noncommercial purposes, credit the creator, and license any derivative under the same license as that of the original work.

In addition to the above six licenses, CC also offers other licenses for more specialized situations. For example, sampling licenses allow people to use a part of some creative works and mix with some original or other parts to create a new work. One can use founders copyright to free works from copyright completely, after it has been created for 14 or 28 years. In general, CC provides the vehicle that "does not mean giving up your copyright. It means offering some of your rights to any member of the public but only on certain conditions" (Creative Commons, n.d.).

Reviews of Issues on OSS and CC Licenses

Since the OSS licensing model appeared in the 1990s, it has started a lot of discussion, for example: What is the free/open source software movement? How does it run? How does it work with the current legal system? (Hill, 1999). Many have questioned about the enforceability of OSS licenses (Nadan, 2002; Ravicher, 2000). Later, because of the lack of precedents regarding OSS licenses' enforceability, Gomulkiewicz (2002) proposes to create an open source license organization (OSLO) to solve issues relating to OSS licenses. Gomulkiewicz thinks that the OSLO could play a role in calling programmers and lawyers together to built up useful licensing practices of OSS and further solve related licensing problems.

Among these various OSS licenses, the GPL receives the most attention. Stoltz (2005) discusses the scope of derivative works under current US copyright laws and how the extent of derivative works affects the GPL. Besides, along with the rising number of successful open source commercial cases, the issues about OSS license policies started to get much attention. For example, Satchwell (2005) provides users a basic understanding of OSS; how to choose a suitable OSS license and how to establish an appropriate OSS policy.

With the increase of OSS licenses, there are more and more articles discussing issues of OSS licenses. However, among these articles, only a few have addressed the OSS license compatibility. Perens (1999) and Maher (2000) point out that OSS license compatibility is a noteworthy issue, but neither offers any concrete solution to the compatibility problem. Rosen (2004) provides some discussion on the issue of OSS license compatibility but does not come up with any concrete solution. Therefore, sensing the need of a simple and clear explanation for various OSS licenses, we use a relatively intuitive licensing model, CC licenses, to examine OSS licenses.

Since CC licenses' release in December 2002, these licenses spread quickly and dramatically. There have been more than 50 million Web pages linked to CC licenses as of August 2005 (Katz, 2006). However, with the rising popularity of CC licenses, many skeptical views have appeared. Some challenge the compatibilities between certain CC licenses or between different free-content contracts (e.g., Elkin-Koren, 2005; Katz, 2006); while others question that a variety of CC licenses will cause confusion or increase the information cost (e.g., Elkin-Koren, 2005; Katz, 2006). Moreover, license translation and legal adaptation may undermine the success of CC licensing (Valimaki, 2005).

Katz (2006) questions that a variety of CC licenses would puzzle users in that users may run into difficulties in determining which CC license is the most suitable for them. Elkin-Koren (2005) challenges the consistency of CC's strategy over license selection. He argues that CC attempts to reduce external information cost by its license choosing platform, but the variety of CC licenses would on the contrary impose extra informational burden on authors. He uses musical works as an example: In addition to six CC core licenses composed of four elements (i.e., attribution, noncommercial, no derivatives, and share alike), there are the other three sampling licenses (i.e., sampling, sampling plus, and noncommercial sampling plus). To find out which license is the most suitable one for musical works would unwittingly increase information costs. Elkin-Koren (2005) further states that the lack of standardization in CC licenses would increase the cost of ascertaining the rights and obligation related to any specific work.

In addition, Katz (2006) argues about the incompatibilities of CC licenses. He concludes that the viral effect of the "share alike" element will result in the incompatibilities problem between different share alike licenses and would further restrict the distribution of derivative works.

Some of the above mentioned literatures discuss the compatibility among different CC licenses, but none offers a systematic analysis on the problem of CC license incompatibility or combining OSS with CC-licensed content. Therefore, we attempt to illustrate what license a user may choose when he (or she) combines OSS with CC-licensed content by a clearly defined table to be discussed next.

MAIN FOCUS OF THE CHAPTER

Examining Compatibility between OSS and CC Licenses

Our study in this chapter is to define a license compatibility table that shows whether the combination of OSS and CC-licensed open content in a creative work may be properly licensed. The table (Table 1) may help people understand if they can continue to re-license a derived work under a specific CC license.

In Table 1, we use "by," "nd," "nc," and "sa" to denote CC's four elements "attribution," "no derivatives," "noncommercial," and "share alike" respectively. The mark "○" indicates that when a derivative work incorporates two or more works under licenses listed in a specific column and a specific row, it can be re-licensed under the CC license shown in the column. The mark "X," on the other hand, shows that a derivative work, incorporating two or more works under licenses listed in a specific column and a specific row, cannot be re-licensed under the license shown in the column. For example, if one combines a program A released under the GPL, with an open content B issued under the CC attribution license, and produces a new work C. One may not re-license C under the CC attribution license because the GPL requires that GPL-applied program or its derivative work must always be governed by the GPL.[1] Thus, A and C must be licensed similarly, and C work will not be able to release under any CC license.

In the following sections, we discuss the table entries in details.

No Derivatives

The 10 OSS licenses chosen to compare with CC licenses in this chapter are all approved by the

Table 1. License compatibility table

	by	by-nd	by-nd-nc	by-nc	by-nc-sa	by-sa	FDL
GPL	X	X	X	X	X	X	X
LGPL	X	X	X	X	X	X	X
MPL	X	X	X	X	X	X	X
QPL	O	X	X	X	X	O	O
CPL	O	X	X	X	X	O	O
Artistic	O	X	X	X	X	O	O
Apache	O	X	X	X	X	O	O
Zlib/libpng	O	X	X	X	X	O	O
BSD	O	X	X	X	X	O	O
MIT	O	X	X	X	X	O	O

OSI. An OSI-certified license must conform to the OSD (The Open Source Initiative, 2006b). Under criterion 3 of the OSD, the license must permit making modifications and derived works (The Open Source Initiative, 2006c). Therefore, these 10 OSS licenses allow modification to the original program. Thus, any of the six CC licenses which contains "No Derivative" element (i.e., CC attribution-no derivatives license, CC attribution-no derivatives-noncommercial license) is not compatible with any of the 10 OSS licenses, and "X" is shown in all cells of the "by-nd" and "by-nd-nc" columns in the table.

Noncommercial

Criterion 1 of the OSD states that an OSS license should not "restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources" (The Open Source Initiative, 2006c). It thus implies that any OSD-compliant license should not restrict any use of commercial purposes. This results in the conflict between 10 OSS licenses and any CC license with "noncommercial" element (i.e., CC attribution-noncommercial licenses, CC attribution-noncommercial-share alike licenses). An "X" is shown in all cells of "by-nc" and "by-nc-sa" in the table.

Copyleft

The *copyleft* mechanism provides that anyone will be granted the rights to use, copy, modify, or distribute a program or its derivative works on the condition that when redistributing a program, with or without change, all rights he (or she) gained must be passed on to subsequent users (Free Software Foundation, 2005b). The GPL, LGPL, FDL, and CPL are terms to implement the *copyleft* mechanism. The implemented result will be the original work and its derivative works must be redistributed under the same license as the original work.

Although not originated from the FSF, the MPL partially employs a *copyleft* mechanism. MPL requires that modifications to MPL-licensed program must be governed by MPL (Mozilla.org, 2006). Because of this viral nature of *copyleft*, the GPL, LGP, MPL, and CPL is not compatible with any of the CC licenses, and thus "X" is shown in all cells of the top four rows.

Author Credit

The Apache, zlib/libpng, BSD, and MIT explicitly indicate that the authors of original work must be credited (Lin, Ko, Chuang, & Lin, 2006). QPL, CPL, and Artistic have copyright notices related regulations, and do not exclude the authors' names of the original work from the copyright notices.Yet, CPL implements copyleft mechanism. Therefore, GPL, LGPL, MPL, in addition to CPL are not compatible with CC attribution licenses. A work incorporating a program licensed under the QPL, Artistic, Apache, zlib/libpng, BSD, or MIT with other works issued under CC attribution license could be re-licensed under CC attribution license. Excluding the GPL, LGPL, CPL, and MPL, "○" is shown in the other cells of the "by" column in the table.

Share Alike

The compatibility between the 10 OSS licenses and the CC attribution-share alike license is discussed next. The GPL, LGPL, CPL, and MPL implement the *copyleft* mechanism. But the other six OSS licenses do not explicitly adopt it and do not have the viral effect on the resulting derivative work. Thus, when a work incorporates a program licensed under the QPL, Artistic, Apache, zlib/libpng, BSD, or MIT with the other work issued under the CC attribution-share alike license, this newly created work may be re-licensed under the

CC attribution-share alike license. Except the top four cells, the mark "○" is shown in the other cells of the "by-sa" column in the table.

FDL

Finally, we examine the compatibility between the 10 OSS licenses and the FDL.

Even though the GPL, LGPL and FDL are all developed by the FSF, because of *copyleft* mechanism's viral effect, when a work incorporates a GPL-licensed or LGPL-licensed program with other FDL-released work, the resulting work may not be re-licensed under the FDL. The same result applies to CPL because CPL implements copyleft as well. The MPL partially employs the *copyleft* mechanism; thus, a created derivative work incorporating a MPL-licensed program with the other FDL-released work may not be re-licensed under the FDL, either.

In principle, the FDL enables the same freedom as the CC attribution-share alike license (Creative Commons, 2005). Due to the *copyleft* mechanism, when a work incorporates a program licensed under the GPL, LGPL, CPL, or MPL with the other FDL-released work, this resulting derivative work may not be re-licensed under the FDL. Except the GPL, LGPL, CPL, and MPL, a work incorporating a program governed by the QPL, CPL, Artistic, Apache, zlib/libpng, BSD, or MIT with the other work issued under the FDL, this new created work could be re-licensed under the FDL. Except the top four cells, "○" is shown in the other cells of the "FDL" column in the table.

Using License Compatibility Table for License Selection

From Table 1, we could identify the following license selection strategies. If a user would like a creative work which combines OSS with CC-licensed open content to be re-licensed under the CC license or the FDL, he (or she) should avoid using OSS licensed under GPL, LGPL, or MPL. In other words, if a creative work is combining OSS licensed under the GPL, LGPL, CPL, or MPL with CC-licensed open content, this work is not possible to be re-licensed under any CC license.

If a user would like a creative work which combines OSS with CC-licensed open content to be re-licensed under some CC licenses, he (or she) should choose OSS licensed under the QPL, Artistic, Apache, zlib/libpng, BSD, or MIT. However, not all CC licenses are compatible with the QPL, Artistic, Apache, zlib/libpng, BSD, or MIT; only CC by and by-sa licenses are compatible with these six licenses. In other words, a creative work combing an OSS license under QPL, Artistic, Apache, zlib/libpng, BSD, or MIT with CC by or by-sa licensed open content, the resulting work could be re-licensed under CC license identical to the original open content. Similar to CC by or by-sa license, FDL is not compatible with the GPL, LGPL, CPL, or MPL, but is compatible with QPL, Artistic, Apache, zlib/libpng, BSD, or MIT. Therefore, if a user would like a creative work which combines OSS with FDL-licensed open content to be re-licensed under the FDL, he (or she) may use OSS licensed under any of these six OSS licenses.

From the above discussions, we can make two simple conclusions. Firstly, OSD-compliant OSS licenses should not restrict any use of commercial purposes[2], and OSD-compliant OSS licenses must allow modifications and derived works.[3] Therefore, CC licenses containing "noncommercial" or "no derivatives" element are not compatible with 10 OSS licenses discussed in this chapter.

The second conclusion is that the *copyleft*'s viral effect requires the original work and its derivative works to be redistributed under the same license as the original work. Thus, the GPL, LGPL, CPL, and FDL, which adopt the *copyleft* mechanism completely, plus MPL, which partially adopts the *copyleft* mechanism, are not compatible with any CC license.

For authors that are not combining OSS with open content, the above discussion may provide some useful insights as well. The *copyleft* mechanism is a strong license requirement that may prevent others from producing derivative works mixed with even the "share alike" element. It is probably better to select other licenses if such a requirement may present a problem in the future.

FUTURE TRENDS

Open Access Publishing

CC licenses are inspired from the GPL. In addition to OSS and CC licensing models, other models have been developed in different fields sharing similar notions with that of OSS and CC. Open access publishing is one of them.

Typically, publishers charge readers a subscription fee, and sometimes also charge authors a page fee. Open access publishing, on the contrary, allows the author to retain his (or her) article's copyright; at the same time, authors or their sponsors, not the users, pay the publishers.

Although the open access publishing model will make authors bear more cost than traditional publishing models, the charged fees are possible to be transferred to the authors' sponsor institutes or even be waived (Suber, 2004). Moreover, open access publishing will increase the possibility that the authors' articles are searched, and help to build the authors' prestige (Harmel, 2005). Recent studies also show the same result: online articles are more frequently cited (Lawrence, 2001) and more often used than offline articles (Lawrence, 2001; Walker, 2004). Therefore, more and more leading publishers, such as the Public Library of Science (PLoS) and BioMed Central have joined the open access movement.

Science Commons, a newly launched project of Creative Commons, was founded to support the sharing of scientific research, such as the field of biotechnology, medicine, and even law[4], with the same "some rights reserved" spirit Creative Commons holds.

Studies have found that the open access publishing model is practicable (Gonzalez, 2005; Odlyzko, 1998), and there are already publishers gain profits from it (Walker, 2004). It is foreseeable that with the increasing subscription fees of academic journals the open access publishing model will continue to gain more support, especially on academic content.

New OSS Elements

Compared with numerous OSS licenses, the CC licensing model built on the basis of four elements is relatively simple. However, although these four elements are less complex and easier to understand, they are not broad enough to cover all major considerations of OSS licenses. Here we include two new concepts, "no endorsement" and "modification record" that should be considered by OSS users when selecting a license.

No Endorsement

One of the OSS's common characteristics is that anyone is free to create derivative works (The Open Source Initiative, 2006c). Because of this, the quality of derivatives is hard to control. When the quality of a derivative is not as good as the original program, but the name of the original developer or the copyright holder is still shown on the derivative, new users may not have enough acknowledgement of this and relies on the name of the original developer or the copyright holder to evaluate the derivatives. Under the circumstance, it may harm the reputation of the original developer or the copyright holder. Sometimes, the developer of the derivatives may intentionally show the name of the original developer or the copyright holder on the derivative work to endorse or promote his (or her) own works.

Therefore, to prevent OSS adopters from using the authorship to implicitly or explicitly show the

support, association of the initial developers or to promote their derivative work, and, even more, to prevent the derivatives from being wrongly trusted, the original program's developer should choose a license which contains "no endorsement" or disclaimer clause[5], such as BSD and Artistic. The Creative Archive License developed by BBC adopts the main ideas of CC licenses but injects such a new element into the license.[6]

Modification Record

We also notice that many OSS licenses have regulations regarding the modification records.[7] Take the 10 OSS licenses we analyze in this chapter for example. Only the QPL does not require that a modification record must be made. Instead, the QPL forbids users to directly make modifications to the original works and requires that all modifications be in a form that is separate from the original works (e.g., patches).[8] All of the other nine licenses have modification record related regulation. These records are very helpful for the convenience of follow-up software modifications. Moreover, they are beneficial to maintain the original works' integrity.[9]

CONCLUSION

OSS licenses have triggered a lot of discussions in the past few years because of their complicacy. The OSI even appeals to reduce the number of approved OSI-licenses to allow programmers and users to understand OSS licenses more easily. In contrast with OSS licenses, CC licenses provide a cleaner licensing model. In this chapter we investigate the compatibility between the six CC licenses and 10 commonly-used OSS licenses including the FDL. OSS authors may use the table to identify which CC license he (or she) can use for his (or her) work that combines OSS with CC-licensed work.

However, CC's four simple elements do not capture all major issues of OSS. We thus raise two new issues, "no endorsement" and "modification record", to address some main concerns by OSS. We believe that by employing CC's four elements, plus our proposed two new elements, OSS community, including both authors and users, will be able to get a more complete picture of OSS licenses.

REFERENCES

Creative Commons. (n.d.). *Choosing a license.* Retrieved March 30, 2006, from http://creativecommons.org/about/licenses/

Creative Commons. (2005). *Discussion draft: Proposed license amendment to avoid content ghettos in the commons.* Retrieved March 30, 2006, from http://creativecommons.org/Weblog/entry/5701

Elkin-Koren, N. (2005). What contracts cannot do: The limits of private ordering in facilitating a creative commons. *Fordham Law Review, 74,* 375-422.

Free Software Foundation. (2005a). *FSF: Licenses.* Retrieved March 30, 2006, from http://www.fsf.org/licensing/licenses/index_html

Free Software Foundation. (2005b). *What is copyleft? GNU Project: Free Software Foundation (FSF).* Retrieved March 30, 2006, from http://www.gnu.org/copyleft/copyleft.html

Gomulkiewicz, R. W. (2002). De-bugging open source software licensing. *University of Pittsburg Law Review, 64,* 75-103.

Gonzalez, A. G. (2005). The digital divide: It's the content, stupid: Part 2. *Computer and Telecommunications Law Review, 11*(4), 113-118.

Harmel, L. A. (2005). The business and legal obstacles to the open access publishing movement for

science, technical, and medical journals. *Loyola Consumer Law Review, 17,* 555-570.

Hill, T. (1999). Fragmenting the copyleft movement: The public will not prevail. *Utah Law Review, 1999,* 797-822.

Katz, Z. (2006). Pitfalls of open licensing: An analysis of Creative Commons licensing. *IDEA: The Intellectual Property Law Review, 46,* 391-413.

Lawrence, S. (2001). *Free online availability substantially increases a paper's impact.* Retrieved July 10, 2006, from http://www.nature.com/nature/debates/e-access/Articles/lawrence.html

Lessig, L. (2005). *CC in review: Lawrence Lessig on supporting the commons.* Retrieved March 30, 2006, from http://creativecommons.org/Weblog/entry/5661

Lin, Y. H., Ko, T. M., Chuang, T. R., & Lin, K. J. (2006). Open source licenses and the Creative Commons framework: License selection and comparison. *Journal of Information Science and Engineering, 22,* 1-17.

Maher, M. (2000). Open source software: The success of an alternative intellectual property incentive paradigm. *Fordham Intellectual Property, Media and Entertainment Law Journal, 10,* 619-695.

Mozilla.org. (2006). *MPL FAQ.* Retrieved March 30, 2006, from http://www.mozilla.org/MPL/mpl-faq.html

Nadan, C. H. (2002). Open source licensing: Virus or virtue? *Texas Intellectual Property Law Journal, 10,* 349-377.

Odlyzko, A. (1998). The economics of electronic journals. In R. Ekman & R. Quandt (Eds.), *Technology and scholarly communication.* University of California Press.

Perens, B. (1999). The open source definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 171-188). Sebastopol, CA: O'Reilly & Associates.

Ravicher, D. B. (2000). Facilitating collaborative software development: The enforceability of mass-market public software licenses. *Virginia Journal of Law & Technology, 115,* 1522-1687.

Rosen, L. (2004). *Open source licensing: Software freedom and intellectual property law.* Upper Saddle River, NJ: Prentice Hall.

Satchwell, M. D. (2005). The tao of open source: Minimum action for maximum gain. *Berkeley Technology Law Journal, 20,* 1757-1798.

Stoltz, M. L. (2005). The penguin paradox: How the scope of derivative works in copyright affects the effectiveness of the GNU GPL. *Boston University Law Review, 85,* 1439-1477.

Suber, P. (2004) *Open access overview.* Retrieved July 10, 2006, from http://www.earlham.edu/~peters/fos/overview.htm

The Open Source Initiative. (2006a). *Open Source Initiative OSI-licensing.* Retrieved March 30, 2006, from http://www.opensource.org/licenses/

The Open Source Initiative. (2006b). *Open Source Initiative OSI—Certification Mark and Program.* Retrieved March 30, 2006, from http://www.opensource.org/docs/certification_mark.php

The Open Source Initiative. (2006c). *Open Source Initiative OSI—The Open Source Definition.* Retrieved March 30, 2006, from http://www.opensource.org/docs/definition.php

Valimaki, M. (2005). *The rise of open source licensing: A challenge to the use of intellectual property in the software industry.* Helsinki, Finland: Turre Publishing.

Walker, T. J. (2004). *Open access by the article: An idea whose time has come?* Retrieved July 10,

2006, from http://www.nature.com/nature/focus/accessdebate/13.html

Wikipedia. (2006). *Wikipedia: Copyrights—Wikipwdia, the free encyclopedia*. Retrieved March 30, 2006, from http://en.wikipedia.org/wiki/Wikipedia:Copyrights

KEY TERMS

Copyleft: Copyleft is a kind of licensing mechanism, with which licensees have to apply the same license the original works adopted to the derivative works.

Creative Commons Licenses: Creative Commons licenses are a kind of licensing model which applies to open content. Creative Commons licenses are composed by four elements (attribution, noncommercial, no derivatives, and share alike). Creative Commons licenses allow the licensees to make use of CC-licensed works with no need to get prior permission from the licensors as long as the licensees follow the conditions the licensors chose for the works.

License: It is a legal permission to commit some act.

License Compatibility: It is an abstract idea to illustrate whether two portions of content regulated by two different licenses can be combined within a work compatibly and produce the other resulting work.

Open Access Publishing: Open access publishing is a kind of publishing model, under which journals open access to the public immediate on publication and usually the authors of the journal articles do not need to pay the page fee for the publication.

Open Content: Open content describes the creative work which allows copying and modifying with no need to get extra permission from the licensors, such as works licensed under Creative Commons licenses.

Open Source Software Licenses: Open source software licenses apply to open source software. Open source software licenses feature that licensees can use, copy, distribute, and modify the regulated software on a royalty-free, worldwide basis.

ENDNOTES

1 Article 2(b) of the GPL stipulates that "You (licensee) must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program (GPL-applied program) or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License (GPL)."

2 OSD # 1 states that an OSS license should not "restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources."

3 OSD # 3 states that "The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software."

4 Open Access Law Project is established under Science Commons' publishing project to promote open access to legal scholarship. For more detailed information about Open Access Law Program, please see http://sciencecommons.org/literature/oalaw

5 Our "no endorsement" wordings are motivated by BBC's Creative Archive License. The detailed terms can be found on http://creativearchive.bbc.co.uk/licence/nc_sa_by_ne/uk/prov/.

6 BBC proposes five rules for Creative Archive Group License, which comprises "non-commercial," "share alike," "crediting" (attribu-

tion), "no endorsement and no derogatory use" and "UK." The first three rules are very similar to CC's; the last two are innovations created by BBC. For more details, please see http://creativearchive.bbc.co.uk/archives/2005/03/the_rules_in_br_1.html

[7] "Modification record" in this chapter includes several possible meanings, e.g., the record about who did the modification; the record about when the modification was made; the record about which part of the original programs has been changed.

[8] See article 2,3 of QPL.

[9] According to Andrew M. St. Laurent's opinion, QPL's requirement that a licensee distributes modifications separately with the initial work can protect the reputation of the initial developers and make clear the primacy of the initial developers' works. See Andrew M. St. Laurent, "Understanding Open Source & Free Software Licensing" (2004, p. 87, O'Reilly). In this chapter, we further extend St. Laurent's viewpoints and come up with the new element "modification record" for OSS's licenses.

This work was previously published in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, edited by K. St.Amant & B. Still, pp. 382-393, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 7.22
Integration of Libre Software Applications to Create a Collaborative Work Platform for Researchers at GET

Olivier Berger
GET/INT, France

Christian Bac
GET/INT, France

Benoît Hamet
GET/INT, France

ABSTRACT

Libre software provides powerful applications ready to be integrated for the build-up of platforms for internal use in organisations. We describe the architecture of the collaborative work platform which we have integrated and designed for researchers at GET. We present the elements we have learned during this project in particular, with respect to contribution to external libre projects, in order to better ensure the maintainability of the internal applications, and to phpGroupware as a framework for specific applications development.

INTRODUCTION

ProGET is a collaborative work platform, built out of a set of specialised libre software applications integrated together. ProGET is designed for the whole of teachers/researchers at GET. It provides every GET research project with the best features found in each application (wiki, mailing-lists management, shared WebDAV folders, Web portal, etc.).

We start with a description of the libre components that have been integrated and of the features that have been selected, as well as elements of architecture of the developed platform. We will

then introduce the strategy for collaboration that we have devised for our contribution to phpGroupware. We finish with a first evaluation at the end of the initial development phase.

RESEARCH AT GET

The *Groupe des Écoles des Télécommunications*[1] (GET) is composed of several engineering and business schools together with research centers in Paris (ENST), Brest (ENST Bretagne), and Évry (INT), in France. The research teams are made up of more than 600 full-time research equivalents. The range of the researchers' expertise is from technologies to social sciences, and enables an integrated approach of characteristic of GET research and fosters its adaptability to new application sectors and new usages in response to current challenges in the fields of information and communication.

To give a clearer view of research at GET, the Research Office started to catalogue the activities from the different locations so that the research may be described in terms of research *projects* and *programmes*. A project is made-up of a group of people working together on closely related subjects. For example the authors belong to the "Collaborative Platforms for Research" (PFTCR) project. A programme associates different projects loosely related. For example our project is related to the "Web and Information Society" programme. Due to the fact that GET teams are located in different areas, the research office also decided to propose a Web platform to help researchers collaborate through groupware tools and animate their research work.

PROGET INTEGRATED PLATFORM FOR COLLABORATIVE WORK

ProGet has been launched in July 2003 with the following goals:

1. Provide all research teams in GET and their external partners (more than 1,250 users) with cutting-edge technologies in terms of Web based groupware tools

Figure 1. Homepage of the GET public research portal

2. Allow the Research Office to manage the *administrative* records describing research projects
3. Generate a public *Web portal* based on information extracted from both previous components.[2] The home page of the Web portal is shown Figure 1.[3]

Groupware Tools for Researchers/ Teachers

The following features have been selected as corresponding to the basic needs for collaborative work in the context of research activities at GET:

- **Document sharing:** People in the same project must have a way to share the documents they produce, whatever the type of document.
- **Asynchronous communication:** Each project manages predefined *mailing lists* and may also create mailing lists as it needs.
- **Online editing:** People in the same project group are allowed to write easily and collaborate on simple hypertext pages using their Web browsers, to create a collaborative Web, in a wiki-type tool.
- **Publishing short announcements:** The project manager can write short news items about the project, and have the project news published online on the public Web portal very easily.

The platform must provide secure access, from any point on the Internet, to a set of tools. These tools will have to be accessible through a Web interface, to allow people to use them without the need to install any specific application on the computer from which they are connected.

Some features must also be accessible through non-Web client tools, in the Microsoft Windows™, or GNU/Linux environments. For instance, as explained in the section entitled, Project Documents Repository Accessible through WebDAV, access to the DAV repositories through DAV-compatible file managers. Information generated in this collaborative part of the platform may be made accessible to the public portal directly by the research teams, without further interaction with a Webmaster.

Tools for the Research Office

The Research Office is supervising the setup and evolution in time of the research projects and programs. In ProGET, it uses, for this purpose, a dedicated tool, which helps managing the life cycle of the *project description forms.*

This module does not provide a replacement for the research information system under development at the present time at GET. It only allows the management of information on the projects which are necessary for the other parts of the platform. It handles the initial list of persons participating in projects teams, people who are project managers, the projects' descriptions, and their yearly goals.

The project description forms are initially filled by the researchers who will be responsible for the projects. They are then validated by the research offices. This validation allows the creation of the collaboration projects in the platform. Those forms are also used to feed the contents of the research public Web portal.

Public Web Portal

The platform publishes a public Web portal which reflects the state of research activity at GET. This portal describes essential information relating to the research programmes and projects at GET. Figure 1 shows the home page of the portal.

This public Web site corresponds to a dynamic extraction of the contents of the project description forms stored in the database. That is why it will be updated immediately whenever the GET Research Office validates new projects creation, or updates the contents of existing forms.

Since the data extraction is dynamic, this portal Web site can be duplicated in several flavors according to the origin of the visit. So when the Web site is browsed from inside the GET intranet, it will provide more detailed information, including some facts about the research projects which are considered to be published to a restricted audience only (teams composition, detailed yearly goals of the projects, etc.).

PREVIOUS WORK WITH LIBRE SOFTWARE COLLABORATION TOOLS

As described by Cousin, Ouvradou, Pucci, and Tardieu (2001), we and other GET researchers and students contributed in the development of a collaborative platform called *PicoLibre*. This platform is targeted at collaborative software development, to help students in computer science curricula and researchers to develop and host their software projects. It provides the necessary collaboration tools (mailing lists, CVS repository, issue trackers, etc.) in a similar way to SourceForge. Another goal of the PicoLibre project was to foster the publication of projects as libre software, since it would be a way to introduce the users to the practice of the common tools used everyday by libre software developers.

This platform was created as a free software tool (published under the GNU GPL license[4]), using free software components and adapting them for PicoLibre. Several PicoLibre instances are in operation since Fall 2001 in GET sites and outside.

In some aspects, PicoLibre is a very successful project, since it allows GET to host a large number of software projects with a minimum burden for its administrators. It is also sometimes used for projects not specifically targeted at software development, for teams that need a collaboration space and associated *groupware* tools. The PicoLibre project has helped us attain a better knowledge of the specifics of the design and development, but also the administration, of a collaborative Web-based platform in the context of a higher education institution.

But PicoLibre also failed to certain extents, especially with respect to its maintainability, and in its capacity to be integrated into a libre software distribution. The project was done in such short time that developers only concentrated on the innovative parts and spent too few efforts in keeping the software they developed in the mainstream of the libre projects it was based on (in particular with respect to phpGroupware [phpgw]).

Although having been used intensively by teams of researchers or students investigating software development, PicoLibre was not well suited for generic needs of teams of nonsoftware developers. For example, the CVS[5] revision management repository is a central tool in PicoLibre. It is very useful for software development, but does not fit well for casual researchers' use. For example, although it is possible with CVS to manage the revisions of a document written with an office suite, it is not very convenient, since it requires installation of a specific CVS client program. Most nonsoftware developers (e.g., researchers in the field of finance or business administration) are thus not comfortable in using such a tool.

To build on top of our previous developments, and considering the aforementioned perspective, we proposed to start and build the ProGet platform using some of the free software modules that had been used to create PicoLibre, and to combine these modules with other existing libre software projects to fulfill new capabilities. We intended also to do our developments and the integration of the numerous applications necessary in a much more maintainable way.

STRUCTURE OF THE PLATFORM

The ProGET platform is composed of several specialised software applications, installed together

on a dedicated machine running the GNU/Linux operating system (Debian). This article does not describe all of the components of the system, but only the modules which provide the highest level features, and the way they have been integrated into a single platform.

The development team has been participating for a long time in communities of research and practice in libre software: development of PicoLibre (see previous work with libre software collaboration tools), organisation of the *Autour du Libre* conference, participation in the *CALIBRE* FP6 European project,[6] and so forth. We have then naturally preferred to use exclusively libre software for the development of the platform, in order to ensure the complete control of all technical aspects, and the conformance to open standards.

Integration Principles

To deliver the required features, we have integrated different existing libre software applications that we will list in the section entitled, Integrated Libre Software High-level Applications. Most of these applications rely on the Apache Web server, in particular for the PHP execution engine.

Each one of the products, taken apart, was often not covering all required features. So instead of trying to add to one of these applications the missing features, we have preferred to integrate several rather specialised applications. We are therefore able to take advantage of the best aspects of each, even if there is a risk of partial redundancy. For instance, phpGroupware provides a wiki module, but a rather limited one. We have preferred to plug phpGroupware and TWiki, an advanced wiki engine, instead of engaging in a tougher effort of enhancement of phpGroupware.

In each integrated libre software application, we have been careful to allow the possibility of a logical *partition* of the data stored in the application, to ensure that the research projects can be kept autonomous and enhance privacy (see data partition).

To make sure users will not have to deal with a set of authentication tokens (login + password), the authentication will be done in the same way in all the modules, although they may be operated by separate applications. This is done in a classical way, by relying on interaction with a dedicated directory implemented by OpenLDAP. Some components also share a database stored in a dedicated MySQL server.

Integration of the applications for the realisation of ProGET was done in a rather traditional way, by sharing the lowest layers (OpenLDAP, MySQL) and with adapters (code: *glue*), when the applications themselves were rather monolithic. It would have been preferable to take advantage of libre applications organised in a more modular way and supporting a Web service paradigm. But very few mature versions of libre software which may have been integrated offered such interoperability mechanisms at the start of our project. Even today, this kind of interface is not always entirely present, even if this approach seems to be preferred now for the future versions of projects like phpGroupware or Sympa, for instance.

Integrated Libre Software High-Level Applications

The main libre software applications that have been integrated in the ProGET platform are:

- **The Apache Web server.** The server provides the link between the client tools (browsers) and the Web applications.
- **A WebDAV repository.** The `mod_dav` Apache module implements the shared Web folders which implement the documents repositories for the projects (see Project Documents Repository Accessible Through WebDAV).

- **The phpGroupware engine and applications.** PhpGroupware provides the Web interface of the "virtual desktop" of the collaborative workspace (see Researchers/ teachers Virtual Desktop in phpGroupware), and brings in standard groupware components. It also serves as the basis for the software infrastructure of the public portal. In addition to standard phpGroupware modules, we have added two new modules developed for the needs of GET. One module which extracts information out of the collaborative work applications to deliver the public portal (see Public Web Portal) and a module which enables the administrative management for the Research Office (see Tools for the Research Office).
- **The Sympa mailing list manager (Sympa).** This powerful mailing-list manager provides some asynchronous communication means inside and outside of the projects.
- **The TWiki wiki server (Wiki).** This wiki system, among the most advanced, provides a whiteboard-like interface to implement the "knowledge base" of the projects.
- **The Agata Reports reporting tool.**[8] It helps the Research Office to issue queries on the projects description forms in order to answer internal or external requests about research activity at GET.

We hereafter describe some of the characteristics of the ProGET modules and the obtained features.

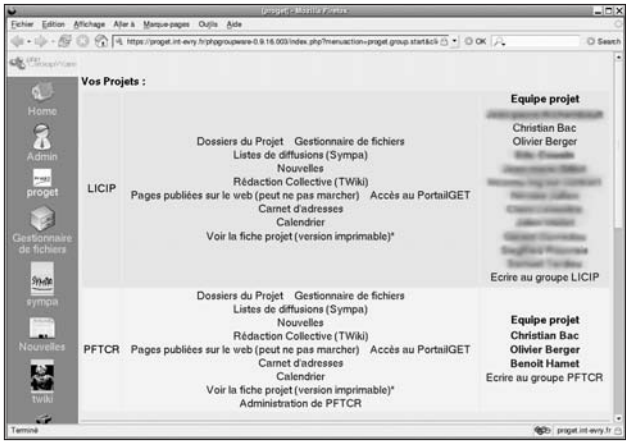
Researchers/Teachers Virtual Desktop in phpGroupware

Each researcher who connects to the platform gets access to the user's own "virtual desktop," provided through phpGroupware. Figure 2 shows this start page for a researcher who belongs to two research projects (here LICIP and PFTCR).

The user will then be provided directly, for each project the user belongs to, with a set of collaboration tools (with the corresponding links in the central column of Figure 2):

- Direct access to browsing of the contents of the documents repository of the project (see

Figure 2. ProGET Collaborative workspace start page



Project Documents Repository Accessible Through WebDAV);
- Web-based file manager (Web interface for the management of the contents of this repository);
- Mailing lists (access to the Web interface of Sympa for the lists of the project);
- News (management of the project's news displayed in the public portal);
- Project's wiki (see Project's Wiki);
- Project's specific Web pages (see Project Dedicated Web Sites);
- Access to the project's description page in the GET research portal (see Project Dedicated Web Sites;
- Shared address book and calendar (standard phpGroupware tool);
- Printable project form (including restricted access information).

Some of these tools are described in more detail hereafter.

Project Documents Repository Accessible Through WebDAV

As described in Dridi and Neuman (1999), WebDAV suits well to create a groupware portal that helps people share their documents. In ProGet each project is provided, for its internal use, with a unique secured documents repository, in the form of a specific Web folder. It is shared among members of the project team. It can be accessed via HTTPS for browsing or through WebDAV (Goland, Whitehead, Faizi, & Jensen, 1999) (over HTTPS) for modification of contents of the repository. Figure 3 shows the contents of a repository through HTTPS in a Web browser (here, the `public_html` subfolder in project PFTCR's folder).

The use of the WebDAV protocol enables transparent access in browsing or modification in the office applications, allowing for instance drag-and-drop in file managers for uploading. The uploading is thus extremely simplified[9] com-

Figure 3. Contents of a project's DAV folder (Web folder)

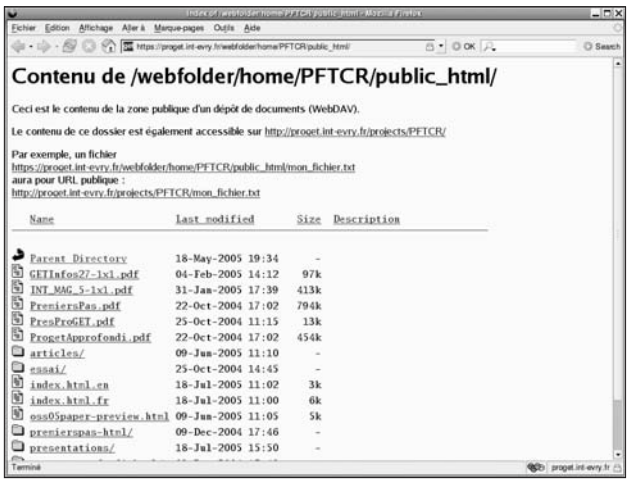
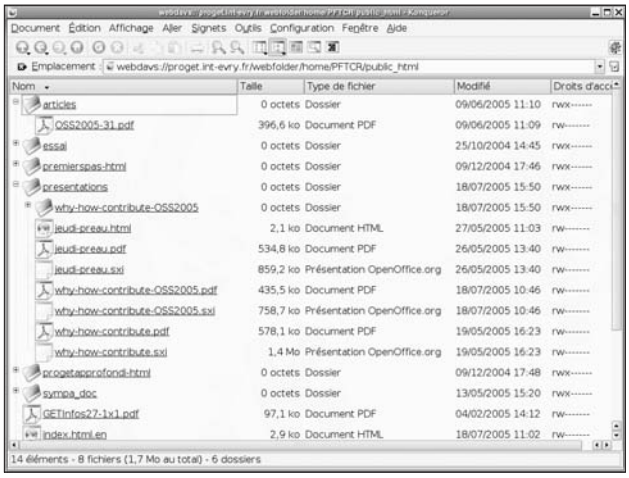


Figure 4. Contents of a project's DAV folder in KDE



pared to the use of protocols like FTP or CVS. DAV compatible clients exist on all platforms, for instance, under KDE (see Figure 4) or on Windows XP.

A simple file manager is also available as a Web application in the phpGroupware collaborative work environment of the project, for cases where the nomad users will not have the possibility to access the repository with DAV clients.

A specific subsection of each project's documents repository, corresponding to the **public_html** subdirectory, may contain the public Web site specific to the project (see Project Dedicated Web Sites). This subtree can then host a set of public pages very easily maintained, in addition to the stereotyped information displayed in the project's form on the portal. Figure 3 shows the contents via DAV of this subdirectory, which is displayed, in this case, as the contents of a folder, a list of documents, instead of as a set of Web pages (pages would be displayed as in Figure 6).

Project's Wiki

Each project is provided, internally, with a *wiki*[10] which can be used to share, in an authenticated way, in the form of Web pages, a set of simple hypertext documents, not necessarily very structured up front.

This wiki can serve many uses. Each project may for example use it as a hypertext repository to set up a knowledge base, which will hold all elements of the project's life, its productions, and ongoing works. The wiki enables the progressive elaboration of information, in a way less structured and more open than by using the project's documents repository.

Of course, the TWiki wiki engine provides all classical features of these tools (most recent modifications list, notifications, search tool, backlinks, etc.) allowing users to better apprehend the mass of hypertextual information such a tool can host. Figure 5 shows an example of a list of most recent changes in a project's wiki.

Figure 5. Most recent changes in a project's wiki

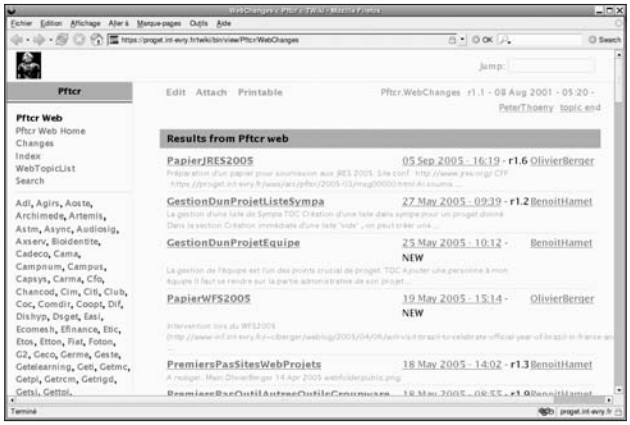


Figure 6. Dedicated project's Web site for project PFTCR

A portion of the wiki can be used to put online hypertextual contents in the project's public Web site, in an automated way (see Project Dedicated Web Sites). To this end, the wiki pages must have been identified individually as *public*. This feature may be used for instance to prepublish on the Web some of the information being elaborated inside the project, which has not yet achieved a sufficiently finalised or structured form required for publication of the documents available on the project's Web site. This is a way to offer some means of transparency on the research works,

by allowing the public to directly view the "lab notepads" of the projects.

Project Dedicated Web Sites

On the same platform, each research project benefits from several dedicated publication spaces, for its external communication on the Web.

Stereotyped Page in the Public Portal

This page describes briefly the research activity and the team. It represents a stereotyped description of a project which has been validated by the Research Office. In addition, the page contains a dedicated news zone for the project which is updated directly by each project manager. This zone helps spreading news about the life of the project. To this end, an RSS[11] feed comes alongside each project's news pages.

Project stereotyped pages are created in the research portal through a phpGroupware module named **sitemgr**. This module is configured specifically and relies on information extracted from the administrative database of project forms.

Specific Web Space

This Web site is made of static HTML pages. The content of this space is managed directly by each project, at will. It may be produced with the HTML editing software of its choice. The upload is made directly by members of the project via publications in the **`public_html2`** subdirectory of the project's WebDAV folder (see Project Documents Repository Accessible Through WebDAV).

No specific competence is required in the teams to be able to contribute to the project's Web site. Figure 6 shows an example of such a Web page. The URL to access this Web space is stereotyped from the project's acronym: `http://proget.intevry.fr/projects/PROJECT_NAME/`

Wiki Excerpts Publication Space

This space complements the previous spaces by allowing the direct publication of excerpts (pages declared *public*) of the resources elaborated by the project inside its internal wiki (see Project's Wiki). This may help with prepublishing, issuing short reference papers, providing a knowledge base, and so forth.

GET Research Projects Administrative Management Module

Whereas other modules of the platform existed previously as generic libre software tools fulfilling the classical needs of every organisation, this project administrative management module has been developed specifically for the needs of the GET Research Office. This module implements the workflow of the contents of research projects forms, during the life cycle of GET research projects (creation, approval, yearly updates, etc.). It was developed in PHP on top of the phpGroupware API, by using the **templates** component, which allows the creation of input masks and screens for the update of the data stored in the underlying MySQL database.

It also uses a set of query and report templates built with *Agata Reports*, a tool for creation of database queries without programming skills, in Web mode. An expert of the domain, who knows the database schema, can then improve the set of queries initially integrated in ProGET, for instance, to produce complex reports in the form of spreadsheet files or even synthetic booklets in the form of word-processor or PDF files.

Data Partition

Inside the same collaborative work platform, each research project must be able to host its own information and documents, which may be confidential. Thus, a partition of the whole data handled by ProGET was set up, relative to the

project groups. This applies in particular to the documents stored in the document repositories (including its dedicated Web site), the messages on the mailing lists in Sympa, the information from the projects description forms in restricted access, and some of the information in the phpGroupware tools.

In a classical way, the OpenLDAP directory is used to allow the authentication of the user's work sessions with a unique password for the different tools integrated in the same platform. The directory is also used as a reference for the description of project teams and their access authorisation for the data handled by the different applications. The system relies on persons and groups defined in a single place, in the LDAP directory. The information are managed through the users and groups management modules in phpGroupware.

A list of persons responsible for the project and a list of participants are associated to each research project. Members of the first group have the right to modify the list of members of the second group via the Web interface in phpGroupware, and to add new co-opted members in their group of responsible persons. The management of the teams for the collaborative work is then done directly in the projects, in an autonomous way, for a better reactivity.

In each of the applications integrated in the platform, access to the data relating to the research projects is controlled by checking if one belongs to these groups. For instance, Sympa is configured to query OpenLDAP directly, which helps defining automatic subscription of members of a project group to a mailing list. That way, each research project receives automatically a discussion list whenever it is created. The record of discussions on this list is only accessible to its subscribers, the members of this research project.

For the phpGroupware tools (calendar, address book, news, etc.), the partition of the data is implemented on the basis of stereotyped phpGroupware *categories*. So each project group defined in the underlying OpenLDAP directory receives a corresponding phpGroupware category automatically, which is then used as an umbrella for classifying the information.

In TWiki, this principle is implemented through standard mechanisms which are devised to host several autonomous wikis on the same site (known as *Webs* in TWiki). All the pages of a project are contained in a TWiki Web, that is a dedicated autonomous wiki. The creation of this space is done at the time a group of users is created, and corresponding privileges are granted to this group (allowed to modify pages, to add new members to the group, etc.). Unlike the other applications integrated in ProGET, it was not possible to let TWiki (at least the standard version we used) interface directly with the LDAP directory for the definition of groups of persons. An adapter module has been developed, to synchronise the groups defined in TWiki with the contents of the research project teams.

Modification and Customisation Efforts

Most of our software development effort has been targeted on:

- **phpGroupware custom components** for administrative management of research projects (see Tools for the Research Office) and Web portal (see Public Web Portal). These developments were very specific to the needs of GET, and should not be released.
- **phpGroupware low-level components.** We have in particular improved the interaction mechanisms between phpGroupware and OpenLDAP, and the code used to access WebDAV. These developments do not specifically fit only our own needs, and improve in a sensible way the standard phpGroupware platform.
- **Bug fixes** and modifications to the other components. Many small modifications

and bug fixes were necessary in particular on the WebDAV server, and in the LDAP server interface of Apache, or on the *TWiki* and *Sympa* programs. These modifications make it easier to integrate these components on top of OpenLDAP. These developments were contributed to the community in the form of patches when they were of general interest.

STRATEGY FOR CONTRIBUTION

This section deals with some methodological aspects of the development project of the ProGET platform. Our project is more an in-house project than an Open Source project and thus case studies like Mockus, Fielding, and Herbsleb (2002), do not always apply to it. Despite the fact that the project was not open to outsiders, it relies profoundly on libre software and we felt the need to set up a strategy for contributing to libre projects. The strategy was adopted in order to establish the conditions of a greater maintainability of the developed system and a clean integration of our patches and improvements to the original projects. These aspects have been described in more detail by the authors in a previous article; see Bac, Berger, Deborde, and Hamet (2005).

Benefits of Libre Software for Their Integration in In-House Applications

Libre software offer today an opportunity to lower the duration and cost of software projects. They also help to raise the quality of the applications (Bauer & Pizka, 2003). The software components are numerous and available for integration or adaptation in order to create applications that fit the needs of organisations (Wheeler, 2005). The development of the initial version of an in-house application can be done quickly, even in a "brutal" way, by modifying components in order to combine them. To be able to use them for its internal projects, one may not be forced to participate directly, or in an indirect way, to their development in the libre communities.[13] These tasks of integration can then be done in a classical way, as if the components had been completely developed internally, without trying to contribute to the initial "upstream" projects. But it is also possible, and even advisable, to contribute, whenever it is possible.

Maintainability of In-House Specific Developments

After the initial integration phase comes the issue of the midterm evolution of the obtained product. This issue becomes crucial when it becomes necessary to react to recurring external evolutions of the integrated components (for instance in case of a published fix for a security problem). We profess, out of the experience gained during this project, that it is necessary to define a policy for contributions on the most important integrated components.

This active contribution brings the possibility of transferring the evolutions that were required by the internal developments into the libre modules that were used. Once they have been adopted by the external projects, they will become part of all future releases of these components, which will remove the need to keep on applying internally some specific patches each time a new update is issued for the original module. In this respect, some amount of the maintenance effort for the internal solution is "outsourced" to the libre community, which helps lower the internal costs.

Strategy Adopted

To make some adaptation on phpGroupware components, we have not limited ourselves to only modifying internally this software (as it had been done before for PicoLibre), but on the contrary, we have decided to collaborate actively with the phpGroupware project. This led us to

undertaking the process for the official application of a member of our team to the phpGroupware project, and the transfer of the copyright on our developments to the FSF which holds the rights for phpGroupware.

Again, our goal, by doing so, is to have our modifications integrated in the standard code base of phpGroupware, so that they will be part of future releases of the software, and we may not have to apply again and again the same changes whenever a new version of phpGroupware is released. This way, we intend to lower the cost for the future maintenance of ProGET.

FIRST OUTCOMES AND PROSPECTS

Even if integration of existing applications requires extensive adaptation (Adams, Boldyreff, Nutter, & Rank, 2005), the effort is far lower than what would have been necessary for a project entirely developed internally. The vast range of features that we thought necessary for our platform was only achievable at a reasonable cost by taking advantage of existing libre software. This platform brings generic tools for nonprogrammer researchers but also features appealing to the most demanding researchers, like DAV repositories or a wiki.

The ProGET platform is in production now and opened for its users since June 2005. The GET research portal and the projects administrative management application are already used on a day to day basis. It is however too early to proceed to an assessment of the use of collaborative work tools, which are still used regularly by only a few research projects.

The use of the collaborative work environment is at the present time proposed for volunteers. Its roll-out to the whole GET research teams is not yet envisioned, some uncertainties remaining on the general infrastructure to be set up, for instance, for the necessary training phase of the users, or the financial costs to ensure the maintenance of the production platform with maximum availability. Some findings can still be learned from the return on experience of the first users.

Conformance to Requirements

The tools which have been integrated offer a far range of features, which address most of the generic needs of the research teams at GET. The TWiki tool, quite versatile, provides in particular a great potential, although it requires some amount of training. Some shortcomings have however appeared during the first months of usage of the platform.

ProGET was devised by focusing on an organisation structured around the notion of research project with a clearly defined perimeter, validated by the GET Research Office. However, needs for collaborative work tools also come from contexts outside the precise boundaries of already validated research projects. Thus, during upstream exploratory phases leading to the creation of projects, for instance, some collaboration is often necessary between members of teams which have already been established independently, and who often work part-time on these exploratory activities. Several users have requested that the kind of tools available in ProGET be offered also for these upstream phases, where they could greatly facilitate participation, reactivity, and communication between various actors. It then seems necessary to be able to handle also collaborative work groups linked to exploratory phases, outside the rigid frame of validated research projects already approved for the GET research portal.

It was also identified, through the first steps of take-up of the tools by research teams, that the principle of partition of data that was implemented, based on a lowest granularity corresponding to the research projects, was not fitting that well. Inside a GET research project, teams have to work for different research contracts, which all have their own list of partners, and their own privacy require-

ments. The granularity chosen for teams which have to access the same set of documents and information in the project is thus not well adapted in the current version of the platform. It seems necessary to be able to define, for collaborative work, some subprojects more or less autonomous in the realm of the same research project, which will each define a specific list of participants and associated access privileges.

Evolution Towards Other Collaborative Activities

The users of the platform are also participating, besides their research activities, in teaching activities, and numerous other work groups, for which Web-based collaborative work tools may not always be available, either at GET or at its partners.'

There is thus a need for the availability of one or more platforms supporting collaborative work, which would be as feature-full as the ProGET platform, but which may allow the easy and quick creation of projects and their work groups, progressively, in a loose environment, as was the case with the PicoLibre platforms.

Need for such a platform could lead to the creation of a "new generation" PicoLibre system, which could integrate the whole set of generic components that have been integrated in ProGET, together with a generic module for creation of collaborative work spaces, and which could be more maintainable than the previous generation.

The ProGET platform could then be split in two parts: on the one hand a module for the administrative management of GET research projects and a GET research Web portal, and on the other hand the use of a "PicoLibre V2" kind of platform, in which the groups corresponding to the GET research projects could be hosted, among others.

Constraints for Administration

The ProGET platform was devised in an effort for reuse and modularisation in integrating the existing libre components, in order to lower the maintenance efforts to be made internally (see Strategy for Contribution). However, the installation and maintenance tasks for such a platform are quite demanding and require specific competence (good knowledge of the internals of the various integrated software), which imply a necessary precise planning in the event of the outsourcing of its administration to the regular IT department which is in charge of the GET information systems.

In this respect, one first hypothesis for improvement would be the availability of the whole set of components in a standard packaging. It would be interesting to have all the components used, and their new adapters, available in the form of packages for the libre software distributions already known by the support teams. For instance, in the Debian distribution, the phpGroupware, Sympa, and TWiki tools are already packaged in the ordinary way. It would be interesting if all modules that have been developed for ProGET were packaged too. This could be achieved by taking advantage of the recent efforts made to package the modules of the PicoLibre application for Debian.

PhpGroupware for the Development of Custom Applications

Several findings can be stated, looking at the use of phpGroupware as the basis for the development of custom applications. PhpGroupware brings a large choice of components which have not all achieved a common level of quality. Some functions suffered from numerous bugs preventing them to be put into production as such, in particular regarding the conditions required for ProGET, for instance if using underlying OpenLDAP and WebDAV layers.

In addition, the structure of data in phpGroupware is in general organised from the perspective of one phpGroupware instance deployed in an organisation where, by default, the whole of the information handled by the system will be available to all the users. This may fit, for instance, to SMEs or very small virtual organisations where all actors have to be informed of everything happening in the organisation. This model happened however to be poorly convenient in our case which requires that projects hosted on the same phpGroupware platform be kept separate, and that a strong level of confidentiality be preserved on the information (see Data Partition).The use of phpGroupware categories partially helps to solve this kind of constraint, but by paying the price of some usage precautions which degrade the global usability of the application.

PhpGroupware provides an interesting API for the PHP developer, for Web based collaborative work applications that will be relatively simple. It happens to be somehow limited though, for advanced needs for the development of complex applications, in particular concerning the ACLs, the corresponding data model, and the independence of the lowest level modules. PhpGroupware lacks also of a module providing applicative workflow facilities.

The use of the *template* "RAD tool" integrated in phpGroupware turns out a poor choice in the case of a complex application, despite its ease of use for relatively simple needs. Indeed, its associate documentation and tutorial only scarcely refer to the *MVC* model which is however generalised in the more complex modules of phpGroupware. In the latest modules developed for ProGET, we thus have preferred to use these more classic MVC *patterns* and rely more on the HTML templating system, that happen to be more maintainable. This choice was made with a midterm vision, as the learning curve for developers is bigger in this case.

PhpGroupware is still an interesting project for a rather simple use of its existing modules. And although it provides rich and quite generic APIs, which allow the development of new applications, it cannot constitute a really generic application development *framework* like other libre environments (Zope, Apache Tomcat, etc.).

CONCLUSION

Although it may be enhanced on numerous aspects, the ProGET platform delivers a range of features without competition in existing libre software collaborative work platforms, for a modest development cost.

There are strong needs in higher-grade research and teaching institutions for tools supporting collaborative work. ProGET can then constitute a reference point for organisations wishing to integrate, for their custom needs, existing libre software applications for collaborative work.

Even if a policy is adopted for the contribution to the libre software projects used, in order to lower certain maintenance costs, the generalisation of the use of the present platform at GET and the future developments necessary to enhance it will only be possible through a substantial investment, which may be far more important than what was spent for the first initial developments.

One possibility for the reduction of these costs could be the mutualisation of the development between several organisations. It could be articulated around the industrialisation as well-packaged libre software programs of the most generic elements of ProGET, leading the way to a new generation PicoLibre platform.

REFERENCES

Adams, P., Boldyreff, C., Nutter, D., & Rank, S. (2005, May 17). *Adaptive reuse of libre software systems for supporting on-line collaboration*. In *Proceedings of the 5th Workshop on Open Source Software Engineering*, St. Louis, Missouri.

Bac, C., Berger, O., Deborde, V., & Hamet, B. (2005, July 11-15). *Why and how to contribute to libre software when you integrate them into an in-house application? In Proceedings of the First International Conference on Open Source Systems*, Genova.

Bauer, A., & Pizka, M. (2003). The contribution of free software to software evolution. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE).*

Cousin, E., Ouvradou, G., Pucci, P., & Tardieu, S. (2002). *PicoLibre: A free collaborative platform to improve students' skills in software engineering.* In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.*

Dridi, F., & Neumann, G. (1999, June). How to implement Web-based groupware systems based on WebDAV. In *Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.*

Mockus, A., Fielding, R.T., & Herbsleb, J.D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering Methodology, 11*(3), 309-346.

Goland, Y., Whitehead, E., Faizi, A., & Jensen, D. (1999). HTTP extensions for distributed authoring. In P*roceedings of WEBDAV.* Retrieved June 16, 2006, from http://Webdav.org/

Wheeler, D.A. (2005). Why open source software/ free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers! Retrieved June 16, 2006, from http://www.dwheeler.com/oss_fs_why.html

ADDITIONAL RESOURCES

Web site of the OpenLDAP project: http://www.openldap.org/

Web site for the phpGroupware project: http://www.phpgroupware.org/

Web site for project PicoLibre: http://www.pico-libre.org/

SourceForge project hosting platform: http://sourceforge.net/

Sympa project's Web site: http://www.sympa.org/

TWiki project's Web site: http://twiki.org/

"Wiki" article in the Wikipedia encyclopedia: http://fr.wikipedia.org/wiki/Wiki

ENDNOTES

1 http://www.get-telecom.fr/

2 The GET research portal can be accessed at http://proget.int-evry.fr/.

3 The screenshots of the application provided in figures of this chapter display pages in French. Not all interface elements have been provided with a translation to English yet in the ProGET application.

4 GNU General Public licence: http://www.gnu.org/copyleft/gpl.html

5 Concurrent Versions System: http://www.nongnu.org/cvs/

6 FP6 IST Project 00433 "Coordination Action for Libre Software": http://www.calibre.ie/

7 DAV module for Apache: http://www.Web Webdav.org/mod_dav/

8 http://agata.org.br/

9 The counterpart of this simplification, compared to CVS in particular, is that the repository does not support documents versioning on its own. This could be enhanced in a future version of ProGET by using *Subversion* (http://subversion.tigris.org/) instead of the mod_dav module used today, which would bring the platform closer to what exists in the

recent Trac software development platform (http://projects.edgewall.com/trac/), which combines a Wiki and Subversion.

[10] The authors assume that the reader is already familiar with this kind of tool which has become very popular in the latest years. For more details, refer to (wiki) and to the Wikipedia project which hosts it, for a good example of application of wikis.

[11] Really Simple Syndication: http://blogs.law.harvard.edu/tech/rssVersionHistory

[12] The name "public_html" is a convention adopted after Web server Apache's naming policy.

[13] Note that contrary to some rumors, it is in no way mandatory to publish the modifications that one has made on a libre software program, as long as this software is not distributed to third parties, even if it was initially published under the GNU GPL.

This work was previously published in the International Journal of Information Technology and Web Engineering, edited by E. Damiani & G. Succi, Volume 1, Issue 3, pp. 1-16, copyright 2006 by IGI Publishing (an imprint of IGI Global).

Chapter 7.23
Exploring the Effects of Process Characteristics on Product Quality in Open Source Software Development

Stefan Koch
Vienna University of Economics and Business Administration, Austria

Christian Neumann
Vienna University of Economics and Business Administration, Austria

ABSTRACT

There has been considerable discussion on the possible impacts of open source software development practices, especially in regard to the quality of the resulting software product. Recent studies have shown that analyzing data from source code repositories is an efficient way to gather information about project characteristics and programmers, showing that OSS projects are very heterogeneous in their team structures and software processes. However, one problem is that the resulting process metrics measuring attributes of the development process and of the development environment do not give any hints about the quality, complexity, or structure of the resulting software. Therefore, we expanded the analysis by calculating several product metrics, most of them specifically tailored to object-oriented software. We then analyzed the relationship between these product metrics and process metrics derived from a CVS repository. The aim was to establish whether different variants of open source development processes have a significant impact on the resulting software products. In particular we analyzed the impact on quality and design associated with the numbers of contributors and the amount of their work, using the GINI coefficient as a measure of inequality within the developer group.

INTRODUCTION

In recent years, free and open source software (OSS) has drawn increasing interest, both from the business and academic worlds. Projects in different application domains, like most notably the operating system Linux, together with the suite of GNU utilities, the office suites GNOME and KDE, Apache, sendmail, bind, and several programming languages, have achieved huge successes in their respective markets. Undeniably, they constitute software systems of high quality. This has led to discussions and analyses of the underlying development process, as OSS is unique not only in its licenses and legal implications.

The main ideas of this development model are described in the seminal work of Raymond (1999), *The Cathedral and the Bazaar*, first published in 1997. Raymond contrasts the traditional model of software development, which he likens to a few people planning a cathedral in splendid isolation, with the new 'collaborative bazaar' form of open source software development. In the latter model, a large number of developer-turned-users come together without monetary compensation to cooperate under a model of rigorous peer review and take advantage of parallel debugging, which altogether leads to innovation and rapid advancement in developing and evolving software products. In order to enable this while minimizing duplicated work, the source code of the software needs to be accessible, which necessitates suitable licenses, and new versions need to be released often. Most often, the license a software is under is used to define whether it is open source software, applying for example the open source definition (Perens, 1999) or the approach of free software as embodied in the GNU GPL (Stallman, 2002). Nevertheless, usually a certain development style and culture are also implicitly assumed, although no formal definition or description of an open source development process exists, and there is considerable variance in the practices actually employed by open source projects. Also the relationship to and insights regarding practices of agile software development (Erickson, Lyytinen, & Siau, 2005; Turk, France, & Rumpe, 2005; Merisalo-Rantanen, Tuunanen, & Rossi, 2005) have been discussed (Koch, 2004a).

Possible advantages and disadvantages to the development of software of this new development model have been hotly debated (Vixie, 1999; McConnell, 1999; Bollinger, Nelson, Self, & Turnbull, 1999; Cusumano, 2004; Feller, Fitzgerald, Hissam, & Lakhani, 2005). For example the question of whether open source development positively or negatively impacts quality and security has been a topic of several analyses (Witten, Landwehr, & Caloyannides, 2001; Hansen, Köhntopp, & Pfitzmann, 2002; Payne, 2002; Stamelos, Angelos, Oikonomou, & Bleris, 2002; Koru & Tian, 2004; Feller et al., 2005). Different viewpoints have also developed regarding whether or not the open source development approach increases efficiency of software production (Feller et al., 2005). Critics argue that the largely missing requirements engineering and design phases, together with the trend to search for bugs in the source code late in the lifecycle, lead to unnecessarily high effort hidden by the relative ease of spreading it throughout the world (McConnell, 1999; Vixie, 1999). Proponents of the OSS development model counter with arguments of very high modularity, fast release cycles, and efficient communication and coordination using the Internet (Bollinger et al., 1999; Raymond, 1999).

Currently, much empirical research is proceeding on OSS processes. Often, the research relies on data available through mining the communication and coordination tools and their repositories (Cook, Votta, & Wolf, 1998; Dutoit & Bruegge, 1998; Atkins, Ball, Graves, & Mockus, 1999; Kemerer & Slaughter, 1999) in place in OSS projects in order to describe and characterize the development team and processes. Most notably, the source code control systems used have been found to be a source of information, together with mailing lists and bug tracking systems. These analyses

have been useful in providing an indication of how OSS development works in practice. Work performed has included both in-depth analyses of small numbers of successful projects (Gallivan, 2001) like Apache and Mozilla (Mockus, Fielding, & Herbsleb, 2002), GNOME (Koch & Schneider, 2002), or FreeBSD (Dinh-Tong & Bieman, 2005) and also large data samples, such as those derived from Sourceforge.net (Koch, 2004; Long & Siau, 2007). Primarily, information provided by version control systems has been used, but so have aggregated data provided by software repositories (Crowston & Scozzi, 2002; Hunt & Johnson, 2002; Krishnamurthy, 2002), meta-information included in Linux Software Map entries (Dempsey, Weiss, Jones, & Greenberg, 2002), or data retrieved directly from the source code itself (Ghosh & Prakash, 2000). Other approaches taken include ethnographic studies of development communities (Coleman & Hill, 2004; Elliott & Scacchi, 2004), sometimes coupled with repository mining (Basset, 2004). Indeed, it can be shown that important information about project characteristics and participating programmers can be retrieved in this fashion.

However, a key problem is that the resulting process metrics (Conte, Dunsmore, & Shen, 1986; Fenton, 1991; Henderson-Seller, 1996) measuring attributes of the development process and of the development environment, such as distinct programmers, number of commits, or inequality, do not address the quality, complexity, or structure of the resulting software product. Therefore, we expanded the analysis in this article by selecting and calculating several product metrics pertaining to these characteristics of the software product.

This allows us to analyze whether different development practices have an impact on product quality. We will use process metrics derived from the respective source code control systems as predictors for quality as portrayed by relevant product metrics. Uncovering these relationships will answer the question of which values for these variables—for example, low inequality in participation—lead to a higher product quality. For this analysis, we use OSS Java frameworks as a data set. The most similar work available is by Koru and Tian (2005), who have used two large open source projects as a dataset to uncover a relationship between high-change modules and those modules rating highly on several structural measures. They used, among others, size measures such as lines-of-code or number of methods, coupling measures such as coupling between objects, cohesion measures such as lack of cohesion in methods, and inheritance measures such as depth in inheritance tree.

The research objective of this article therefore is as follows: We investigate whether there is an influence of different forms of open source software development processes characterized by process metrics on the resulting software. Most importantly, we check for impacts on different quality aspects as measured by appropriate product metrics. A comparison with proprietary products and processes is out of scope and will not be treated in this study.

In the following section the method employed for arriving at the necessary data is described, starting with the data set chosen and its importance, and proceeding to the data collection of both product and process metrics and their combination. Then we present the analysis regarding any relationship between process and product metrics, both on the level of classes and of projects, followed by a discussion. The article finishes with conclusions and future research directions.

METHOD

Data Set

For this empirical study, a certain fixed domain of OSS was chosen, in order to limit variance to the areas of interest by holding the application domain constant. All projects included therefore roughly implement the same requirements and

with the same programming language, so differences in software design and quality can directly be attributed to different development practices in place.

We examine 12 OSS frameworks for the presentation layer of Web applications. A framework is a reference architecture for a concrete application which offers basic structures and well-defined mechanisms for communication. Only specific application functionality has to be implemented by the programmer, which is achieved by using abstract classes and interfaces that have to be overridden (Johnson, 1997; Fayad & Schmidt, 1997). All frameworks are based on J2EE components like JSP, Servlets, and XML, and can be used within every Servlet container that implements the J2EE standard. The frameworks are: ActionServlet, Barracuda, Cocoon, Expresso, Jetspeed, Struts, Tapestry, Turbine, Japple, Jpublish, Maverick, and Echo.

Besides having a fixed domain thus reducing any noise in the results, frameworks are an important part in modern software development. Frameworks are one possibility of reusing existing software, thus promising reduced costs, faster time to market, and improved quality (Morisio, Romano, & Stamelos, 2002). OSS especially lends itself to white box reuse (Prieto-Diaz, 1993), as it per definition contains the source code, offers a deeper view into the architecture, and may be modified or adapted. This reduces the disadvantages encountered with using components-off-the-shelf (COTS) offered by software companies. Another critical issue that can be solved by using OSS is the maintenance of frameworks, which is usually done by the contributors of project. On the other hand, although the source code is available and the program could be maintained by the community, some serious problems could accompany the development process, due to low-quality code, design, or documentation. Object-oriented metrics as used here provide a capability for assessing these qualities (Chidamber & Kemerer, 1991, 1994) and may help to estimate the development effort for adaptation and adjustment.

First, all classes are treated as a single data set; afterwards an analysis on project level is presented. An analysis on class level is performed for two reasons: As we analyze the development process and style, the differences between classes might be larger than those between projects, and indeed for some metrics the variation is higher within the projects than between them. For example an abstract class for database access might be developed similarly in all projects. We therefore might find paired classes among different projects. In addition, using a framework does not necessarily mean adopting all classes within this framework. Therefore an analysis on this detailed level is of interest out of a reuse perspective. Afterwards, we will try to consolidate both perspectives by using multilevel modeling which explicitly incorporates effects on both levels.

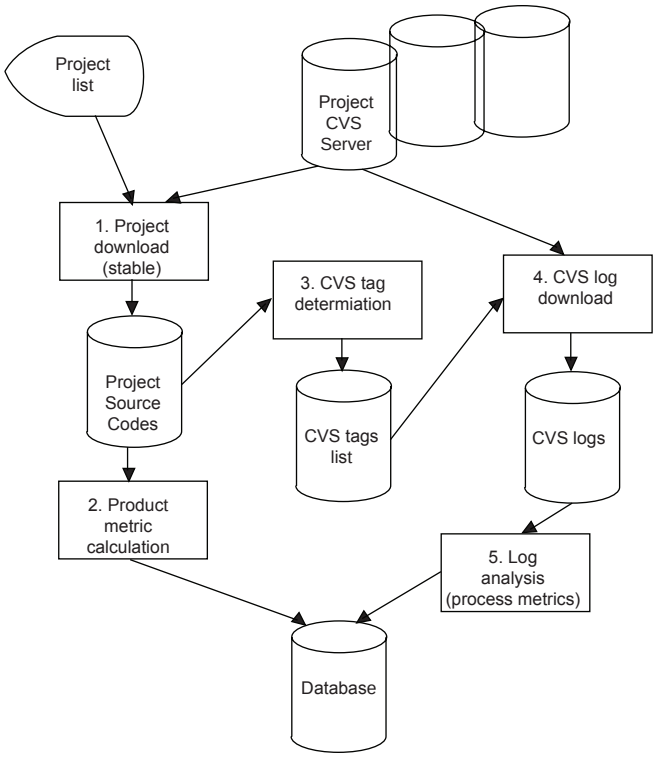
Data Collection

For the following analysis, several steps of data collection were conducted. As mentioned above, this study focuses on frameworks for Web applications written in an object-oriented language. Many of the available frameworks are not written in object-oriented languages but scripting languages like Perl or PHP. This would preclude using most of the product metrics designed for object-oriented languages. Therefore we focused on frameworks written in Java. We conducted preliminary research to identify potential candidates that fulfilled the criteria of both language and application area. This initial phase consisted of performing extended Web research (online developer forums, search engines) and perusing reports in professional publications for developers. This led to the identification of 12 frameworks. The functions and features of the resulting frameworks were compared in a prior study (Neumann, 2002) and are not part of this article.

After the data set as defined above had been identified, both product and process metrics had to be retrieved and merged for further analysis. In order to calculate the product metrics, the latest stable version of each framework was determined and downloaded as a packed distribution. We used the metric plug-in (*http://metrics.sourceforge.net/*) for the Eclipse SDK (*http://www.eclipse.org*) to calculate these product metrics. The necessary compilation of the downloaded source files required utilization of stable versions over the current snapshot from the source code repository, the latter of which might produce complications due to inconsistent code and the exclusion of additional libraries. The plug-in creates an XML representation of the calculated metrics which we used in our study. This is done for source code files only (i.e., .java-files in Java). A simple Java program was written to process this XML file and to store the metrics on class level into a database. The resulting product metrics will be described in the next section.

To retrieve the required process metrics, we used the methodology applied in other studies (Mockus et al., 2002; Koch & Schneider, 2002; Robles-Martinez, Gonzalez-Barahona, Centeno-Gonzalez, Matellan-Olivera, & Rodero-Merino, 2003; Dinh-Tong & Bieman, 2005; Hahsler & Koch, 2005), relying on mining the source code control repositories, for the data set in all cases of the concurrent version system (CVS). First, we looked up the CVS tag associated in the repository with the stable version already downloaded. Using this information, a local checkout of the files

Figure 1. Data-collection process



was performed, and a log file was generated from the initial check-in until the corresponding date of the stable release. This assures that the same source code is used to calculate both the product and process metrics. Data from the log files were extracted for every check-in for every available file in the local CVS repository. Once extracted, these were stored in a normal database as has been done in prior studies (Fischer, Pinzger, & Gall, 2003; Koch & Schneider, 2002; Koch, 2004; Hahsler & Koch, 2005). Each database entry therefore consists of the filename, the name of the committer which was anonymized for privacy reasons (Thuraisingham, 2005), LOC added and deleted, and the date. The end result was a total of 45,164 records within a single table. We then used database queries to calculate process metrics, for example, overall commits, number of different committers, and so on, for each class (i.e., .java-file). Using another program, additional metrics like the standardized GINI coefficient were computed for every file and again stored in the database. The product and process metrics were merged using the file name as a unique key, resulting in one entry for every class containing both types of metrics. We therefore only consider source code files (i.e., .java-files) and exclude additional files possibly found in the CVS repository, like documentation files or the projects' Web sites. Figure 1 gives a graphical overview of the data-collection process.

Description of Process Metrics

In selecting the metrics used in this study, we both considered the goals of the analysis (i.e., to be able to both characterize the software process and quality aspects of the resulting product) and the availability of metrics within the data. We use several well-discussed process metrics to characterize the OSS development processes in the projects analyzed. The metric of commit refers to a single change of a file by a single programmer. Therefore the number of commits of a file is the sum of changes conducted over a certain period of time and is also an indicator for the activity of a file. In our study we cover the time from the initial commit of a file until the last commit before the stable version was released. The total lifetime of a file includes all the time elapsed, not only that time which was spent on developing and coding. Another important process metric is the total number of distinct programmers involved in writing and maintaining a file. A programmer is defined by counting those people committing source code changes through their CVS account, thus only people with such accounts are measured. In some projects, depending on the change and commit policy in place, people could be contributing code without CVS account, which sometimes is only granted to long-time participants, by sending it to one of those persons who then does the actual commit. For example, German (2006) found that 110 out of 364 modification records of a user were patches submitted by 46 different individuals. Therefore, the number of programmers might actually be higher than the number reported here. This fact is very problematic to check. In general, there are several possibilities of attributing authorship of source code to persons, which are to use the associated CVS account (as done here), to mine the versioning system comments for any additional attributions, to infer from attributions in the source code itself, or by questionnaires or intimate knowledge of a project and its participants. Attributions in source code or commit comments are highly dependent on existence and form of a project's standards, and therefore are also difficult to implement for larger data sets. Ghosh and Prakash (2000) have implemented a solution based on source code attributions for a set of more than 3,000 projects, with about 8.4% of the code base remaining uncredited, and with the top authors containing organizations like the Free Software Foundation or Sun Microsystems. Nevertheless, they have found a similar distribution of participation as found in this study's data set, as have most other approaches like question-

naires (Hertel, Niedner, & Hermann, 2003) or case studies of larger projects (Mockus et al., 2002, Koch & Schneider, 2002; Dinh-Trong & Bieman, 2005). In a case study of the OpenACS project under participation of project insiders and using the strict standards for CVS comments, Demetriou, Koch, and Neumann (2006) have found that only 1.6% of revisions pertained to code committed for someone without CVS privilege. In this study, we have used two approaches for checking the validity of this measure: Using simple heuristics, we have checked all commit comments for attributions. This shows that 11.7% of revisions seem to be contributed by other people. We have also manually inspected all revisions of the Maverick project: no revision seemed to have been committed for somebody else, which was identical to the heuristics result.

As the participation of programmers in open source projects is less continuous than in commercial development, the number of programmers alone does not adequately reflect the effort invested. Therefore we include the open source software person month (OSSPM) as a new process metric that characterizes the amount of work that is committed to the object considered. This is defined as the cumulated number of distinct active programmers per month over the lifetime of the object of analysis. As Koch and Schneider (2002) have shown, this number of active programmers can be used as an effort predictor. It should be noted that this measure assumes that the mean time spent is constant between objects of analysis.

As several prior studies (Koch, 2004; Mockus et al., 2002; Ghosh & Prakash, 2000; Dinh-Tong & Bieman, 2005) have shown the distribution of effort between participants to be highly skewed and differing from commercial software development, we add an additional process metric to characterize the development style. We used the normalized GINI coefficient (Robles-Martinez et al., 2003), a measure of concentration, for this. The GINI coefficient is a number between 0 and 1, where 0 is an indicator for perfect equality and 1 for total inequality or concentration. We calculated the GINI coefficient both based on LOC added per person (which can be extracted from the CVS repository) and on the number of commits a person has done. As the further analyses did not show significant differences between both measures, we will only report the findings for the GINI coefficient based on LOC added. Therefore in the terms of OSS development, a GINI coefficient of 1 means that one person has written all the code. We performed a slight modification: As some files only have one author, calculating the normalized GINI coefficient results in 0 (equality). For these cases we changed the value from 0 to 1 because, for us, the fact that one person has written all the code is an indicator of inequality rather than equality.

Description of Product Metrics

The most popular product metric is the size of a program, which can be derived by counting the number of lines-of-code (LOCs). There are many different ways to count LOCs (Humphrey, 1995; Park, 1992; Jones, 1986). In this analysis we apply the definition used by the CVS repository, therefore including all types of LOCs: source code lines as well as commentaries (Fogel, 1999). The size of the largest method (*LOCm*) is another important descriptor in object-oriented classes which can also be measured by counting LOCs. These size metrics can be regarded as indicators for complexity as it is very difficult to read and understand classes with long methods and many fields (Henderson-Seller, 1996). Other indicators are the number of regular/static methods (NOM/NSM) and the number of regular/static fields (NOF/NSF). We propose that these size measures are affected by nearly all process metrics: If more people are working on a class, its size will increase. The same will tend to be true for the time the class exists and the number of commits performed. Especially the amount of effort invested in the

class will increase the size. Most importantly, we propose that the inequality in contributions will affect different size measures: If the class is programmed and maintained by a small team, or a small core group within a team, these participants will tend not to see the need for promoting higher modularity. This would presumably lead to them not splitting up a class, thus affecting LOC, or a method, thus affecting LOCm.

The probably most well-known complexity metric is McCabe's definition of cyclomatic complexity (VG) (McCabe, 1976). *VG* counts the number of flows through a piece of code, (i.e., a method). Each time a branch occurs (if, for, while, do, case, catch, and logic operators), this metric is incremented by one. We determined the maximum (*VGmax*) and the average (*VGavg*) method complexity on class level. Weighted Methods per Class (WMC) are part of the Chidamber and Kemerer suite, but they leave the weighting scheme as an implementation decision (Chidamber & Kemerer, 1994). In our study WMC is defined as the sum of all method's complexities (*VG*) that occur within a class. VG and WMC are indicators of how much time and effort must be spent to understand, test, maintain, or extend this component (Chidamber & Kemerer, 1991; 1994), with McCabe giving *VG = 10* as a reasonable limit for proper testing (McCabe, 1976). But this measure should be treated with special care, as this metric is based on experiences in procedural languages including C or COBOL (Lorenz & Kidd, 1995). Subramanyam and Krishnan (2003) have shown that WMC is highly correlated to LOC, which supports the thesis that LOC can be used as a low-level complexity metric. The influence of WMC on software quality was examined in several studies (Basili, Briand, & Melo, 1996; Subramanyam & Krishnan, 2003). Regarding the relationship of complexity measures with process metrics, the most important effect is proposed to exist in connection with the inequality: Analogous to the reasoning for size, complexity reduction will not be a high priority when a small core group who would know the code in any case is present. Also, classes, and software overall, tend to accumulate more complexity as time passes, if no counter-measures are taken. This will decrease maintainability, which again is less of an issue if the software is consistently maintained by a small group.

The object-oriented product metrics we investigated are mostly based on a subset of the Chidamber-Kemerer-Suite (Chidamber & Kemerer, 1991, 1994; Chidamber, Darcy, & Kemerer, 1998). The authors argued that the product metrics commonly used before were not suitable for object-oriented development (Chidamber & Kemerer, 1991). From their point of view, the modern object-oriented analysis, design, and programming processes, which encapsulate functionality and entities in objects, were too different from the traditional software engineering process. The prior product metrics were not designed to measure object-oriented characteristics like classes, inheritance, and the usage of methods and attributes. They proposed six metrics, derived from a theoretical analysis, which should be able to assist in making predictions about the complexity and quality of object-oriented programs. We used a subset of the CK-suite (NOC, DIT, WMC) for which concrete threshold values were suggested. The remaining metrics (LCOM, RFC, CBO) are not part of this study, as no threshold values are available. In addition, CBO and RFC have been found to be highly correlated with WMC (Chidamber et al., 1998), so they would not give additional information. These CK-metrics for our analysis are complemented by some of the metrics defined by Lorenz and Kidd (1995).

Number of Children (NOC) and Depth in Inheritance Tree (DIT) are metrics for the level of inheritance of a class. Chidamber and Kemerer (1994) state that the deeper a class in the hierarchy, the more complicated it is to predict its behavior and the greater its design complexity. Though this may lead to greater effort in maintenance and testing, it has greater potential for the reuse of inherited methods. In a Java environment, DIT

is defined as the longest path from the class to the root in the inheritance hierarchy—that is, the class Object. Some studies have shown that DIT is related to fault-proneness (Basili et al., 1996; Briand, Wüst, Ikonomovski, & Lounis, 1998). NOC counts the number of classes inherited from a particular ancestor—that is, the number of children in the inheritance hierarchy beneath a class. A class implementing an interface counts as a direct child of that interface. Chidamber and Kemerer (1991) expose a similar relationship between design complexity and NOC. The greater the number of children of a class, the greater is the reuse. However, an excessive number of children may indicate the misuse of sub-classing. NOC also hints to the importance of that class within the application, as well as to the corresponding additional effort likely required for testing and maintaining. NOC was evaluated by Basili et al. (1996) and Briand et al. (1998), who differ in their findings related to fault-proneness. NORM, like NOC and DIT, is an inheritance metric for class design (Lorenz & Kidd, 1995). It measures the number of inherited methods overridden by a subclass. Lorenz and Kidd (1995) state that, especially in the context of frameworks, methods are often defined in a way that requires them to be overridden. However, very high values may indicate a design problem because a subclass should extend new abilities to its super-class that should result in new method names. Similar to the other product measures, we again propose a relationship of the process metrics with these object-oriented metrics. Especially the metrics giving an indication of the use of inheritance will be affected by different process attributes, most importantly on project level. The correct use of inheritance helps in achieving a modular design which in turn allows for parallel work by many participants. In addition, it significantly enhances maintainability. We therefore propose that analogous mechanisms will be found here as for complexity measures.

We suggest two additional metrics that can be used to describe the interior design of a class. The number of classes (NCL) counts the number of classes within a class and should be either 0 for interfaces or 1 for classes. Other values indicate the utilization of interior classes, which should be avoided in object-oriented design. The number of interfaces within a class (NOI) aims at the same direction. Interfaces are used to define entry points within or even across applications and therefore should not be defined within a class but in separate files.

Most of these product metrics presented are discrete variables, where increasing (or decreasing) values are not necessarily a sign of good or bad quality, or aspects thereof. For example, whether the cyclomatic complexity VG of an entity is 4 or 6 is mostly determined by its function, and does not signal any deviation from good practice or negatively influence maintainability. Only if a certain value is surpassed does this metric give an indication of possible problems. Therefore, most of these metrics can be assigned a threshold for this purpose. Currently, there is a paucity of threshold values for the defined metrics provided by literature based on empirical studies, especially using Java. This requires us for most metrics to use the values proposed by Lorenz and Kidd (1995) for C++ classes.

Based on the threshold values in Table 1, we created dummy variables that take on the value of one or zero, depending on whether the associated metric values exceed the threshold value for that class. These dichotomous variables try to categorize the given metrics based on different aspects to be explored like size or complexity (see Box 1).

MSIZE and *CSIZE* depend on metrics that measure size, *MCOMP* on complexity, *CINH* on inheritance, and *CDESIGN* on interior class design.

Table 1. Overview of metrics with corresponding threshold values

Metric	Name	Thresh-old	Definition
NOC	Number of Children		Total number of direct subclasses of a class
NOI	Number of Interfaces		Total number of interfaces of the file
DIT	Depth of Inheritance Tree	< 6	Distance from class Object in the inheritance hierarchy
NORM	Number of Overridden Methods	< 3	Total number of methods that are overridden from an ancestor class
NOM	Number of Methods	< 30-40	Total number of methods
NOF	Number of Fields	< 3-9	Total number of class variables
NSM	Number of Static Methods	< 4	Total number of static methods
NSF	Number of Static Fields	< 3	Total number of static variables
LOCm	Lines of Code	< 24	Total lines of code of the greatest method in the selected scope
VG-max	McCabe Cyclomatic Complexity Maximum	< 10	Maximum VG for all methods within a class
VGavg	McCabe Cyclomatic Complexity Average	< 10	Average VG for all methods within a class
WMC	Weighted Methods per Class	< 65	Sum of the McCabe Cyclomatic Complexity for all methods in a class
NCL	Number of Classes	= 1	Indicates possible interior classes

Box 1.

$$
\begin{aligned}
&MSIZE && \begin{cases} 1 & if \quad LOC / NOM > 18 \vee LOCm > 24 \\ 0 & else \end{cases} \\
&MCOMP && \begin{cases} 1 & if \quad VGmax > 10 \vee VGavg > 10 \\ 0 & else \end{cases} \\
&CSIZE && \begin{cases} 1 & if \quad NOM > 30 \vee NSM > 4 \vee NOF > 9 \vee NSF > 4 \\ 0 & else \end{cases} \\
&CINH && \begin{cases} 1 & if \quad DIT > 6 \vee (NOC * DIT) > 15 \\ 0 & else \end{cases} \\
&CDESIGN && \begin{cases} 1 & if \quad NCL > 1 \vee NOI > 1 \\ 0 & else \end{cases}
\end{aligned}
$$

ANALYSIS ON CLASS LEVEL

In total, 6,235 Java classes (i.e., distinct files) have been analyzed, for which a total of 45,164 commits were made, with 2,109,989 LOCs added and 913,455 LOCs deleted. A total of 133 distinct programmers have contributed with at least one commit. The number of classes investigated therefore is considerably higher than the datasets used in former studies on object-oriented metrics (634 by Chidamber & Kemerer, 1994; 97 by Chidamber et al., 1998; 180 by Basili et al., 1996; 698 by Subramanian & Corbin, 2001; 180 by Briand, Wüst, Daly, & Porter, 2000).

Descriptive Statistics

Descriptive statistics for all product and process metrics can be found in Table 2. The highest number of commits (209) can be found in the Barracuda project. This file is a change history in Java format containing only comments. The file with the second highest number of commits (188) is also the class with the highest value of LOCs added (19,252), LOCs deleted (11,706), and the largest file overall (7,546 LOCs). This file is one of the most important classes of the Expresso framework (DBObject.Java) and is responsible for DB communication. The class that

Table 2. Descriptive statistics for all classes

Process Metrics							
	N	**Min**	**Max**	**Mean**	**s.d.**	**75% Percentile**	**Median**
Authors	6,235	1.00	15.00	2.66	1.59	3.00	2.00
Commits	6,235	1.00	209.00	7.24	9.96	8.00	5.00
Days	6,235	0.00	1,628.91	357.44	298.90	459.81	350.78
GINI	6,235	0.00	1.00	0.78	0.24	0.98	0.85
OSSPM	6,235	1.00	58.00	4.80	4.02	6.00	4.00
Product Metrics							
	N	**Min**	**Max**	**Mean**	**s.d.**	**75% Percentile**	**Median**
LOC	6,235	0.00	7,546.00	207.99	279.44	237.00	124.00
DIT	5,339	1.00	10.00	2.60	1.58	3.00	2.00
NCL	5,339	1.00	51.00	1.16	1.24	1.00	1.00
NOF	5,339	0.00	119.00	2.50	4.67	3.00	1.00
NOI	915	1.00	28.00	1.07	1.19	1.00	0.00
NOM	5,339	0.00	252.00	8.37	12.32	10.00	4.00
NORM	5,339	0.00	65.00	0.61	1.89	1.00	0.00
NOC	5,339	0.00	185.00	1.18	7.05	0.00	0.00
NSF	5,339	0.00	69.00	1.54	4.30	1.00	0.00
NSM	5,339	0.00	69.00	0.71	3.03	0.00	0.00
VGavg	5,339	0.00	42.00	2.41	2.60	2.77	1.67
WMC	5,339	0.00	871.00	20.77	37.22	23.00	10.00
LOCm	5,339	0.00	601.00	22.96	35.85	30.00	24.00
VGmax	5,339	0.00	159.00	5.51	8.38	7.00	3.00

is responsible for dispatching the requests of the Struts framework (ActionServlet.Java) is the file with the third highest number of commits (150). An abstract class of the Jetspeed framework that forms the behavior of a portlet has another high value of commits. It is obvious that components providing key functionalities need a special amount of interest because they are usually engaged with several other objects. In accordance with prior studies, all of the process metrics are not 'normal distributed' which can be ascertained using a Kolmogorov-Smirnov test.

In accordance with other studies (Koch, 2004), the number of distinct programmers is quite small with low standard deviation. The histogram of distinct programmers per file shows a heavily skewed distribution. Only 12.2% of the files have more than three distinct authors. Most of the files have one (24.0%) or two (56.1%) programmers, and only 3% have more than five distinct authors. The number of commits per file follows a similar distribution. Only 16.3% have more than 10 commits. Although our values depend on files' respective classes, there are similarities to other studies that have investigated the distribution of distinct authors and commits (Koch, 2004; Krishnamurthy, 2002; Mockus et al., 2002; Ghosh & Prakash, 2000) on the project level.

All of the product metrics are clearly not 'normal distributed' as well. The distribution of LOC is also heavily skewed, which is in accordance with other studies (Koch, 2004; Krishnamurthy, 2002).

Due to the fact that most of the metrics mentioned above measure attributes of classes, we regard real interfaces as missing values (*NOI = 1 and NCL = 0*). Classes that have interior interfaces are valid. Most of the median values are below the threshold suggested by Lorenz and Kidd (1995), as are most of the values for the 75% percentile. The only metric that exceeds this recommendation is the 75% percentile of the size of a method (30>24). The median of the average method complexity per class *VGavg* (1.67) and of the maximum method complexity (3) are below the threshold of 10 suggested by McCabe (1976), and only 11.5% of the classes have a maximum method's complexity greater than 10. Most of the studies which investigate object-oriented metrics used C++ source files (Briand et al., 2000; Chidamber & Kemerer, 1994), so our results cannot directly be compared to them. We are aware of only one study that investigates Java classes (Subramanyam & Krishnan, 2003). Compared to that study we have higher WMC values and our classes are more deeply nested in the inheritance hierarchy. One possible reason for this may be the fact that we examined frameworks that provide abstract classes that are meant to be overridden. The percentage of classes that exceed our dichotomous variables are 5.3% (*CINH*), 6.0% (*MCOMP*), 6.1% (*CDESIGN*), 14.9% (*CSIZE*), and 34.9% (*MSIZE*). The fact that one-third of the classes investigated do not meet the requirements for small method size gives rise to the question whether these threshold values are suitable for object-oriented Java programs. Our data set consists of frameworks that provide functionality for a lot of different scopes. Therefore the average and maximum values may be greater than in normal applications. However we do not adjust the threshold value as it is an indicator for easy understanding and maintenance. The remaining values for the dichotomous variables seem to be reasonable.

RESULTS

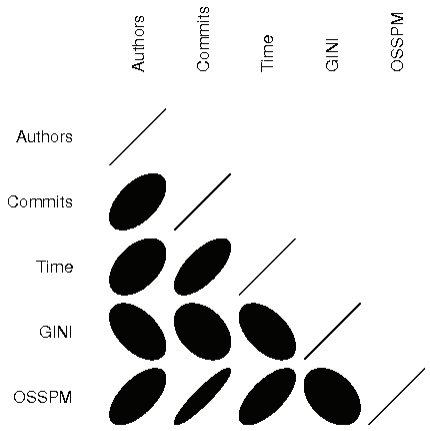
In this analysis, we explore relationships between the metrics mentioned above. Results for correlations between the different process metrics can be found in Table 3, respectively Figure 2, using ellipses (Murdoch & Chow, 1996). Due to the fact that all metrics are not 'normal distributed', we used the nonparametric Spearman coefficient.

The correlation analysis shows expected relationships, like the older a file the more distinct

Table 3. Correlation between process metrics (Spearman coefficient, all at a significance level of p<0.01)

	Authors	**Commits**	**Days**	**GINI**	**OSSPM**
Authors	1.000				
Commits	0.554	1.000			
Days	0.471	0.685	1.000		
GINI	-0.524	-0.370	-0.528	1.000	
OSSPM	0.639	0.925	0.689	-0.393	1.000

Figure 2. Correlation between process metrics (Spearman coefficient, black showing significance level of p<0.01)



programmers are involved (0.471), the more commits are conducted (0.685), and the more work is contributed (0.689). The amount of work (OSSPM) is highly correlated to authors, commits, and the active time, what is indeed reasonable. More interesting are the relations between the inequality as measured by the GINI coefficients and the remaining process metrics. The results show that the older a file, the more homogeneous is the distribution of the added input. The negative correlation between authors and GINI reveals the same tendencies. The more people are involved, the more the work is equally distributed among the participating authors. The number of commits only has slight influence on the GINI coefficient. The correlation between product metrics is not

*Table 4. Correlation between selected process and product metrics (Spearman coefficient, * p < 0.05, ** p < 0.01)*

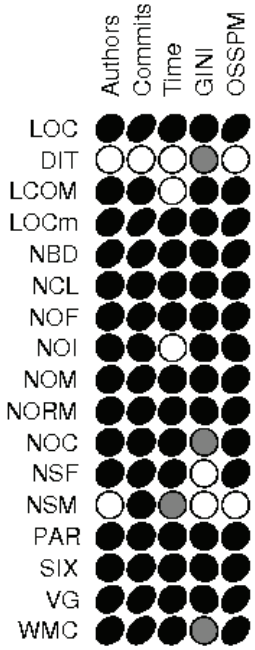
	Authors	Commits	Time	GINI	OSSPM
LOC	**0.157	**0.379	**0.179	**0.057	**0.370
DIT	0.015	0.019	0.025	*−0.027	0.021
LCOM	**0.109	**0.102	0.020	**0.044	**0.137
LOCm	**0.237	**0.432	**0.181	**−0.058	**0.408
NBD	**0.273	**0.290	**0.138	**−0.152	**0.292
NCL	**0.092	**0.139	**0.042	**0.048	**0.131
NOF	**0.149	**0.199	**0.071	**0.038	**0.199
NOI	**−0.080	**−0.097	0.001	**−0.034	**−0.119
NOM	**0.103	**0.253	**0.066	**0.037	**0.232
NORM	**0.095	**0.169	**0.108	**−0.078	**0.181
NOC	**0.085	**0.084	**0.091	*−0.029	**0.093
NSF	**0.129	**0.244	**0.155	−0.022	**0.235
NSM	−0.019	**0.044	*0.027	0.006	0.018
SIX	**0.076	**0.149	**0.108	**−0.086	**0.162
VGavg	**0.242	**0.332	**0.168	**−0.112	**0.337
WMC	**0.214	**0.389	**0.163	*−0.032	**0.366

that important, but it should be mentioned that metrics that measure size attributes of a class (NOM, NSM, NOF, and NSF) are positively correlated to the total size in LOC. Furthermore there is a very strong correlation of WMC to LOC (0.734), which is almost identical to the correlation coefficient of 0.741 found by Subramanyam and Krishnan (2003). More importantly, correlations between product and process metrics have been explored, and the results are shown in Table 4, respectively Figure 3, using ellipses (Murdoch & Chow, 1996).

The complexity metrics *WMC* and *VGavg* have a slight correlation to the number of authors and commits as well as to the effort indicator OSSPM. A similar slight relationship appears regarding the group of metrics that measure the size of a class like *LOC*, *LOCm*, or *NOM*. The influence of the active time on the product metrics can be disregarded. Metrics concerned with the use of inheritance (DIT and NOC) do not seem to be correlated to any of the process attributes. As DIT and NOC are important indicators of reuse and well-structured programming, a deeper look into source code is necessary to gather that kind of information. The GINI coefficient does not seem to be correlated to any product metric.

As described above we created dichotomous variables that indicate whether a class exceeds a certain quality threshold or not and compared these two samples with a non-parametric rank-sum test, the Mann-Whitney-U test, also known as Wilcoxon rank-sum test, for example also applied by Koru and Tian (2005). The test assesses whether

Figure 3. Correlation between selected process and product metrics (Spearman coefficient, grey $p < 0.05$, black $p < 0.01$)



the degree of overlap between the two observed distributions is less than would be expected by chance. The resulting hypotheses are:

H0: *There is no difference in process characteristics between the group S1 that exceeds the threshold values and the group S0 that does not.*

HA: *There is a difference between these groups.*

If H0 is rejected, an additional, one-sided Mann-Whitney U-test is used with the hypotheses:

HA1: *The rank-sum in S1 is greater than in S0, indicating that high values of process metrics foster bad quality.*

HA0: *The rank-sum in S1 is lesser than in S0, indicating that high values of process metrics foster good quality.*

The results of these tests are shown in Table 5. Except for the combinations *MCOMP*/*GINI* and *CSIZE*/*GINI*, the significance is smaller than 0.05, so in these cases we can accept the alternative hypothesis *HA* that the corresponding process metrics have an influence on the product metric. In this case we performed a one-sided Mann-

Table 5. Results of Mann-Whitney U-tests (↑ indicates that high values of the process metrics foster bad quality and ↓ indicates good quality)

	Authors	**Commits**	**Time**	**GINI**	**OSSPM**
MSIZE					
relationship (HA)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)
direction	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↓, *HA0* (p<0.01)	↑, *HA1* (p<0.01)
MCOMP					
relationship (HA)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	rejected	accepted (p<0.01)
direction	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)		↑, *HA1* (p<0.01)
CSIZE					
relationship (HA)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	rejected	accepted (p<0.01)
direction	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)		↑, *HA1* (p<0.01)
CINH					
relationship (HA)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)
direction	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↓, *HA0* (p<0.01)	↑, *HA1* (p<0.01)
DESIGN					
relationship (HA)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)	accepted (p<0.01)
direction	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)	↑, *HA1* (p<0.01)

Whitney U-test to determine the direction of relationship—that is, whether the process metrics have a positive (accept HA1) or negative (accept HA0) influence on the product metrics.

In case of a positive relationship (↑), the sum of the ranks in the group that exceeds the limit is higher than in the group that does not. In quality terms, these results indicate that the higher the process metric is, the lower the quality is. Therefore a higher number of distinct programmers, commits, time, and invested effort have a negative influence on the quality.

To validate our results we performed the same tests only with classes that have at least five different authors (n=668). The results are mainly the same, but we could not reject H0 for the combinations *MCOMP*/Time, *CSIZE*/Authors, and *CSIZE*/Time. The change in the number of authors had an influence on the relationship between GINI and the dichotome product metrics. All combinations had a positive influence (accept HA1 with $p<0.01$), which confirms our prior results that the more the work is concentrated, the worse is the quality of software. Or the other way around, an equal distribution of commits fosters good quality. We will discuss this important finding in more detail later on.

ANALYSIS ON PROJECT LEVEL

For an analysis on project level, we aggregated the product metrics from class level and calculated the process metrics for the whole project, based on those files that were examined in the former section. We stored these results in another table in the database.

Descriptive Statistics

Cocoon is the project with the highest number of distinct programmers, commits, and Java classes. The project ActionServlet, Jpublish, and Echo only have one author. Whether these should be included in further analysis can be discussed. Using a definition of OSS based on the respective license, these projects constitute open source projects, but they conflict with the development model normally associated. On the other hand, these projects might possibly have more participants but a very central control regarding the source code, such that any change must be reviewed and committed by the single maintainer, although other people actually write the code and submit it to this person. We have already discussed this problem with the respective metric description. In the following, we base the analysis on both the full set and a subset with these projects removed.

Struts is the framework with the lowest GINI coefficient, which is an indicator for equality of input. Cocoon, Jetspeed, Struts, Tapestry, and Turbine are projects that are hosted by the Apache Software Foundation. The great popularity of the Apache Web server may explain the encourage-

Table 6. Process metrics for all projects (ordered by number of authors)

	Authors	Commits	Days	GINI	Files	OSSPM
cocoon-2.1	40.00	10,131.00	439.49	0.85	2,298.00	244.00
jakarta-jetspeed	17.00	4,962.00	1,637.92	0.68	677.00	160.00
jakarta-turbine-2	17.00	2,621.00	748.17	0.81	388.00	83.00
jakarta-struts	16.00	3,092.00	1,122.33	0.60	496.00	146.00
expresso	10.00	6,389.00	761.08	0.84	649.00	94.00
jakarta-tapestry	9.00	3,001.00	409.62	0.85	535.00	53.00
Barracuda	9.00	3,543.00	1,279.08	0.75	453.00	71.00
japple	7.00	1,612.00	450.10	0.68	238.00	56.00
maverick	6.00	358.00	1,137.92	0.71	78.00	27.00
ActionServlet	1.00	199.00	223.15	1.00	106.00	4.00
echo	1.00	1,690.00	894.92	1.00	220.00	27.00
jpublish	1.00	886.00	1,172.03	1.00	97.00	34.00

ment of these frameworks. The very low number of commits for the ActionServlet is an indicator for inactivity of the project.

Table 7 shows the mean values of the most important product metrics. The Japple framework has the largest files and the highest WMC. As we have discussed in the previous chapter, there is a very strong linear relationship between LOC and WMC. Therefore this combination is not astonishing. Struts and Jetspeed are the projects with the highest DIT, which indicates extensive usage of subclassing, a form of reuse. The number of children differs across the projects. The frameworks with the lowest average number of children only have one author (ActionServlet, Jpublish).

Table 7. Product metrics for projects (mean values)

	DIT	NORM	NOC	SIX	VGavg	WMC
cocoon-2.1	2.54	0.50	1.13	0.24	2.30	18.02
jetspeed	2.97	0.81	0.81	0.45	2.63	22.41
turbine	2.53	0.42	1.40	0.26	1.82	16.33
struts	3.39	0.69	0.72	0.53	2.97	23.01
expresso	2.82	0.78	1.07	0.49	2.59	30.22
tapestry	2.28	0.29	0.92	0.18	1.65	13.49
Barracuda	2.44	1.29	2.88	0.27	2.83	26.74
japple	1.97	0.77	1.22	0.13	3.16	32.60
maverick	2.40	0.29	1.27	0.33	1.86	9.62
ActionServlet	1.87	0.23	0.33	0.12	2.73	17.01
echo	1.92	0.50	1.75	0.17	2.23	18.44
jpublish	1.97	0.24	0.46	0.10	1.73	13.70

Table 8. Percentage of classes that exceeds the limits of quality metrics

	MSIZE	MCOMP	CSIZE	CINH	CDESIGN
cocoon-2.1	24.06	4.05	11.27	4.35	7.57
jetspeed	37.08	6.06	15.36	8.71	2.81
turbine-2	26.55	1.55	12.37	4.12	2.32
jakarta-struts	34.68	8.67	17.34	3.63	1.61
expresso	45.30	8.01	15.25	4.47	2.77
tapestry	17.38	0.75	5.05	1.87	3.36
Barracuda	33.55	9.71	18.32	6.84	17.44
japple	53.78	7.98	13.03	1.26	3.36
maverick	17.95	0.00	10.26	14.10	11.54
ActionServlet	23.58	4.72	3.77	0.94	4.72
echo	28.18	5.45	22.27	3.18	15.00
jpublish	19.59	2.06	6.19	2.06	1.03

To get an indication of quality and design, we again apply the dichotomous variables used for capturing different possible problem areas (*MSIZE, MCOMP, CSIZE, CINH,* and *DESIGN*). As the total number of classes that exceed our limits is not appropriate due to different numbers of classes between projects, we calculated the relative amount of faulty classes within a project (see Table 8). Metric *MSIZE* depicting problems with method size has rather high values for all projects, but more than 35% of the classes of Japple, Expresso, and Jetspeed exceed the limit. These three frameworks also have a large amount of methods that outrun the upper bound for complexity. The relative amount of misuse of inheritance *CINH* is small except for the Maverick framework (14.1%). The number of classes with interior classes or interfaces is small except for Barracuda, Maverick, and Echo.

Results

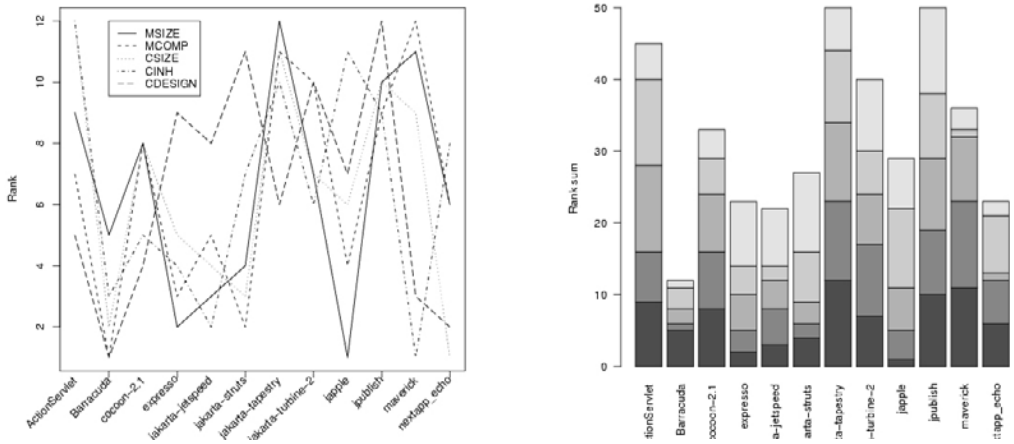
Due to the fact that only a small data set on project level is available, the usage of correlation analysis is not sufficient as the small number precludes any statistically significant findings. Therefore we performed a simple ranking based on the relative amount of classes that exceed our threshold values. High relative amounts of 'faulty' classes result in high ranks (i.e., the project with the highest percentage of classes violating the threshold is ranked on the first place in this variable), and therefore the higher the sum of ranks the higher the overall quality. We do not weight the quality indicators. This ranking can be used to choose the best alternative among concurrent projects depending on their software quality. This ranking is on an ordinal scale and therefore should not be misused to perform any kind of quantitative comparisons, but we try to find some indicators for our findings on class level.

It is interesting that the two projects with only one author have the highest rank overall. Jpublish and ActionServlet also have very low numbers of Java-files and commits, and the OSS development effort is rather low as well. In contrast to these one-man-projects, Tapestry has nine distinct authors but the same Ranksum as Jpublish. But this project is not that old and the invested development

Table 9. Ranks and sum (ordered by decreasing ranksum)

	MSIZE	MCOMP	CSIZE	CINH	CDESIGN	Ranksum
jakarta-tapestry	12	11	11	10	6	50
jpublish	10	9	10	9	12	50
ActionServlet	9	7	12	12	5	45
jakarta-turbine-2	7	10	7	6	10	40
maverick	11	12	9	1	3	36
cocoon-2.1	8	8	8	5	4	33
japple	1	4	6	11	7	29
jakarta-struts	4	2	3	7	11	27
expresso	2	3	5	4	9	23
nextappecho	6	6	1	8	2	23
jakarta-jetspeed	3	5	4	2	8	22
Barracuda	5	1	2	3	1	12

Figure 4. Ranking of projects based on the dichotomous variables (high relative amounts result in low ranks, the higher the sum of the ranks the better the quality)



(a) ranks *(b) sum*

effort is rather small. This can be seen as another proof for the hypothesis that over project lifetime, the quality decreases due to a missing necessary redesign of the software structure.

The largest project overall with 40 distinct authors, more than 10,000, commits and 244 OSSPM is Cocoon. Cocoon is ranked in sixth place—right in the middle—so that we cannot state the quality as extremely bad or good. The second largest project measured by OSSPM and .java-files is Jetspeed, which has the second worst quality ranking, which supports the findings on class level.

In order to statistically underline these results, we used the order produced by the ranksum to compare those projects ranking highly overall to those ranking very low. This was done by a set of Mann-Whitney U-tests as applied above. This time, membership in a project was used as a dividing factor for the classes, and the distribution of relevant process metrics was tested to uncover whether the top projects consistently have different distributions than the lower rated ones. We tested each of the top three projects against each of the bottom three projects, resulting in nine comparisons per process metric. For validation, we also eliminated the one-person projects within the top group, using the next lower ones with more participants. The results indicate that projects in the high-quality region have more authors and commits, but consistently lower GINI coefficient representing more equal distributions (in six, respectively seven, out of nine comparisons in the validation sample). While the first result seems in contradiction with the results on class level, the effects of a high concentration are valid on both levels. These results will be discussed in the following section.

MULTILEVEL ANALYSIS

Multilevel models (also sometimes termed nested or mixed-effect models) are statistical models with parameters arranged in a hierarchical structure (Goldstein, 1999; Snijders & Bosker, 2003; Kreft & de Leeuw, 2002). They are appropriate for data which involves multiple levels, for example on individual level and group level. A classical example is a study of students from different schools, attributes of which might have an impact on individual performance, or research in organizational science (Klein, Tosi, & Cannella, 1999). Multilevel models can account for direct effects of variables on each other within any one level, and also cross-level interaction effects between variables located at different levels.

In our study, we have data within two distinct levels: class and project, with classes being grouped into projects. Therefore, it is possible that aspects of a project like different processes or practices have an influence on the quality of a class. Using a multilevel model, these effects can be accounted for and tested. In the following, we use Akaike's information criterion (AIC) to compare the goodness of fit of the estimated models, which incorporates the number of parameters in selecting the best model, thus penalizing overfitting. For all analysis, we employed R, a freely available language and environment for statistical computing, using the nlme package for multilevel modeling.

First, we computed for comparison classical linear models without hierarchical effects for each dichotomous quality metric (*MSIZE, MCOMP,* etc.), using the independent factors Authors, Commits, Time, and GINI. The results are congruent with the class-level analysis and show the same general trend of negative effects on quality: In general, all of the parameters are significant, positive, and introducing them in a stepwise linear regression increases model quality significantly (all at $p < 0.01$). The following exceptions apply: Time has generally a positive effect on quality (except for *CINH* where the effect is negative, and for *CDESIGN* where it is not significant), and for *MSIZE* the GINI coefficient has a positive influence as well (again congruent with the prior analysis). The GINI coefficient also does not have a significant effect on *CINH*. Overall, the resulting models only account for a relatively small part in overall variation, as the R-squared value ranges from about 0.05 to 0.10.

Following from this, we expand the analysis into multilevel models. We therefore both introduce additional fixed variables from the project level (i.e., the overall effort OSSPM of a project); the total number of programmers, files, and commits; total lifetime and GINI coefficient; and define an increasing range of first-level variables as random. This implies that for each unit, a different slope and intercept is estimated, so that the effect of these can differ between units. These different setups resulted in more than 10 different models being estimated for each quality indicator. Using statistical tests based on AIC, these models were compared with each other and also with the linear models without hierarchical effects computed before.

The first result is that the inclusion of project attributes like total number of programmers does not increase model quality. In all cases, these parameters are not significant in the regression. In the model comparison, introducing these terms does therefore lead to a significant reduction in model fit measured by AIC (except for introducing the project's GINI coefficient, where the reduction is not significant) due to the penalty associated with a higher number of parameters. In comparison to the linear models without hierarchical effects, the results are generally slightly better if no or a small number of project attributes are included, due to the random slope introduced. This underlines that differences between the projects are significant. If the models which define more variables like authors as random (i.e., these are allowed to have a

different intercept and slope depending on project) are inspected, the model quality does in all cases increase significantly. This is, with a few exceptions, true for an increasing number of variables becoming random, even though more parameters are penalized by AIC. The exceptions are: the GINI coefficient for both *MCOMP* and *CINH* does not exhibit significant random effects. This again shows that the differences between projects are manifold and encompass the effects of several attributes like concentration or number of developers. If the random effects estimated are evaluated further, we find that there are even differences in effect direction between projects: for example, the number of authors has a positive effect on method size in six projects, a negative effect in the others. For problems in inheritance structure on the other hand, the number of authors almost uniformly shows a negative effect throughout the projects. Also the concentration has negative effects almost throughout the project set.

From this analysis, we can draw the conclusion that the results achieved by other means hold mostly valid, but that the multilevel approach shows additional insights. We found that there are indeed differences between the projects in the way that the different process metrics have a relationship with product quality concepts, which can be accounted for with this analysis. We also found that the mechanisms and attributes of projects mitigating these effects do not currently seem to be captured by the measurements performed, as the metrics like total number of developers of projects did not show a significant impact. The reasons for the different effects might therefore lie in other attributes like process design which need to be incorporated in future analyses and models.

DISCUSSION

The analyses on class and project level showed several results which need to be discussed in their reasons and in their implications. As shown, a high number of programmers and commits, as well as a high concentration, is associated with problems in quality on class level, mostly to violations of size and design guidelines. This underlines the results of Koru and Tian (2005), who have found that modules with many changes rate quite high on structural measures like size or inheritance. On project level, there is a distinct difference: those projects with high overall quality ranking have more authors and commits, but a smaller concentration than those ranking poorly. We will first address the effects associated with high concentration on few heads, which turn out on both levels, afterwards touching on the differences found.

A high concentration is often seen as a trademark of open source software development and has turned up in almost any study of open source projects (e.g., Koch, 2004; Ghosh & Prakash, 2000; Dinh-Tong & Bieman, 2005). Mockus et al. (2002) have shown this difference to commercial projects in a comparison. Reasons for this concentration are manifold: they reach from motivational aspects like status games which lead to different invested effort between participants, hugely different skills sets of participants in combination with self-selection for tasks, the founding process by one or a few people, to possible delays in achieving commiter status in some projects. On the other hand, we find that a high concentration is correlated with possible problems in the product quality and maintainability. It has to be noted that the direction of this relationship between design aspects and development organization is not determined: If the architecture is not modular enough, a high concentration might show up as a result of this, as it can preclude more diverse participation. The other explanation is that classes that are programmed and/or maintained by a small core team are more complex due to the fact that these programmers ‘know’ their own code and do not see the need for splitting large and complex methods. One possibility in this case is a refactoring

(Fowler, 1999) for a more modular architecture with smaller classes and more pronounced use of inheritance. This would increase the possible participation, thus maybe in turn leading to lower concentration and maintainability, together with other quality aspects. At the beginning of the development process, a core developer team sets up the design which is not adjusted to cope with the increasing number of classes and complexity. In this case it might be better to split huge classes into several subclasses, which may also improve the quality of inheritance and abstraction.

Underlining these results, MacCormack, Rusnak, and Baldwin (2006) have in a similar study used design structure matrices to study the difference between open source and proprietary developed software, without further discrimination in development practices. They find significant differences between Linux, which is more modular, and the first version of Mozilla. The evolution of Mozilla then shows purposeful redesign aiming for a more modular architecture, which resulted in modularity even higher than Linux. They conclude that a product's design mirrors the organization developing it, in that a product developed by a distributed team such as Linux was more modular compared to Mozilla developed by a collocated team. Alternatively, the design also reflects purposeful choices made by the developers based on contextual challenges, in that Mozilla was successfully redesigned for higher modularity at a later stage.

Regarding the number of authors, the results need to be explored further and put into context of the findings on concentration: We found on class level a negative impact, while on project level a positive effect. This underlines a central statement of open source software development on a general level, that as many people as possible should be attracted to a project. On the other hand, these resources should, from the viewpoint of product quality, be organized in small teams. Ideally, on both levels, the effort is not concentrated on too few of the relevant participants. This is certainly not contrary to conventional software engineering knowledge, which can be found to hold in this context as well.

The implications of these findings need to be discussed in two different contexts, the first one being within open source projects, and also in general. These two settings differ significantly, most relevantly in the general aims, the possibilities for intervention by project management, and also the motivation of participants. In an open source project, a management in classical form does not exist, although often a maintainer, inner circle, or other authority (although with mostly minimal impact) could be interested in the organization of work within the project. Also the aims of a project, and interwoven with this, the motivations of participants are very much different from commercial settings, and they need to be considered. Therefore there are very limited possibilities for any central agency to manage and steer the participants, or they might lose motivation and leave the project. On the other hand, management responsibilities are often taken up by the founding group of a project. In case of early phases of a project, the design should therefore strive to allow for these teams to form by providing an appropriate number of classes within a modular architecture, termed by MacCormack et al. (2006) as "architecture of participation." Executing a refactoring within the context of a large and well-established open source project often might prove difficult, but a central agency should carefully monitor the respective metrics as described in this article to gain an understanding of possible future problems, both in quality and participation aspects. If those are identified, soft measures might be applied to encourage the participants to adjust, for example by using increased reputation and recognition for people participating in such efforts. In addition, the lack of formal design specification often associated with open source projects should be overcome. Again, taking up these tasks should be rewarded within the reputation structure, while other possible motivational factors like training

are naturally offered in this context. MacCormack et al. (2006) have shown with the Mozilla case that such efforts can be successful. In our study, we have found evidence for a refactoring having taken place in the Maverick project based on log messages, which is now top ranking in method size and complexity measures.

In a commercial context, many of the problems as discussed above do not apply, so management has more possibilities to enforce a certain organization of work or a necessary refactoring. The organizational form of 'chief programmer team organization' (Mills, 1971; Baker, 1972), also termed 'surgical team' by Brooks (1995), has system development divided into tasks each handled by a chief programmer who is responsible for the most part of the actual design and coding, supported by a larger number of other specialists like a documentation writer or a tester. A similar form of development seems to be adopted by open source projects, although a too highly concentrated form does not perform well given the negative effects associated with high concentration. Possibly the single-author projects in our sample form an example of this organization. Only one person has access to the source code and is assisted by a larger number of other participants.

In arriving at the results of this study, we found that the creation of dichotomous variables helped in several ways, although the thresholds remain a problematic point. The huge number of available and sometimes highly correlated product metrics can be aggregated into a more manageable and interpretable set in this way, and effects on quality can more easily be analyzed. For the process metrics applied, we found that different calculation approaches for the GINI coefficient did not change the results in a significant way. The effort indicator OSSPM introduced did not give much additional information as well, although the high correlation to other metrics like commits need not be present in all data sets. We propose that the invested effort might still be considered as an important factor.

CONCLUSION

The analysis described in this article has tried to enhance prior studies on OSS by providing an empirical validation of relationships between process attributes and product quality. We presented and applied a method to calculate and merge both metrics, addressing both dimensions from online versioning repositories. In this article we have focused on the investigation of frameworks for the development of Web-based applications, which therefore offer similar functionalities and are suitable for a comparison. The results clearly show that it is possible to gather the necessary information to find relationships between process and product metrics. Using mostly object-oriented product metrics focusing on quality by employing a subset of the well-known Chidamber and Kemerer (1994) metrics, complemented with several metrics proposed by Lorenz and Kidd (1995) and several process metrics including total number of commits and the number of distinct programmers as well as the GINI coefficient as a measure of inequality within the developer group, we found that indeed significant relationships exist. This underlines the results of MacCormack et al. (2006). We identify the number of commits, the number of distinct programmers, and the active time as factors of influence which have a negative effect on quality. In particular, complexity and size are negatively influenced by these process metrics. Furthermore a high concentration of added work fosters bad quality. In discussing reasons for this finding, one explanation for this relationship might be found in a missing necessary refactoring of the design. We have also discussed the reasons for this and implications for practice.

Limitations of this work can certainly be found in the thresholds applied for defining methods as faulty based on experiences with C++ projects. Using preliminary sensitivity analysis, we have explored the impact of small changes of up to 20% on the threshold values and found that the main results presented here are still valid. Nevertheless,

more work should be invested in this area to arrive at sensible thresholds, especially for Java and related programming languages. Another issue to be further explored in later studies are effects on different levels: we have tried to account for project-level influences on classes using a multilevel modeling approach, but the fact that some classes might be matched pairs across projects, while others are not, might still pose a problem. We have also found that differences between the projects in the effects of process metrics exist, but the attributes mitigating these still remain to be explored. Although we have tried to achieve a relatively homogeneous set of projects, differences in functionality and other aspects persist. Naturally, larger data samples would also be of high interest, especially a comparison of OSS projects with commercial software development, which might more prominently show differences in the development process. Furthermore, a longitudinal study of both product and process metrics over the lifetime and evolution of a project might provide more insights, as well as exploring the influence of process metrics on maintainability, which has been investigated in some studies (Deligiannis, Shepperd, Roumeliotis, & Stamelos, 2003; Fioravanti & Nesi, 2001; Samoladas, Stamelos, Angelis, & Oikonomou, 2004). Our study only gives qualitative evidence of maintainability.

Overall, we think that this study provides a first step despite these limitations. We have provided evidence regarding relationships between process and product measures in open source software development, and pointed out several characteristics tending to lead to lower product quality. This serves as a starting point for devising strategies to effectively manage projects for achieving higher quality and maintainability. Additional research can also benefit from observations regarding the method applied in this study, and might yield even more insights, leading to improvements in OSS and other software development processes.

REFERENCES

Atkins, D., Ball, T., Graves, T., & Mockus, A. (1999). Using version control data to evaluate the impact of software tools. *Proceedings of the 21st International Conference on Software Engineering* (pp. 324–333). Los Angeles: ACM Press.

Baker, F.T. (1972). Chief programmer team management of production programming. *IBM Systems Journal, 11*(1), 56–73.

Basili, V.R., Briand, L.C., & Melo, W.L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering, 22*(10), 751–761.

Basset, T. (2004). Coordination and social structures in an open source project: Videolan. In S. Koch (Ed.), *Open source software development* (pp. 125-151). Hershey, PA: Idea Group.

Bollinger, T., Nelson, R., Self, K.M., & Turnbull, S.J. (1999). Open-source methods: Peering through the clutter. *IEEE Software, 16*(4), 8–11.

Briand, L., Wüst, J., Ikonomovski, S., & Lounis, H. (1998). *A comprehensive investigation of quality factors in object-oriented designs: An industrial case study.* Technical Report ISERN-98-29, International Software Engineering Network.

Briand, L.C., Wüst, J., Daly, J.W., & Porter, D.V. (2000). Exploring the relationship between design measures and software quality in object-oriented systems. *Journal of Systems and Software, 51*(3), 245–273.

Brooks, F.P. Jr. (1995). *The mythical man-month: Essays on Software engineering* (anniv. ed.). Reading, MA: Addison-Wesley.

Chidamber, S., & Kemerer, C.F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493.

Chidamber, S.R., Darcy, D.P., & Kemerer, C.F. (1998). Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions on Software Engineering, 24*(8), 629–639.

Chidamber, S.R., & Kemerer, C.F. (1991). Towards a metric suite for object oriented design. *Proceedings of the 6th ACM Conference of Object Oriented Programming, Systems, Languages and Applications* (pp. 197–211). Phoenix, AZ: ACM Press.

Coleman, E.G., & Hill, B. (2004). The social production of ethics in debian and free software communities: Anthropological lessons for vocational ethics. In S. Koch (Ed.), *Open source software development* (pp. 273–295). Hershey, PA: Idea Group.

Conte, S.D., Dunsmore, H., & Shen, V. (1986). *Software engineering metrics and models.* Menlo Park, CA: Benjamin/Cummings.

Cook, J.E., Votta, L.G., & Wolf, A.L. (1998). Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering, 24*(8), 650–663.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings—Software Engineering, 149*(1), 3–17.

Cusumano, M.A. (2004). Reflections on free and open software. *Communications of the ACM, 47*(10), 25–27.

Deligiannis, I., Shepperd, M., Roumeliotis, M., & Stamelos, I. (2003). An empirical investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software, 65*(2), 127–139.

Demetriou, N., Koch, S., & Neumann, G. (2006). The development of the OpenACS community. In M. Lytras & A. Naeve (Eds.), *Open source for knowledge and learning management: Strategies beyond tools* (pp. 298–318). Hershey, PA: Idea Group.

Dempsey, B.J., Weiss, D., Jones, P., & Greenberg, J. (2002). Who is an open source software developer? *Communications of the ACM, 45*(2), 67–72.

Dinh-Tong, T.T., & Bieman, J.M. (2005). The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering, 31*(6), 481–494.

Dutoit, A.H., & Bruegge, B. (1998). Communication metrics for software development. *IEEE Transactions on Software Engineering, 24*(8), 615–628.

Elliott, M.S., & Scacchi, W. (2004). Free software development: Cooperation and conflict in a virtual organizational culture. In S. Koch (Ed.), *Open source software development* (pp. 152–172). Hershey, PA: Idea Group.

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management, 16*(4), 88–99.

Fayad, M.E., & Schmidt, D.C. (1997). Object-oriented application frameworks. *Communications of the ACM, 40*(10), 32–39.

Feller, J., Fitzgerald, B., Hissam, S.A., & Lakhani, K.R. (Eds.). (2005). *Perspectives on free and open source software.* Cambridge, MA: MIT Press.

Fenton, N.E. (1991). *Software metrics—a rigorous approach.* London: Chapman & Hall.

Fioravanti, F., & Nesi, P. (2001). Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Transactions on Software Engineering, 27*(12), 1062–1084.

Fischer, M., Pinzger, M., & Gall, H. (2003). Populating a release history database from version control and bug tracking systems. *Proceedings of the 19th IEEE International Conference on*

Software Maintenance (pp. 23–32), Amsterdam, The Netherlands.

Fogel, K. (1999). *Open source development with CVS*. Scottsdale: CoriolisOpen Press.

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Boston: Addison-Wesley.

Gallivan, M.J. (2001). Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal, 11*(4), 277–304.

German, D. (2006). A study of contributors of PostgreSQL. *Proceedings of the International Workshop on Mining Software Repositories* (MSR'06), Shanghai.

Ghosh, R.A., & Prakash, V.V. (2000). The Orbiten free software survey. *First Monday, 5*(7).

Goldstein, H. (1999). *Multilevel statistical models*. London: Arnold.

Hahsler, M., & Koch, S. (2005). Discussion of a large-scale open source data collection methodology. *Proceedings of the Hawaii International Conference on System Sciences* (HICSS-38), Big Island, HI.

Hansen, M., Köhntopp, K., & Pfitzmann, A. (2002). The open source approach—opportunities and limitations with respect to security and privacy. *Computers & Security, 21*(5), 461–471.

Henderson-Seller, B. (1996). *Object-oriented metrics: Measures of complexity*. Upper Saddle River, NJ: Prentice Hall.

Hertel, G., Niedner, S., & Hermann, S. (2003). Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel. *Research Policy, 32*(7), 1159–1177.

Humphrey, W. (1995). *A discipline for software engineering*. Reading, MA: Addison-Wesley.

Hunt, F., & Johnson, P. (2002). On the pareto distribution of sourceforge projects. *Proceedings of the Open Source Software Development Workshop* (pp. 122–129), Newcastle, UK.

Johnson, R. (1997). Frameworks=(components+ patterns). *Communications of the ACM, 40*(10), 39–42.

Jones, C. (1986). *Programming productivity*. New York: McGraw-Hill.

Kemerer, C.F., & Slaughter, S. (1999). An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering, 25*(4), 493–509.

Klein, K.J., Tosi, H., & Cannella, A.A. Jr. (1999). Multilevel theory building: Benefits, barriers, and new development. *Academy of Management Review, 24*(2), 243–248.

Koch, S. (2004). Profiling an open source project ecology and its programmers. *Electronic Markets, 14*(2), 77–88.

Koch, S. (2004a). Agile principles and open source software development: A theoretical and empirical discussion. *Extreme Programming and Agile Processes in Software Engineering: Proceedings of the 5th International Conference XP 2004* (pp. 85–93). Berlin: Springer-Verlag (LNCS 3092).

Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12*(1), 27–42.

Koru, A.G., & Tian, J. (2004). Defect handling in medium and large open source projects. *IEEE Software, 21*(4), 54–61.

Koru, A.G., & Tian, J. (2005). Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. *IEEE Transactions on Software Engineering, 31*(8), 625–642.

Kreft, I., & de Leeuw, J. (2002). *Introducing multilevel modeling.* London: Sage.

Krishnamurthy, S. (2002). Cave or community? An empirical investigation of 100 mature open source projects. *First Monday, 7*(6).

Long, Y., & Siau, K. (2007). Social network structures in open source software development teams. *Journal of Database Management, 18*(2), 25–40.

Lorenz, M., & Kidd, J. (1995). *Object oriented metrics.* Upper Saddle River, NJ: Prentice Hall.

MacCormack, A., Rusnak, J., & Baldwin, C.Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science, 52*(7), 1015–1030.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320.

McConnell, S. (1999). Open-source methodology: Ready for prime time? *IEEE Software, 16*(4), 6–8.

Merisalo-Rantanen, H., Tuunanen, T., & Rossi, M. (2005). Is extreme programming just old wine in new bottles: A comparison of two cases. *Journal of Database Management, 16*(4)**,** 41–61**.**

Mills, H.D. (1971). *Chief programmer teams: Principles and procedures.* Report FSC 71-5108, IBM Federal Systems Division, USA.

Mockus, A., Fielding, R.T., & Herbsleb, J.D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309–346.

Morisio, M., Romano, D., & Stamelos, I. (2002). Quality, productivity and learning in framework-based development: An exploratory case study. *IEEE Transactions on Software Engineering, 28*(8), 340–357.

Murdoch, D.J., & Chow, E.D. (1996). A graphical display of large correlation matrices. *The American Statistician, 50*(2), 178–180.

Neumann, C. (2002). *Jsp- und Servlet-basierte frameworks für Web-applikationen*. Master's Thesis, Universität Karlsruhe, Germany.

Park, P. (1992). *Software size measurement: A framework for counting source statements.* Technical Report CMU/SEI-92-TR-20, Software Engineering Institute, Carnegie Mellon University, USA.

Payne, C. (2002). On the security of open source software. *Information Systems Journal, 12*(1), 61–78.

Perens, B. (1999). The open source definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 171–188). Cambridge, MA: O'Reilly & Associates.

Prieto-Diaz, R. (1993). Status report: Software reusability. *IEEE Software, 10*(3), 61–66.

Raymond, E.S. (1999). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary.* Sebastopol, CA: O'Reilly & Associates.

Robles-Martinez, G., Gonzalez-Barahona, J.M., Centeno-Gonzalez, J., Matellan-Olivera, V., & Rodero-Merino, L. (2003). Studying the evolution of libre software projects using publicly available data. *Proceedings of the 3rd Workshop on Open Source Software Engineering—25th International Conference on Software Engineering* (pp. 111–115), Portland, OR.

Samoladas, I., Stamelos, I., Angelis, L., & Oikonomou, A. (2004). Open source software development should strive for even greater code maintainability. *Communications of the ACM, 47*(10), 83–87.

Snijders, T.A.B., & Bosker, R.J. (2003). *Multilevel analysis: An introduction to basic and advanced multilevel modeling.* London: Sage.

Stallman, R.M. (2002). *Free software, free society: Selected essays of Richard M. Stallman*. Boston: GNU Press.

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G.L. (2002). Code quality analysis in open source software development. *Information Systems Journal, 12*(1), 43–60.

Subramanian, G., & Corbin, W. (2001). An empirical study of certain object-oriented software metrics. *Journal of Systems and Software, 59*(1), 57–63.

Subramanyam, R., & Krishnan, M.S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering, 29*(4), 297–309.

Thuraisingham, B. (2005). Privacy-preserving data mining: Development and directions. *Journal of Database Management, 16*(1), 75–87.

Turk, D., France. R., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management, 16*(4)**,** 62–87.

Vixie, P. (1999). Software engineering. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution* (pp. 91–100). Cambridge, MA: O'Reilly & Associates.

Witten, B., Landwehr, C., & Caloyannides, M. (2001). Does open source improve system security? *IEEE Software, 18*(5), 57–61.

This work was previously published in the Journal of Database Management, edited by K. Siau, Volume 19, Issue 2, pp. 31-57, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.24
A Graphical User Interface (GUI) Testing Methodology

Zafar Singhera
ZAF Consulting, USA

Ellis Horowitz
University of Southern California, USA

Abad Shah
R & D Center of Computer Science, Pakistan

ABSTRACT

Software testing in general and graphical user interface (GUI) testing in particular is one of the major challenges in the lifecycle of any software system. GUI testing is inherently more difficult than the traditional and command-line interface testing. Some of the factors that make GUI testing different from the traditional software testing and significantly more difficult are: a large number of objects, different look and feel of objects, many parameters associated with each object, progressive disclosure, complex inputs from multiple sources, and graphical outputs. The existing testing techniques for the creation and management of test suites need to be adapted/enhanced for GUIs, and new testing techniques are desired to make the creation and management of test suites more efficient and effective. In this article, a methodology is proposed to create test suites for a GUI. The proposed methodology organizes the testing activity into various levels. The tests created at a particular level can be reused at higher levels. This methodology extends the notion of modularity and reusability to the testing phase. The organization and management of the created test suites resembles closely to the structure of the GUI under test.

INTRODUCTION

Graphical user interfaces (GUl) are an important part of any end-user software application today and can consume significant design, development, and testing activities. As much as half

of the source code of a typical user-interaction intensive application can be related to user interfaces (Harold, Gupta, & Soffa, 1993; Horowitz & Singhera, 1993). GUIs provide an easier way of using various functions of the application by organizing them in a hierarchy of options and presenting only the options which make sense in the current working context. GUIs help users concentrate on the problem instead of putting efforts in remembering all the options provided by the software application that is being used to solve the problem, or searching for the right option from a huge list of options provided by the application. Graphical user interfaces organize the standard user actions and working paradigms into various components that are presented graphically to the user during various usage and application contexts. GUIs enhance the usability of an application significantly. However it also makes application development, testing and maintenance significantly more difficult (Myers, 1993; Wittel & Lewis,1991). The nature of GUI applications, their asynchronous mode of operation, nontraditional input and output, and hierarchical structure for user interaction make their testing significantly different and difficult from the traditional software testing.

Functional and regression testing of graphical user interfaces is significantly more complex than testing of traditional non-GUI applications because of the additional complexities mentioned in the previous paragraph. A number of commercial tools, like Mercury Interactive's WinRunner, XRunner and Segue Software's SilkPerformer, are used in the industry to test graphical user interfaces. These tools provide capture/replay capabilities to test a graphical user interface. Although functionality provided by these tools is sufficient for typical recored/replay scenarios but they lack an underlying model that can provide more information about the test coverage or to determine the quality of the user interface from a particular functional or implementation perspective. These tools also do not provide a framework that assists in organized and modular testing. The methodology presented in this article uses user interface graphs (UIG) as a framework for organization of test scripts, generation of modular test suites, and coverage analysis of a test execution.

In this article, we propose a methodology for regression testing of graphical user interfaces, with and without a formal specification of the application under test. The remainder of this article is organized as follows: Section 2 highlights some of the best practices and recommendations that help in testing a GUI application in an organized fashion, improve efficiency and effectiveness of testing, reduces possibility of errors, and minimizes repeated work. Section 3 describes the major steps of the proposed methodology. It also introduces a sample X application, called Xman, which is used to demonstrate the effectiveness of the suggested strategy. Section 4 demonstrates the testing methodology when formal specifications of the application under test are not available. Section 5 illustrates the proposed testing methodology when the formal specifications of the application under test are available. This section also describes the way statistics are collected during a testing activity and how those can be used to improve the quality of the testing. Section 6 points out the situations when a modification to the application under test might require tuning or recapturing of some of the test scripts. Section 7 concludes the article by summarizing our contribution and providing hints about the future related work.

GUI TESTING: BEST PRACTICES AND RECOMMENDATIONS

In this section, we highlight some of the sought features, well-knows best practices and recommendations for planning a testing activity for a graphical user interface.

- Every element of the GUI should be considered as an object uniquely identifiable by a name. The objects should have a well-defined set of parameters and their response to the outside events should also be well defined.
- The testing activity should be planned carefully around a formal model of the application under test. This model should be powerful enough to provide automatic test generation and coverage analysis.
- Testing of a GUI should be performed in a layered fashion. The list of objects to be tested at a particular level, is either built dynamically while testing at lower levels or from the specifications of the application under test. The list of objects for the lowest level is the basic widget, supported by the underlying toolkit. While testing the highest level, it considers the entire application as a single object. The decision about the number of testing levels and the qualifying criteria for a particular testing level must be made before creating any tests.
- The tests should be organized as a hierarchy of scripts, that is, files containing commands to simulate user actions and verify results. This hierarchy should closely correspond to the object hierarchy of the application under test. Each directory in the hierarchy holds scripts that are related to a particular object and its descendents. The individual scripts should be as small as possible and should test one particular feature of each object. However, if the features are related and simple, then they can be grouped together in the same script.
- Each script should begin with a cleat and precise description of the intended purpose of the script and the state of the application required for its proper execution. A script should be divided into three sections. The first section of the script builds the environment required to test the particular feature of the application, the script is intended for. The second section of the script tests the intended feature of the application being tested by the script. The third section restores the state of the AUT and the operating environment, to a point that existed before entering the script.
- A script should be created in such a way that some or all the sections of the script can be executed by calling the script from another script. It provides reusability feature in testing also.
- Instead of manually capturing or replaying the test scripts, a tool should be used to perform these functions automatically and verify the behavior of the application under test. The tool should be capable of addressing an object in the GUI by its symbolic name, instead of its location, dimensions or any other contextual information.
- The data for the result verification should be captured in terms of object attributes when possible and only those attributes should be captured which are critical to verify the functions of the application, being tested by the current script. If image comparisons are unavoidable, then the images should be captured with reference to the smallest enclosing object and area of the captured images should not be more than absolutely required. The number of verifications should also be kept to an absolute minimum especially when image comparisons are involved.
- The script commands to simulate user actions during the replay and the data for verification of the AUT behavior should be kept separately. This separation is required because the verification data might change depending on the environment while the script commands should be independent of the environment and should be valid across multiple platforms. If script commands and verification data are stored separately, then

it is easier to port a test suite across multiple platforms. In fact, a good tool should automatically perform the porting from one hardware platform to the other.

Proposed Methodology for GUI Testing

This section proposes a methodology for the testing of a graphical user interface (GUI). This proposed methodology is suitable particularly when one has a tool similar to Xtester (Horowitz & Singhera, 1993). It follows the recommendations provided in the previous section. The methodology works in the following two scenarios:

i. Testing without formal specifications/model of the application under test
ii. Testing with a formal model of the application.

Both these scenarios are described in the following subsections.

Testing without a Formal Model

Creation of formal specifications of a GUI application for its testing purposes is a difficult task and requires a significant amount of effort. It is also not feasible to invest resources in creating the formal specifications of the application; hence the testing has to be performed without it. The best thing that can be done in such a situation is to incrementally build a test hierarchy for the application, by capturing-user sessions in an organized way. Automatic test generation or coverage analysis is not possible without formal specifications of the application under test. The major steps of the proposed methodology to test an application without a specification are given below:

Step 1: Initialization

Make basic decisions about the testing activity. Some of the most important decisions, which must be taken at this point, are:

- The number of testing levels and criteria to build list of objects for a particular level.
- Initialize a list of objects for each level that holds the names and some information about the objects of the user interface. The information includes the way the object can be mapped on the screen, the mechanism to unmap it from the screen, and if it has been tested or not.
- The location of the test suite and its organization.
- The application resources that will be used during this testing activity should be listed.

Step 2: Building the Initial Object List

Go through the documentation of the application under test and find all the top-level windows, which might appear as starting windows of the application. These windows and, their related information is added to the list of objects for the first testing level and marked as tested.

Step 3: Building Test Suite

Take the first object from the top of the object list, which has not been tested, and create test scripts for it. The procedure for creating test scripts for a particular object is given in Figure 1. The sub-objects of the object under test are added to the list of objects by scanning the object from left to right and top to bottom. Keep on taking objects from the top of the object list and testing them until all the objects in the list are marked as tested. When the list associated with a particular level has no

Figure 1. Strategy for testing without specifications

```
Display the_ object on the screen and verify its ap-
pearance;
If the object has sub-objects
    Add its immediate sub-objects at the top of the
list;
If the object qualifies for a higher testing level
Add it to the list of objects for the higher level;
Send expected events to the object and verify its
response;
If a new object appears in response to the event
if the new object is not listed as a tested object
    Add it to the end of the list;
```

untested object, start testing objects from the list, associated with the next higher level. This process continues until all the levels are tested.

Step 4: Creating Script Drivers

Write higher level scripts for all the top level windows for any other complex objects, each of the testing levels and for the entire test suite. These scripts will replay all the scripts related to the object and its descendents. The highest-level script driver should replay each and every script in the suite.

Step 5: Testing the Test Suite

Make sure that all the scripts and script drivers work properly and cover all the features of the application, which needs to be tested. One cannot do much automatically to determine the quality of a test suite, in the absence of a formal model of the application under test. After the creation of the test suite, run the highest-level script driver to verify that all the scripts in the suite are capable of properly replaying and trying to match the features covered by the test suite with those included in the test requirements or application documentation

Testing with a Formal Model

The strategy to build test scripts without a formal specification, discussed in the previous subsection, puts a lot of responsibility on the person creating those scripts. The strategy also requires that the application under test should be running reliably before the capturing of script is even started. The scripts created without any formal specification are also vulnerable to any modification in the application, which affects its window hierarchy. It requires that after making any changes to the application under test, the affected scripts should be located manually and recaptured or tuned to offset the modification in the application. It is also

not possible to create test scripts automatically or to get a coverage measure after running a set of test suites. To overcome these drawbacks and get access to advanced features like automatic test generation and coverage analysis, one has to invest some effort to formally specify the application under test. This section provides a methodology to test an application when its formal specification is provided or resources are available to build such a specification. The following are the major steps of the testing methodology when a formal specification of an application under test is available.

Step1: Building the Model

Build a user interface graph of an application under test. When resources permit, the very first step in testing the application should be to build a formal model of the application under test. XTester provides such a formal model, called *user interface graph (UIG).* UIG provides information about the object hierarchy of the application. It also provides information about the nature of a particular object and the effects of an event in an object to the other objects in the user interface. A UIG can be built manually by creating a user interface description language (UIDL) file, or it can be created semi-automatically by steering through the application under test and filling in the missing information about the objects (see Horowitz & Singhera, 1993) for more details on syntax of UIDL and UIG).

Step 2: Initialization

Make basic decisions about the testing activity. Some of the most important decisions, which must be taken at this point, are listed as follows:

- The number of testing levels and qualifying criteria for each testing level.
- The location of the test suite and its organization.
- The application resources that will be used during this testing activity.

Step 3: Build Object Lists

Build a list of objects for each testing level. After building the formal model, it is possible to build the object lists for all testing levels. The procedure for building those lists is to start from the root of the UIG and perform a post-order walk of the object hierarchy. Add each visited node to the object lists associated with the levels, for which it qualifies.

Step 4: Building Test Suite

The strategy for capturing scripts without any formal specification, which has been discussed in the previous section, can also be used for capturing scripts when the application has been specified formally. However, capturing scripts with formal specifications provides us some additional advantages over the scripts which have been captured without any specification. These advantages include an overall picture of the application under test, and hence a more efficient test suite, a test suite which is less affected by the changes in the application, automatic test generation, and coverage analysis.

Step 5: Creating Script Drivers

Write higher level scripts for all the top level windows or any other complex objects, each of the testing levels and for the entire test suite. These scripts will replay all the scripts related to the object and its descendents. A highest-level script driver should replay each and every script in the suite. These scripts can also be created automatically.

Step 6: Coverage Analysis

Once a test suite has been created; it should be replayed in its entirety to determine the coverage provided by the test suite. This coverage should be performed at each level, that is, the coverage criteria for level-I should be the verification of all the objects in the application that have been created, mapped, unmapped and destroyed, at least once and every event expected by an object, has been exercised on it at least once. The criteria for higher levels is to make sure that all the interactions and side effects among objects which make a composite object at the corresponding level, has been verified.

AN INTRODUCTION TO XMAN

In this section, we introduce the application Xman, which is used to demonstrate the effectiveness of the methodology, discussed in the previous section. Xman is a small application which is distributed with the standard *X* release. It provides a graphical user interface to the UNIX *man* utility. It has been developed using the Athena widget set and some of its windows are shown in Figure 1. The following paragraphs briefly describe the functionality of Xman.

When Xman is started, it displays its main window, called Xman, by default. This main window contains three buttons: Help, Manual Page and Quit. Clicking on the manual page button, it displays a window, called manual page. A Help window is displayed when the help button is clicked. The quit button is used to exit from Xman.

The manual page window is organized into various sections. A breathe top of the window contains two menu buttons, options and sections, in the left half and a message area in the right. The rest of the area below the bar, called text area, is used to display the names of the available manual pages in the currently selected section, and the contents of the currently selected manual page. Both the names and contents portions of the text area are resizable and have vertical scrollbars on the left.

The Options menu contains the following entries:

- **Display Directory:** It displays names of manual pages in the entire text area.
- **Display Manual Page:** It displays contents of the currently selected manual page in the entire text area.
- **Help:** It displays a help window.
- **Search:** It displays a dialog box to enter the name of a manual page to search for.
- To show both screens, the area is vertically divided into two halves, with the upper half showing the directory contents of currently selected man page section and the lower half showing the contents of the currently selected manual page. This option toggles to Show One Screen and also disables menu entries Display Directory and Display Manual Page.
- **Remove This Manpage:** It removes the Manual Page window from the screen.
- **Open New Manpage:** It creates another Manual Page window.
- **Show Version:** It displays the current version of Xman in the Message area
- **Quit:** exits from the Xman application.

The Sections menu contains one option for each manual page section available on the system. The standard options are User Commands, System Calls, Subroutines, Devices, File Format, Games, Miscellaneous, and System Administration.

The Help window displays a limited version of the Manual Page window. The window has exactly the same structure as the Manual Page window but the Sections menu button and the first five options in the Opt ions menu are disabled.

It displays man page for Xman itself. No more than one Help window can exist at a time while an arbitrary number of Manual Page windows can be created.

The testing of Xman has been organized in three layers. The first layer verifies the behavior of individual GUI objects. The second layer verifies the inter-object effects among objects belonging to the same top level window. The third layer verifies the inter-object effects among objects belonging to different top level windows. The following sections provide details of this testing activity.

TESTING WITHOUT FORMAL SPECIFICATIONS

This section provides a demonstration for the testing of Xman, when no formal model is available for it. The following subsections demonstrate each step of the methodology, described in Section 2.

Initialization

- **Number of testing levels:** As Xman is a fairly small and simple application, so the testing activity is organized in three (3) levels. The first level tests the individual objects in Xman. The second and third levels verify the interactions and side effects of the objects which belong to the same top level window and different top level windows, respectively.
- **Location of the test suite:** Suppose that the root directory for the test suite being captured, is Xman Test. This directory contains resource file(s) used during testing, and the result files are created in the directory, by default. It also has three subdirectories, that is, *Leve1-1*, *Leve1-2* and *Leve1-3*, one for each testing level.
- **Object list:** A list of objects, called *Obj List*, is initialized. This list contains information about the objects and is initialized to be empty.
- **Application resources:** ~/XmanTest/Xman.Defaults is the file which contains the default application resources of Xman for this testing activity and these resources always remains the same.

Building the Initial Object List

By default, Xman starts with a top *box*, called Main Window in Figure 2. However, a command line option, -notopbox, is available which can be used to bypass the Main Window and display the Manual Page window directly. As Main Window and Manual Page window are the only windows, which can appear when Xman is started, so the initial object list contains only these two objects.

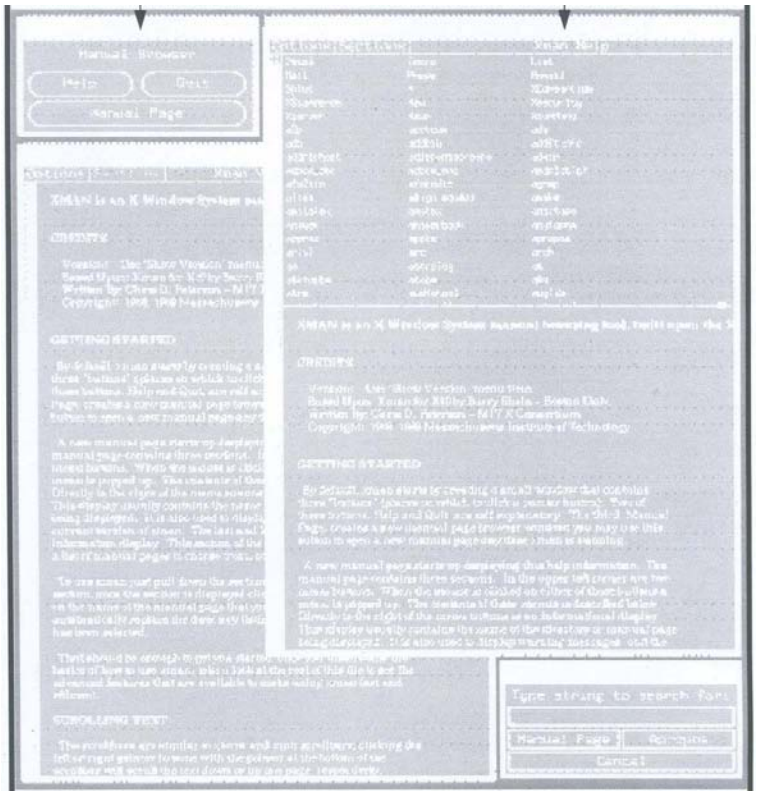
Building the Test Suite

This section provides the details on building all the three testing levels for Xman. To make things simple, we only discuss the creation of test suites related to the Main Window of Xman. We ignore any keyboard accelerators to which Xman responds. The test suites for other windows can be created in a similar way. Figure 3 provides the hierarchy of scripts related to the Main Window for all the three testing levels.

First Level

Suppose that the initial object list is built in such a way so that Xman Main Window is on the top of the list. We select it as the object under test and create a new directory, called ~/Xman Test/Leve1-1/Xman to build test scripts related to the first level testing. The scripts related to XMan Main window object display itself the window on the screen, exercise all the window manager

Figure 2. Main windows of Xman

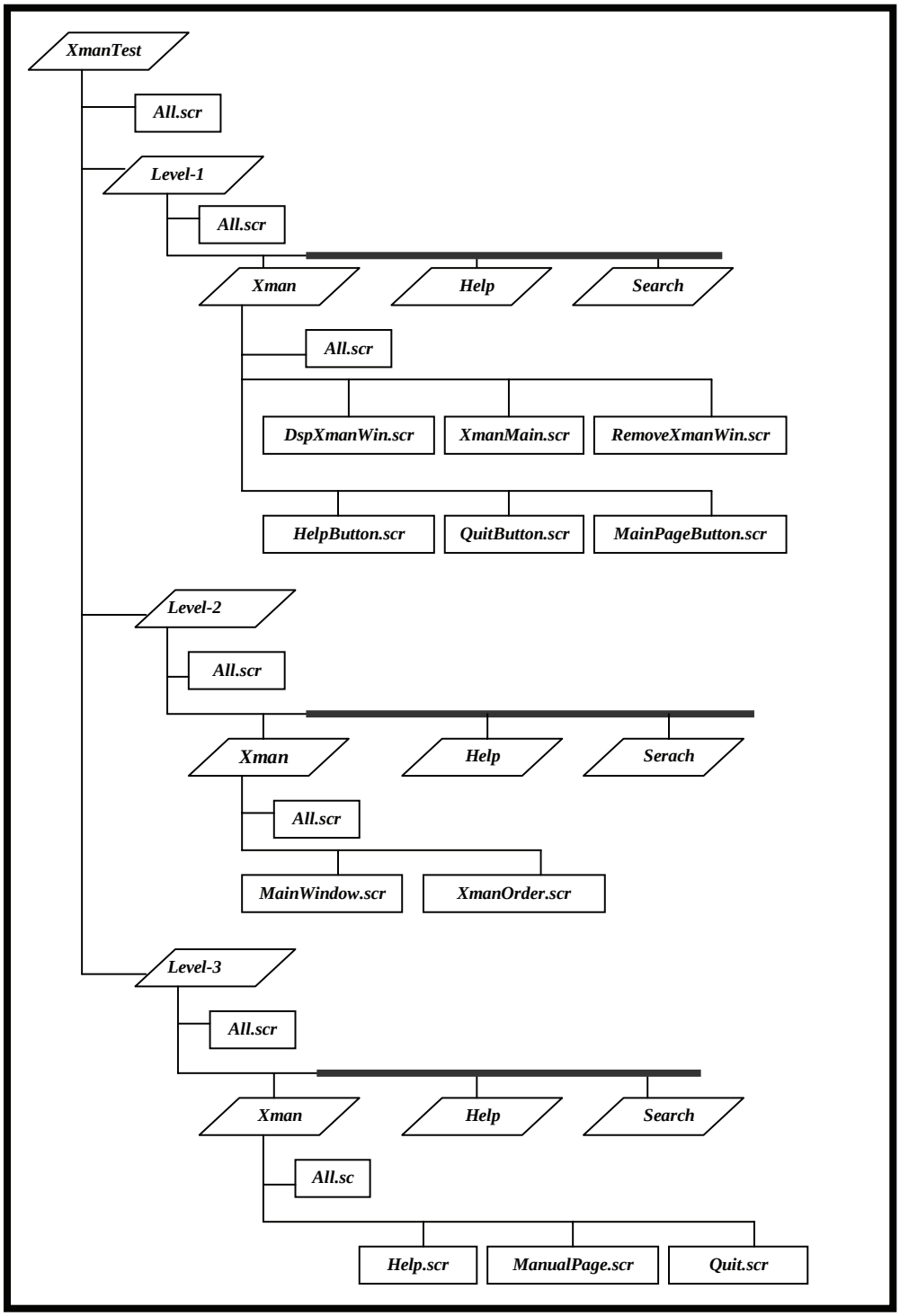


operations on it, and then finally pop-down the window. DspXmanWin.scr script pop-up the Xman window and verifies that looks right RemoveXman Win.scr pops down the Xman window. The script XmanMain.scr ruses DspXmanWin.scr in its entering section to display the Xman window. It verifies the window manager operations in its core section and then uses RemoveHe1p Win. scr script in its leaving section to pop-down the Xman Window. As soon as the Xman window pops up on the screen, we see that it contains *four* objects, i.e., Manual

Browser Label, Help

There are buttons which are Quit and Manual Page. Manual Browser label is a static piece of text and does not respond to any user actions, so we do not need a script for it. The other three (3) objects are active objects and respond to user events so we create one script for each of them. The entering section of each one of these scripts calls DspXmanWin.scr to display the Xman Main Window on the screen. The ending section of each of these scripts call RemoveXmanwin.

Figure 3: Test Suite Hierarchy of Xman



scr to remove the Xman Main Window from the screen. The core section of Help Button.scr script verifies the behavior of Help button in Xman Main Window when it is clicked on by a mouse button. The core sections of QuitButton. scr and ManPageBut ton.scr scripts verify the same thing for Qui t and Manual Page buttons in Xman Main Window.

Second Level

The object list of the second level contains all the top level windows of Xman. As we are considering the Main Window only in this discussion so we assume that it is at the top of the list and is selected for the testing. There is not much interaction going on in the objects which belong to the Xman Main Window. The only interaction is the disappearance of Main Window, in response to click on the Quit button. So there will be only one script related to the Main Window which will verify that a click on the Quit button actually destroys the Main Window of Xman. This script is called MainWindow.scr and is located in ~/XmanTest/Level2/. This script is also used DspXmanWin. scr and RemoveXman.scr script to display and remove the Main Window from the screen. Another potential script, let us call it XmanOrder. scr, related to the Main Window verifies that the order in which Help or Manual Page buttons are pressed is insignificant. No matter the Help button is pressed before or after the Manual Page button, it displays the Help window properly. The same is also true for the Manual Page button.

Third Level

The object list of the third level includes the root object only, and tests any interactions among the top level windows of Xman can be done. Such interactions which involve the Main Window of Xman include display of the Help window and the Manual Page window in response to mouse clicks on the Help and the Manual page buttons, respectively. Similarly, it also includes disappearance of all the windows related to Xman in response to a click on the Quit button. The three scripts provided at this level, that is, Help.scr, ManualPage.scr and Quit.scr, verify the behavior, related to the corresponding button, mentioned above. This level might also include scripts which verify application behavior, like multiple clicks on the Help button and do not create more than one Help windows while each click on the Manual Page button create a new Manual Page window.

Creating Script Drivers

Once all the scripts for Xman has been captured, we need driver scripts so that all the scripts in the entire suite, all the scripts in a particular testing level or all the scripts related to a particular object can be executed automatically in the desired sequence. For example, we create a script driver, at each testing level, which executes all the scripts created for testing Xman Main Window and its descendents, at that particular level. These scripts are ~/XmanTest/Level-l/Xman/All.scr,-/XmanTest/Level-2/Xman/All. scr, and ~/XmanTest/Level-3/Xman/All. scr, respectively. The script ~/ XmanTest/Level-l/All.scr drive all the scripts created for the first testing level and similarly the other two drivers execute scripts related to the other two levels. The script ~/XmanTest/All.scr drives all the scripts in all the three levels of the test suite.

Testing the Test Suite

After the creation of the test suite, it is necessary to replay all the scripts in the suite and verify if they work properly and also to make sure that they cover all the features which need to be tested. Although, without a formal specification, it is impossible to perform any reasonable automatic coverage analysis but at least the replayed events and the objects which appear during the replay,

can be matched against application documentation to determine if any object or event has not been covered by the generated test suite.

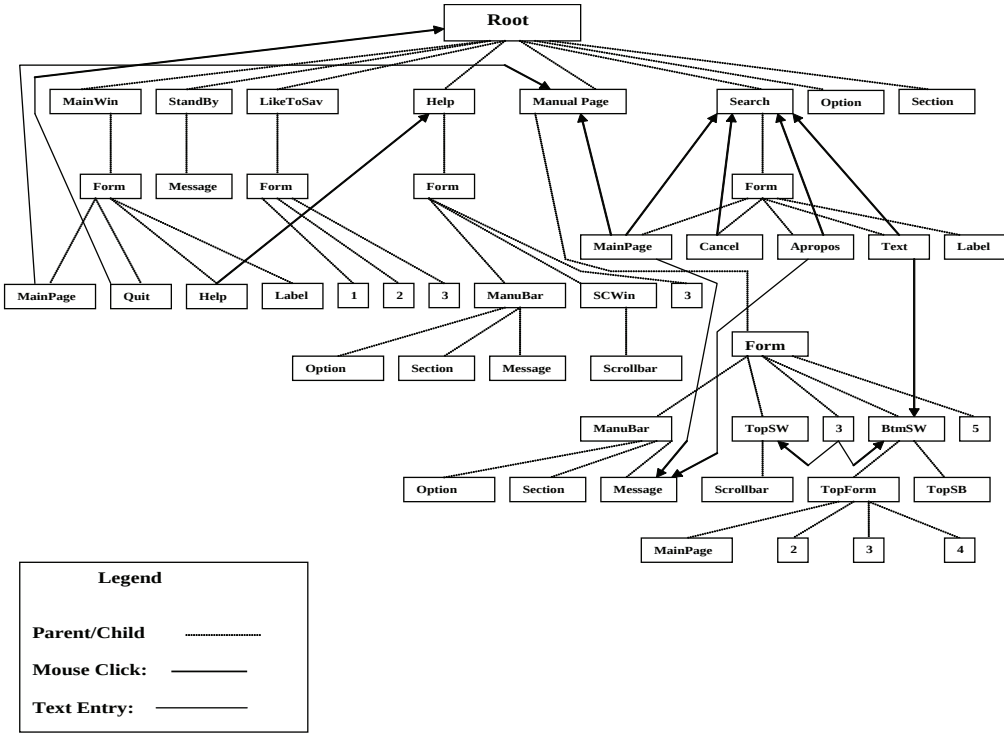
TESTING WITH FORMAL SPECIFICATIONS

This section demonstrates the testing of Xman when we have enough resources to build a formal model for Xman. The following subsection illustrates each step of the methodology, described in Section 2.2.

Building the Model

When the resources permit, the first step for testing is building a formal model for the application under test. Figure 4 displays a User Interface Graph built for Xman. The Root node of the graph represents the root window of the screen. The children of the Root node represent the six top level windows of Xman. The nodes at lower levels in the graph represent the descendents of the top level windows. Let us take the main window of Xman as an example. It is represented as MainWin node in the User Interface Graph. The child of the MainWin node is the Form node which acts as a container widget for the buttons and the label in the main window. The ManPage, Quit and Help nodes represent the Manual. Page, Quit and Help command buttons in the main window, respectively. The label node represents the Manual Browser label in the main window. The dark black arc from the Manpage noqe to the Manual Page node represents the fact that clicking a mouse button over the Manpage

Figure 4. User interface graph built for Xman



button in the main window affects the top level window, called Manual Page. The arc from the Quit node to the Root node represents that a click on the Quit button affects all the top level windows of Xman. The type of event represented by an arc is reflected by the drawing pattern of the arc. The two arcs mentioned above have the same pattern and represent button clicks. An arc with a different pattern is the arc from the Text node to the search node. This pattern represents text entry and the arc represents that entering text in the Text node affect the search dialog box.

Initialization

All the decision and actions taken at the initialization step, that is, the number of testing levels and their organization, the location of the test suite and the application default resources, is kept the same as for testing without a formal specification, described in Section 4.1.

Building Object Lists

After building the user interface graph for Xman, it is possible to build object lists for all levels of testing. This can be done either manually or automatically by scanning each and every node in the User Interface Graph and verifying if it qualifies to be tested on a particular testing level. All the objects in Xman qualify for the first testing level and hence are placed on the list associated with it. The qualifying criterion for the second level is that the object must be a top level window, that is, its corresponding node must be a child of the root of the user interface graph. Some of the objects which qualify for the second testing level are Xman Main Window, Manual Page window, Help window, Search dialog box and so forth. The third level treats the entire Xman application as a single object, and its corresponding node, the root, is the only object which qualifies for the third level of testing.

Building Test Suite

The strategy for capturing scripts without any formal specifications, discussed in Section 4.3, can be used for capturing scripts when the application has been specified formally. However, capturing scripts with formal specifications also provides us the capability to write the test scripts manually or generate test scripts automatically by using the formal model and the test requirements. All of these three techniques for building a test suite are explained in the following section.

Capturing Scripts

A tool can be used for capturing user sessions for building a test suite, in exactly the same way as for capturing without a formal model, as mentioned in Section 4.3. However, the presence of a formal model makes the generated scripts more robust and easy to follow and debug. A formal model also provides the flexibility that the captured scripts can be executed in any order without any conflict in window names. It becomes easier to modify the test suite in case of a modification to the application under test. In fact in most cases the modification can be reflected in the specification file and the test suite remains valid without making any changes.

Writing Scripts Manually

The formal specifications of the application, under test, also allows the user to write scripts manually, even for the first level of testing. This feature is particularly helpful when the test scripts need to be created in parallel with the application development in order to reduce the total development time. The way it can be organized is to formally specify the graphical user interface of the application once the design is complete. The developers and testers agree on this formal specification and any future changes are properly communicated

between the two groups. Having the formal specification of the graphical user interface at hand, the testers can develop the scripts manually in parallel with the application development. Once the object hierarchy and the behavior of the objects is known, the manual writing of test scripts is as easy as writing a UNIX shell script.

Automatic Test Generation

The formal specifications of an application under test also provide capabilities to generate test scripts automatically. A tool can be developed, which will read the specifications of an application and create test scripts for a particular level of testing or the entire test suite. Similar tools can be developed and used to generate test scripts to test all the objects of a particular type or all the user actions of a particular type. For example, such a tool can generate test scripts to exercise clicks on each command button in an entire application. A similar tool can generate a suite of test scripts to verify that all windows in an application under test are displayed on the screen at least once. Another tool might generate scripts to individually select all the options of each menu in the application and verify that the application responds in the expected manner. There can be another tool that will create scripts for the selection of multiple menu options and/or command buttons in different sequences to verify the application response. All the mentioned tools will only create scripts

Figure 5. Pseudo Code for Automatic Test Generator

```
Build the User Interface Graph of the application under
test;
Build an object list by a pre-order traversal of the User
Interface Graph using parent-child relationship arcs.
for each "element on the list
do
   If the element is a top level window
         Create a new directory and change to the
      directory.
         Display the element on the screen.
   fi
   if the element accepts any user events
         Create a script for the element
         for each kind of user event accepted by the
   element
         do
               Add commands in script to
                     Generate the event on the element;
                     Verify the effect of that event;
                     Undo the effect of the event;

         done
   fi
done
```

and validation data that have to be captured by replaying these scripts and by using caprep (Horowitz & Singhera, 1993) or a similar tool to replay these scripts in Update mode.

The logic behind these automatic test generation tools is the same as it is used in Section 5 for manual creation of test suites. The tool starts at the root of the UIG and builds a list of GUI elements by performing a pre-order walk of the UIG. During this walk only the arcs that represent parent-child relationships are considered. After building this list, its entries are taken one by one to create test scripts for them. If the current GUI element, taken from the list, is a top-level

Figure 6. Statistics file created by XTester

```
Legend:
CW=Create Window DW=Destroy Window MW=Map Window
UMW=Unmap Window BP=Button Press BR=Button Release
KP=Key Press KR=Key Release
 Object Name                    CW DW MW UMWBP BR     KP KR
Xman :                          1   0  2  1  0   0    0 0
Xman*Form :                     1   0  2  1  0   0    0 0
Xman*ManualPage :               1   0  2  1  50  50   0 0
Xman*Quit :                     1   0  2  1  0   0    0 0
Xman*Help :                     1   0  2  1  10  10   0 0
Xman*Label :                    1   0  2  1  0   0    0 0
StandBy :                       28  23 2  1  0   0    0 0
StandBy*Message :               28  23 2  1  0   0    0 0
LikeToSave :                    1   0  0  0  0   0    0 0
LikeToSave*Form :               1   0  0  0  0   0    0 0
LikeToSave*Message :            1   Q  0  0  0   0    0 0
LikeToSave*Yes :                1   0  0  0  0   0    0 0
LikeToSave*No :                 1   0  0  0  0   0    0 0
Help :                          1   0  7  7  0   0    0 0
Help*Form :                     1   0  7  7  0   0    0 0
Help*MenuBar :                  1   0  7  7  0   0    0 0
Help*Options :                  1   0  7  7  20  2    0 0
Help*Seetions :                 1   0  7  7  0   0    0 0
Help*Message :                  1   0  7  7  0   0    0 0
Help*TextArea :                 1   0  7  7  0   0    0 0
Help*Serollbar :                1   0  7  7  10  10   0 0
Help. Form. 3 :                 1   0  7  7  0   0    0 0
Manpage :                       27  23 27 1  0   0    0 0
ManPage*Form :                  27  23 27 1  0   0    0 0
ManPage*MenuBar :               27  23 27 1  0   0    0 0
ManPage*Options :               27  23 27 1  92  6    0 0
ManPage*Seetions :              27  23 27 1  18  2    0 0
Manpage*Message :               27  23 27 1  0   0    0 0
ManPage*TextArea :              27  23 27 1  0   0    0 0
ManPage*Serollbar :             27  23 27 1  8   8    0 0
ManPage.Form.3 :                27  23 27 1  0   0    0 0
ManPage*DirArea :               27  23 27 1  0   0    0 0
ManPage*DirList :               27  23 27 1  0   0    0 0
ManPage*List :                  27  23 27 1  0   0    0 0
```

window, then a separate directory is created for it and the scripts for the element and all of its descendents are created in that directory. If the currently selected element belongs to the category for which a script is required, then following the arcs that represent user actions on the element creates one. Figure 4 shows the pseudo code for such an automatic test generator.

Coverage Analysis

No testing activity is useful unless it provides some coverage measures/analysis. This coverage measure reflects the quality of the testing activity. The UIG provides us with a framework to determine such a coverage. During capture or replay of scripts, XTester keeps track of the user actions and their effects on individual objects. This information is available in a .stt file at the end of the testing activity. Currently, the information captured in .stt about a particular object includes the number of times it was created, mapped, unmapped, and destroyed. It also accumulates the number of times a mouse button or keyboard key was pressed or released over the object. This information helps the user to locate any particular objects in the application which have not been created, destroyed, mapped, unmapped, or received a user event. These statistics can also be used for improving the efficiency of the test suite by removing the repetitive testing of the same characteristics, whenever possible. Figure 6 shows a file created by XTester after replaying a certain test suite for Xman. The legend is displayed at the top of the file to describe acronyms for various actions. The next line after the legend provides the heading for the table. Each line in the table provides statistics about a particular object. For example, the very first line of the table provides statistics about the Main Window of Xman, named Xman. This particular line shows that the object named Xman was created once, never destroyed, mapped twice on the screen and unmapped once, by the corresponding test script. It also shows that the corresponding test suite never exercised a button press/release or key press/release events on the Xman object.

Tools can be developed to extract information from a .stt file. Analysis tools are developed that take a .uidl file to build a UIG of the application under test, and a .stt file to get statistics collected from a particular test run. The tool maps the collected statistics on to the object hierarchy and produces an annotated graph. Queries can be run on this annotated graph to determine the effectiveness of the test suite. For example, one can see how many nodes and arcs in the object hierarchy have not been exercised by this particular test suite, or see the objects and inter-object arcs to determine objects that did not receive an expected user action. Similarly, one can filter out the objects that did not receive an expected user event to create an annotated graph that satisfies a particular criterion.

INVALIDATION OF TEST DATA

XTester captures information as two entities, script commands and verification data and saves them in different files, .scr and .img, respectively. This section describes the scenario in which created test script(s) might fail and has to be re-captured or re-written. The following scenarios invalidate some of the captured scripts:

- An application is modified in such a way that WM _NAME property of a top-level window is changed. This modification will only affect the scripts that have been captured without a specification and are related to the window whose WM _NAME property was changed. The scripts captured with a formal specification remain unaffected, provided the relevant specifications are also modified to reflect the change in the application.
- The application is changed so that the order or number of children of a particular node

Figure 7. Pseudo code for quality analyzer

```
Read in the required criteria for analysis;
Build the User Interface Graph of the application
under test;
Read in the specified statistics file(s);
for each element in the User Interface Graph
do
    If the element qualifies for the given criteria
        Display the required information in proper
```

is changed. The scripts that are captured without a specification and address objects in the modified hierarchy, are affected and have to be recaptured or tuned. However, the scripts captured with a formal specification remain unaffected provided the specification is also modified accordingly. XTester provides option to build either full or short object names. If the application is modified so that the depth of a hierarchy is changed, then all the fully qualified names belonging to that hierarchy will no longer be valid names and has to be modified to reflect the change. However, short names that are relative to their immediate parents will still remain valid.

- If object information has been captured as an image, then trying to replay the scripts on another workstation that is incompatible with the workstation on which image was captured will give false alarms. The scripts work fine across multiple platforms, however, verification data is the platform specific in case of images. An easier way of creating verification data for a new hardware platform will be to replay all the scripts in Update mode that replaces the current verification data with the newly available one.

CONCLUSION AND FUTURE DIRECTIONS

In this article, we have suggested guidelines that are useful in planning a testing activity for a graphical user interface (GUI). We have also presented a methodology for testing a GUI, both when specifications of an application under test is not available, and when such specifications are provided or resource code is available to build such a specification. This article also demonstrates the use of the proposed methodology to test a sample X application, Xman, with or without specifications. It also illustrates how the model is helpful in automatic test generation and coverage analysis. In the end, we describe the situations in which the scripts captured by XTester become invalid.

The methodology and the underlying UIG framework discussed in this article can be very effectively used to model and test web sites and web applications. We are actively working to extend this proposed methodology to be used

for testing web-based applications and semantic web applications.

REFERENCES

Berstel, J., Reghizzi, S. C., Roussel, G., & San Pietro, P. (2001). A scalable formal method for design and automatic checking of user interfaces. *In Proceedings of the 23rd International Conference on Software Engineering,* (pp. 453-462).

Campos J., & Harrison, M. (2001). Model checking interactor specifications. *Automated Software Engineering, 3*(8), 275-310.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns*. Addison Wesley Publishers.

Horowitz, E.& Singhera, Z. (1993). Graphical user interface testing. *In proceedings of the Eleventh Annual Pacific Northwest Software Quality Conference.*

Horowitz, E.& Singhera, Z. (1993). XTester – A System for Testing X Applications. *Technical Report No. USC-CS-93-549,* Department of Computer Science, University of Southern California, Los Angeles, CA.

Horowitz, E. & Singhera, Z. (1993). A Graphical User Interface Testing Methodology. *Technical Report No. USC-CS-93-550,* Department of Computer Science, University of Southern California, Los Angeles, CA.

Harold, M. J., Gupta, R., & Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology, 2*(3), 270-285.

Mercury Interactive, Mountain View, CA., http://www.mercury.com

Myers, B. A. (1993). Why are human-computer interfaces difficult to design and implement? *Technical Report CS-93-183,* Carnegie Mellon University, School of Computer Science.

Myers, B. A., Olsen, D. R., Jr., & Bonar, J. G. (1993). User interface tools. In proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems, Adjunct Proceedings, Tutorials, (p. 239).

Sommervill, I. (2001). *Software engineering* (6th ed.) Addison Wesley Publishers.

Segue Software Inc., Newton, MA, http://www.segue.com

Wittel, W. I., Jr. & Lewis, T. G. (1991). Integrating the mvc paradigm into an object-oriented framework to accelerate gui application development. *Technical Report 91-60-D6,* Department of Computer Science, Oregon State University.

This work was previously published in the International Journal of Information Technology and Web Engineering, edited by G. Alkhatib & D. Rine, Volume 3, Issue 2, pp. 1-18, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.25
On Concept Algebra:
A Denotational Mathematical Structure for Knowledge and Software Modeling

Yingxu Wang
University of Calgary, Canada

ABSTRACT

Concepts are the most fundamental unit of cognition that carries certain meanings in expression, thinking, reasoning, and system modeling. In denotational mathematics, a concept is formally modeled as an abstract and dynamic mathematical structure that encapsulates attributes, objects, and relations. The most important property of an abstract concept is its adaptive capability to autonomously interrelate itself to other concepts. This article presents a formal theory for abstract concepts and knowledge manipulation known as "concept algebra." The mathematical models of concepts and knowledge are developed based on the object-attribute-relation (OAR) theory. The formal methodology for manipulating knowledge as a concept network is described. Case studies demonstrate that concept algebra provides a generic and formal knowledge manipulation means, which is capable to deal with complex knowledge and software structures as well as their algebraic operations.

INTRODUCTION

In cognitive informatics, logic, linguistics, psychology, software engineering, and knowledge engineering, concepts are identified as the basic unit of both knowledge and reasoning (Anderson, 1983; Colins & Loftus, 1975; Ganter & Wille, 1999; Hampton, 1997; Hurley, 1997; Matlin, 1998; Murphy, 1993; Wang, 2006a, 2006b, 2006c, 2007a, 2007c; Wang & Wang, 2006; Wilson & Keil, 1999). The rigorous modeling and formal treatment of concepts are at the center of theories for knowledge presentation and manipulation (Smith & Medin, 1981; Wille, 1982; Murphy, 1993; Codin, Missaoui, & Alaoui, 1995; Wilson & Keil, 1999; Yao, 2004; Chen & Yao, 2005). A *concept* in linguistics is a noun or noun-phrase that serves

as the subject of a *to-be* statement (Hurley, 1997; Wang, 2002a, 2006a, 2006c, 2007d). Concepts in cognitive informatics (Wang, 2002a, 2006c, 2007b, 2007e) are an abstract structure that carries certain meaning in almost all cognitive processes such as thinking, learning, and reasoning.

Definition 1. *A concept is a cognitive unit to identify and/or model a real-world concrete entity and a perceived-world abstract subject.*

Based on concepts and their relations, meanings of real-world concrete entities may be represented and semantics of abstract subjects may be embodied. Concepts can be classified into two categories, known as the *concrete* and *abstract* concepts. The former are proper concepts that identify and model real-world entities such as the sun, a pen, and a computer. The latter are virtual concepts that identify and model abstract subjects, which cannot be directly mapped to a real-world entity, such as the mind, a set, and an idea. The abstract concepts may be further classified into *collective* concepts, such as collective nouns and complex concepts, or *attributive* concepts such as qualitative and quantitative adjectives. The concrete concepts are used to embody meanings of subjects in reasoning while the abstract concepts are used as intermediate representatives or modifiers in reasoning.

A concept can be identified by its intension and extension (Hurley, 1997; Smith & Medin, 1981; Wang, 2006c; Wille, 1982; Yao, 2004).

Definition 2. *The intension of a concept is the attributes or properties that a concept connotes.*

Definition 3. *The extension of a concept is the members or instances that the concept denotes.*

For example, the intension of the concept *pen* connotes the attributes of being a writing tool, with a nib, and with ink. The extension of the pen denotes all kinds of pens that share the common attributes as specified in the intension of the concept, such as a ballpoint pen, a fountain pen, and a quill pen.

In computing, a concept is an identifier or a name of a class. The intension of the class is a set of operational attributes of the class. The extension of the class is all its instantiations or objects and derived classes. Concept algebra provides a rigorous mathematical model and a formal semantics for object-oriented class modeling and analyses. The formal modeling of computational classes as a dynamic concept with predesigned behaviors may be referred to "system algebra" (Wang, 2006b, 2007d, 2008b, 2008d).

This article presents a formal treatment of abstract concepts and an entire set of algebraic operations on them. The mathematical model of concepts is established first. Then, the abstract mathematical structure, *concept algebra*, is developed for knowledge representation and manipulation. Based on concept algebra, a knowledge system is formally modeled as a concept network, where the methodology for knowledge manipulating is presented. Case studies demonstrate that concept algebra provides a denotational mathematical means for manipulating complicated abstract and concrete knowledge structures as well as their algebraic operations.

THE MATHEMATICAL MODEL OF ABSTRACT CONCEPTS

This section describes the formal treatment of abstract concepts and a new mathematical structure known as concept algebra in cognitive informatics and knowledge engineering. Before an abstract concept is defined, the semantic environment or context (Chen & Yao, 2005; Ganter & Wille, 1999; Hampton, 1997; Hurley, 1997; Medin & Shoben, 1988) in a given language, is introduced.

Definition 4. *Let $\mathcal{O}$ denote a finite or infinite nonempty set of objects, and $\mathcal{A}$ be a finite or in-*

finite nonempty set of attributes, then a semantic environment or context Θ *is denoted as a triple, i.e.:*

$$\Theta \triangleq (\mathcal{O}, \mathcal{A}, \mathcal{R}) = \mathcal{R} : \mathcal{O} \to \mathcal{O} \,|\, \mathcal{O} \to \mathcal{A} \,|\, \mathcal{A} \to \mathcal{O} \,|\, \mathcal{A} \to \mathcal{A} \quad (1)$$

where $\mathcal{R}$ is a set of relations between $\mathcal{O}$ and $\mathcal{A}$, and | demotes alternative relations.

According to the Object-Attribute-Relation (OAR) model (Wang, 2007c, 2007d; Wang & Wang, 2006), the three essences in Θ can be defined as follows.

Definition 5. *An object o is an instantiation of a concrete entity and/or an abstract concept.*

In a narrow sense, an object is the identifier of a given instantiation of a concept.

Definition 6. *An attribute a is a subconcept that is used to characterize the properties of a given concept by more specific or precise concepts in the abstract hierarchy.*

In a narrow sense, an attribute is the identifier of a subconcept of the given concept.

Definition *7. A relation r is an association between any pair of object-object, object-attribute, attribute-object, and/or attribute-attribute.*

On the basis of OAR and Θ, an abstract concept is a composition of the above three elements as given below.

Definition 8. *An abstract concept c on* Θ *is a 5-tuple, i.e.:*

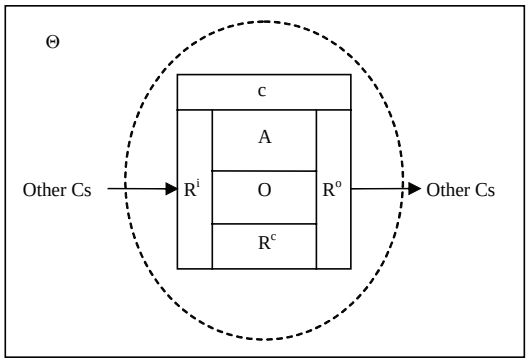
$$c \triangleq (O, A, R^c, R^i, R^o) \quad (2)$$

where

- O is a nonempty set of objects of the concept, $O = \{o_1, o_2, \ldots, o_m\} \subseteq \text{Þ}\mathcal{O}$, where Þ$\mathcal{O}$ denotes a power set of $\mathcal{O}$.
- A is a nonempty set of attributes, $A = \{a_1, a_2, \ldots, a_n\} \subseteq \text{Þ}\mathcal{A}$.
- $R^c = O \times A$ is a set of internal relations.
- $R^i \subseteq C' \times C$ is a set of input relations, where C′ is a set of external concepts.
- $R^o \subseteq C \times C'$ is a set of output relations.

The structure of the concept model $c = (O, A, R^c, R^i, R^o)$ can be illustrated in Figure 1, where c,

Figure 1. The structured model of an abstract concept



A, O, and R, $R = \{R^c, R^i, R^o\}$, denote the identifier of the concept, its attributes, objects, and internal/external relations, respectively.

It is interesting to compare the formal model of abstract concepts as given in Definition 8 with the notion of the concept lattice proposed by Wille (1982). Wille defined a formal concept as a pair of sets of objects and attributes, i.e.:

$$c \triangleq (O, A),\ O \subseteq \mathcal{O} \wedge A \subseteq \mathcal{A} \qquad (3)$$

It is obvious that the abstract concept extends Wille's concept model from a pair to a triple, where the set of relations is explicitly and formally modeled in three categories known as the internal, input, and output relations. The I/O relations enable a conventional static concept to be dynamically associated to other concepts in order to represent and manipulate complicated concept operations and compositions in a concept network and knowledge hierarchy.

Theorem 1. *The dynamic and adaptive property of concepts states that an abstract concept is a dynamic mathematical structure that possesses the adaptive capability to interrelate itself to other concepts via R^i and R^o.*

Based on Definition 8, an object derived from a concept and the intension/extension of a concept can be formally defined as follows.

Definition 9. *An object of a concept o is a derived instantiation of the concept that implements an end product of the concept,$o \subset O$, i.e.:*

$$\forall c(O, A, R^c, R^i, R^o), o = c.o_i, o_i \subset O, R^o{}_o \equiv \varnothing$$
$$\Rightarrow o(A_o, R^c{}_o, R^i{}_o \mid A_o \supseteq A, R^c{}_o = o \times A_o, R^i{}_o = \{(c, o)\}) \qquad (4)$$

Equation 4 indicates that an object is a tailored end-product of a concrete concept where there is no any output-oriented relation to another concept.

Definition 10. *The intension of a concept $c = (O, A, R^c, R^i, R^o)$, c^*, is represented by its set of attributes A, i.e.:*

$$c^*(O, A, R^c, R^i, R^o) \triangleq A = \bigcap_{i=1}^{\#O} A_{o_j} \subseteq \mathbb{P}\mathcal{A} \qquad (5)$$

where $\mathbb{P}\mathcal{A}$ denotes a power set of $\mathcal{A}$, and # is the cardinal operator that counts the number of elements in a given set.

Definition 10 indicates that the *narrow* sense or the exact semantics of a concept is determined by the set of common attributes shared by all of its objects. In contrary, the *broad* sense or the rough semantics of a concept is referred to the set of all attributes identified by any of its objects as defined below.

Definition 11. *The complete set of attributes of a concept $c = (O, A, R^c, R^i, R^o)$, or the instant attributes denoted by all objects of c, is a closure of all objects' intensions, A^*, i.e.:*

$$A^* \triangleq \bigcup_{j=1}^{\#O} A_{o_j} \qquad (6)$$

Definitions 10 and 11 specify that (a) The intension of a concept is a finite set of objectively identifiable attributes at a given level of abstraction, and (b) the intension of a concept is dynamic. When more objects for the same concept are denoted, the domain of the intension is usually shrinking in order to accommodate the new objects in the same structure of the concept.

Conventionally, the *domain* of a concept's intension is used to be perceived subjectively in literature (Hurley, 1997; Matlin, 1998). In this approach, it is deemed that a concept connotes the attributes, which occur in the minds of people who use that concept, or where something must have in order to be denoted by the concept. Both the above informal perceptions are not objectively operational in defining a complete and unambiguity domain of intensions. To solve this

fundamental problem, Definition 10 provides a unique and objective determination of any given concept.

Definition 12. *The extension of a concept $c = (O, A, R^c, R^i, R^o)$, c^+, is represented by its set of objects O, i.e.:*

$$c^+(O,A,R^c,R^i,R^o) \triangleq O = \{o_1, o_2, ..., o_m\} \subseteq \wp\mathcal{O} \tag{7}$$

A formal and objective definition of the domain of intension is provided below.

Definition 13. *The domain of a concept $c = (O, A, R^c, R^i, R^o)$ is a set of attributes with a narrow sense D_{min} referring to its intension and a broad sense D_{max} referring to its closure, i.e.:*

$$D(c) \triangleq \begin{cases} D_{\min}(c) = A = \bigcap_{j=1}^{\#O} A_{o_j} \\ D_{\max}(c) = A^* = \bigcup_{j=1}^{\#O} A_{o_j} \end{cases} \tag{8}$$

It is noteworthy that in conventional literature, it is only believed that the intension of a concept determines its extension (Hurley, 1997; Matlin, 1998). However, Definition 13 reveals that the extension of a concept, particularly the common attributes elicited from the extension, determines its intension as well.

Theorem 2. *The nature of concept hierarchy states that in an abstraction hierarchy, the higher the level of a concept in abstraction, the smaller the intension of the concept; and vice versa.*

Relationships between concepts in a concept hierarchy can be illustrated in Figure 2 at three levels known as the knowledge, object, and attribute levels. The internal relations of concepts, $R^c = O \times A$ can be formally represented by concept matrixes.

Example 1. *The concept matrix of concept c_1 is given in Table 1. According to Definition 13, the intension and extension of concepts c_1 as specified in Table 1 can be objectively and uniquely*

Figure 2. The hierarchical relations of concepts and their internal structures

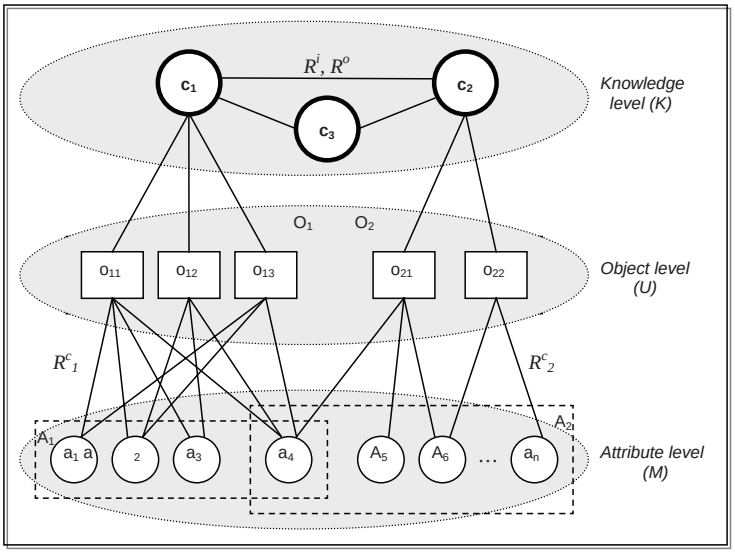


Table 1. The concept matrix of c_1

$\mathbf{c_1}$	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$
$\mathbf{o_{11}}$	1	1	1	1
$\mathbf{o_{12}}$		1	1	1
$\mathbf{o_{13}}$	1	1		1

determined as: $A_{c1} = \{a_2, a_4\}$ and $O_{c1} = \{o_{11}, o_{12}, o_{13}\}$.

Definition 14. *The identification of a new concept $c(O, A, R^c, R^i, R^o)$ is the elicitation of its objects O, attributes A, and internal relations R^c, from the semantic environment $\Theta = (\mathcal{O}, \mathcal{A}, \mathcal{R})$, i.e.:*

$$c \triangleq (O, A, R^c, R^i, R^o \mid O \subset U \ \ A \subset M, R^c = O \times A, R^i = \varnothing, R^c = \varnothing) \quad (9)$$

In Definition 14, $R^i = R^o = \varnothing$ denotes that the identification operation is an initialization of a newly created concept where the input and output relations may be established later.

Definition 15. *A qualification of a concept $c(O, A, R^c, R^i, R^o)$, denoted by $*c$, is the identification of its domain, i.e.:*

$$*c \triangleq D(c) = \begin{cases} D_{\min}(c) = A = \bigcap_{j=1}^{\#O} A_{o_j} \\ D_{\max}(c) = A* = \bigcup_{j=1}^{\#O} A_{o_j} \end{cases} \quad (10)$$

Definition 16. *A quantification of a concept $c(O, A, R^c, R^i, R^o)$, denoted by $\#c$, is the cardinal evaluation of its domain in term of the number of attributes included in it, i.e.:*

$$\#c \triangleq \#D(c) = \begin{cases} \#D_{\min}(c) = \#A = \#(\bigcap_{j=1}^{\#O} A_{o_j}) \\ \#D_{\max}(c) = \#A* = \#(\bigcup_{j=1}^{\#O} A_{o_j}) \end{cases} \quad (11)$$

Example 2. *According to Definitions 15 and 16, the qualification and quantification of concept c_1 as given in Figure 2 are as follows, respectively:*

$$*c_1 = D(c_1) = \begin{cases} D_{\min}(c_1) = A_{c_1} = \{a_2, a_4\} \\ D_{\max}(c_1) = A_{c_1}* = \{a_1, a_2, a_3, a_4\} \end{cases}$$

$$\#c_1 = \#D(c_1) = \begin{cases} \#D_{\min}(c_1) = \#A_{c_1} = 2 \\ \#D_{\max}(c_1) = \#A_{c_1}* = 4 \end{cases}$$

Concept algebra is an abstract mathematical structure for the formal treatment of concepts and their algebraic relations, operations, and associative rules for composing complex concepts.

Definition 17. *A concept algebra CA on a given semantic environment Θ is a triple, i.e.:*

$$CA \triangleq (C, OP, \Theta) = (\{O, A, R^c, R^i, R^o\}, \{\bullet_r, \bullet_c\}, \Theta) \quad (12)$$

where $OP = \{\bullet_r, \bullet_c\}$ are the sets of relational and compositional operations on abstract concepts.

Concept algebra provides a denotational mathematical means for algebraic manipulations of abstract concepts. Concept algebra can be used to model, specify, and manipulate generic "to be" type problems, particularly system architectures, knowledge bases, and detail-level system designs in computing, software engineering, system engineering, and cognitive informatics. The relational and compositional operations on concepts will be formally described in the following sections.

RELATIONAL OPERATIONS OF CONCEPTS

The relational operations of abstract concepts are static and comparative operations that do not change the concepts involved. It is recognized that *relationships* between concepts are solely determined by the relations of both their intensions *A* and extensions *O*. The relational operations on abstract concepts in concept algebra are described below.

Lemma 1. *The relational operations* $\bullet_r$ *in concept algebra encompasses 8 comparative operators for manipulating the algebraic relations between concepts, i.e.:*

$$\bullet_r \triangleq \{\leftrightarrow, \nleftrightarrow, \prec, \succ, =, \cong, \sim, \triangleq\} \tag{13}$$

where the relational operators stand for *related, independent, subconcept, superconcept, equivalent, consistent, comparison,* and *definition*, respectively.

Definition 18. *The related concepts* c_1 *and* c_2 *on* Θ*, denoted by* $\leftrightarrow$*, are a pair of concepts that share some common attributes in their intensions* A_1 *and* A_2*, i.e.:*

$$c_1 \leftrightarrow c_2 \triangleq A_1 \cap A_2 \neq \varnothing \tag{14}$$

Definition 19. *The independent concepts* c_1 *and* c_2 *on* Θ*, denoted by* $\nleftrightarrow$*, are two concepts that their intensions* A_1 *and* A_2 *are disjoint, i.e.:*

$$c_1 \nleftrightarrow c_2 \triangleq A_1 \cap A_2 = \varnothing \tag{15}$$

It is obvious that related and independent concepts are mutually exclusive. That is, if $c_1 \leftrightarrow c_2$, then $\neg (c_1 \nleftrightarrow c_2)$; and vice versa.

Definition 20. *A subconcept* c_1 *of concept* c_2 *on* Θ*, denoted by* $\prec$*, is a concept that its intension* A_1 *is a superset of* A_2*, i.e.:*

$$c_1 \prec c_2 \triangleq A_1 \supset A_2 \tag{16}$$

Definition 21. *A superconcept* c_2 *over concept* c_1 *on* Θ*, denoted by* $\succ$*, is a concept that its intension* A_2 *is a subset of* A_1*, i.e.:*

$$c_2 \succ c_1 \triangleq A_2 \subset A_1 \tag{17}$$

According to Definitions 20 and 21, a subconcept and a superconcept are reflective. That is, *if* $c_1 \prec c_2$, then $c_2 \succ c_1$.

Definition 22. *The equivalent concepts* c_1 *and* c_2 *on* Θ*, denoted by* $=$*, are two concepts that their intensions* (A_1, A_2)*, and extensions* (O_1, O_2) *are identical, i.e.:*

$$c_1 = c_2 \triangleq (A_1 = A_2) \wedge (O_1 = O_2) \tag{18}$$

Definition 23. *The consistent concepts* c_1 *and* c_2 *on* Θ*, denoted by* $\cong$*, are two concepts with a relation of being either a sub- or superconcept, i.e.:*

$$\begin{aligned} c_1 \cong c_2 &\triangleq (c_1 \succ c_1) \vee (c_1 \prec c_2) \\ &= (A_1 \subset A_1) \vee (A_1 \supset A_2) \end{aligned} \tag{19}$$

Definition 24. *A comparison between two concepts* c_1 *and* c_2 *on* Θ*, denoted by* $\sim$*, is an operation that determines the equivalency or similarity level of their intensions, i.e.:*

$$c_1 \sim c_2 \triangleq \frac{\#(A_1 \cap A_2)}{\#(A_1 \cup A_2)} * 100\% \tag{20}$$

The range of equivalency between two concepts is among 0 to 100 %, where 0% means no similarity and 100% means a full similarity. According to Definition 24, It is obvious that:

$$c_1 \sim c_2 = \begin{cases} 100\%, & c_1 = c_2 \\ \frac{\#A_2}{\#A_1} \bullet 100\%, & c_1 \prec c_2 \\ \frac{\#A_1}{\#A_2} \bullet 100\%, & c_1 \succ c_2 \end{cases} \quad (21)$$

Definition 25. *A definition of a concept c_1 by c_2 on Θ, denoted by $\triangleq$, is an association between two concepts where they are equivalent, i.e.:*

$$c_1(O_1, A_1, R^c_1, R^i_1, R^o_1) \triangleq c_2(O_2, A_2, R^c_2, R^i_2, R^o_2)$$
$$\triangleq c_1(O_1, A_1, R^c_1, R^i_1, R^o_1 \mid O_1 = O_2, A_1 = A_2,$$
$$R^c_1 = O_1 \times A_1, R^i_1 = R^i_2, R^o_1 = R^o_2) \quad (22)$$

COMPOSITIONAL OPERATIONS OF CONCEPTS

The compositional operations of concept algebra are dynamic and integrative operations that always change all concepts involved in parallel.

Compositional operations on concepts provide a set of fundamental mathematical means to construct complex concepts on the basis of simple ones or to derive new concepts on the basis of exiting ones.

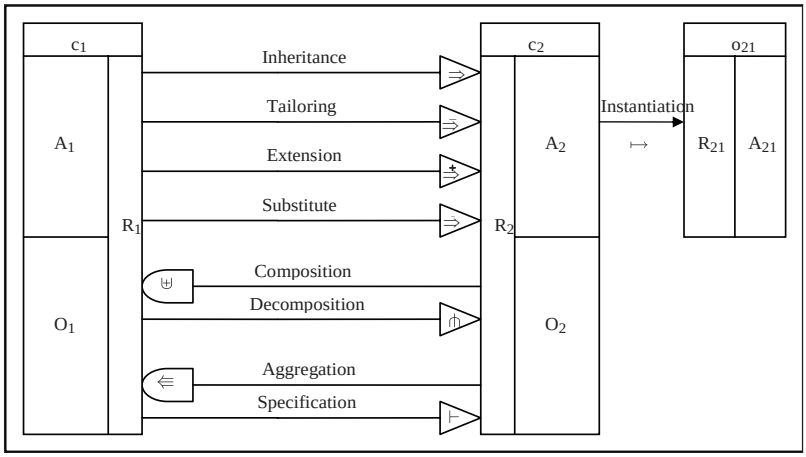
Lemma 2. *The compositional operations $\bullet_c$ in concept algebra encompasses nine associative operators for manipulating the algebraic compositions among concepts, i.e.:*

$$\bullet_c \triangleq \{\Rightarrow, \overset{-}{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \uplus, \pitchfork, \Leftarrow, \vdash, \mapsto\} \quad (23)$$

where the compositional operators stand for *inheritance, tailoring, extension, substitute, composition, decomposition, aggregation, specification,* and *instantiation*, respectively.

The compositional operations of concept algebra can be illustrated in Figure. 3. In Figure 3, $R = \{R^c, R^i, R^o\}$, and all nine compositional operations define composing rules among concepts, except instantiation that is an operation between a concept and a specific object.

Figure 3. The compositional operations of concept algebra as concept manipulation rules



Definition 26. *An inheritance of concept c_2 from concept c_1, denoted by $\Rightarrow$, is the creation of the new concept c_2 by reproducing c_1, and the establishment of new associations between them in parallel, see Box 1, where c_1 is called the parent concept, c_2 is the child concept, and || denotes that an inheritance creates new associations between c_1 and c_2 in parallel via (R^o_1, R^i_2) and (R^o_2, R^i_1).*

Definition 27. *The multiple inheritance of concept c from n parent concepts c_1, c_2, …, c_n, denoted by $\Rightarrow$, is an inheritance that creates the new concept c via a set of n conjoint concepts and establishes new associations among them, see Box 2, where $\overset{n}{\underset{i=1}{R}}(\kappa_i)$ is known as the big-R notation (Wang, 2002b, 2008c) that denotes a repetitive behavior or recurrent structure.*

Definition 28. *The tailoring of concept c_2 from the parent concept c_1, denoted by $\overset{-}{\Rightarrow}$, is a special inheritance that creates the new concept c_2 based on c_1 with the removal of the subsets of objects O′ and attributes A′; at the same time, it establishes new associations between the two concepts, see Box 3.*

Definition 29. *The extension of concept c_2 from the parent concept c_1, denoted by $\overset{+}{\Rightarrow}$, is a special inheritance that creates the new concept c_2 based c_1 with additional objects O′ and/or attributes A′, and establishes new associations between the two concepts, see Box 4.*

Definition 30. *The substitute of concept c_2 from the parent concept c_1, denoted by $\tilde{\Rightarrow}$, is a special*

Box 1.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \Rightarrow c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2 \mid O_2 = O_1, A_2 = A_1, R^c{}_2 = O_2 \times A_2, \\
& \quad R^i{}_2 = R^i{}_1 \cup \{(c_1, c_2)\}, R^o{}_2 = R^o{}_1 \cup \{(c_2, c_1)\}) \\
& \| \, c_1(O_1, A_1, R^c{}_1, R^{i'}{}_1, R^{o'}{}_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{(c_2, c_1)\}, R^{o'}{}_1 = R^o{}_1 \cup \{(c_1, c_2)\})
\end{aligned}
\tag{24}
$$

Box 2.

$$
\begin{aligned}
& \overset{n}{\underset{i=1}{R}}(c_i \Rightarrow c\, O, A, R^c, R^i, R^o) \\
& \triangleq c(O, A, R^c, R^i, R^o \mid O = \bigcup_{i=1}^{n} O_{c_i}, A = \bigcup_{i=1}^{n} A_{c_i}, R^c = O \times A, \\
& \quad R^i = \bigcup_{i=1}^{n} R^i{}_{c_i} \cup \{\overset{n}{\underset{i=1}{R}}(c_i, c)\}, R^o = \bigcup_{i=1}^{n} R^o{}_{c_i} \cup \{\overset{n}{\underset{i=1}{R}}(c, c_i)\}) \\
& \| \overset{n}{\underset{i=1}{R}} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\})
\end{aligned}
\tag{25}
$$

Box 3.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \overset{=}{\Rightarrow} c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2 \mid O_2 = O_1 \setminus O', A_2 = A_1 \setminus A', R^c{}_2 = O_2 \times A_2 \subset R^c{}_1, \\
& \quad R^i{}_2 = R^i{}_1 \cup \{(c_1, c_2)\}, R^o{}_2 = R^o{}_1 \cup \{(c_2, c_1)\}) \\
& \| c_1(O_1, A_1, R^c{}_1, R^{i'}{}_1, R^{o'}{}_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{(c_2, c_1)\}, R^{o'}{}_1 = R^o{}_1 \cup \{(c_1, c_2)\})
\end{aligned}
\tag{26}
$$

Box 4.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \overset{+}{\Longrightarrow} c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2 \mid O_2 = O_1 \cup O', A_2 = A_1 \cup A', R^c{}_2 = O_2 \times A_2 \supset R^c{}_1, \\
& \quad R^i{}_2 = R^i{}_1 \cup \{(c_1, c_2)\}, R^o{}_2 = R^o{}_1 \cup \{(c_2, c_1)\}) \\
& \| c_1(O_1, A_1, R^c{}_1, R^{i'}{}_1, R^{o'}{}_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{(c_2, c_1)\}, R^{o'}{}_1 = R^o{}_1 \cup \{(c_1, c_2)\})
\end{aligned}
\tag{27}
$$

Box 5.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \overset{\sim}{\Rightarrow} c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2 \mid O_2 = (O_1 \setminus O'_{c_1}) \cup O'_{c_2}, A_2 = (A_1 \setminus A'_{c_1}) \cup A'_{c_2}, \\
& \quad R^c{}_2 = O_2 \times A_2, R^i{}_2 = R^i{}_1 \cup \{(c_1, c_2)\}, R^o{}_2 = R^o{}_1 \cup \{(c_2, c_1)\}) \\
& \| c_1(O_1, A_1, R^c{}_1, R^{i'}{}_1, R^{o'}{}_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{(c_2, c_1)\}, R^{o'}{}_1 = R^o{}_1 \cup \{(c_1, c_2)\})
\end{aligned}
\tag{28}
$$

inheritance that creates the new concept c_2 based on c_1 by replacing the inherited subsets of objects O'_{c1} and attributes A'_{c1} with corresponding ones O'_{c2} and A'_{c2} that share the same identifiers but possess different objects or attributes; at the same time, it establishes new associations between the two concepts, see Box 5, where $O'_{c1} \subset O_1 \wedge O'_{c2} \subset O_2 \wedge \#O'_{c1} = \#O'_{c2}$ and $A'_{c1} \subset A_1 \wedge A'_{c2} \subset A_2 \wedge \#A'_{c1} = \#A'_{c2}$.

Binary concept tailoring, extension, and substitution can be extended to corresponding *n*-nary

operations, similar to that of inheritance as given in Definition 27.

Definition 31. *The composition of concept c from n subconcepts c_1, c_2, …, c_n, denoted by $\uplus$, is an integration of them that creates the new super concept c via concept conjunction; at the same time, it establishes new associations between them, see Box 6.*

It is noteworthy that, according to the calculus of incremental union $\boxplus$ (Wang, 2006b, 2008b), the composition operation as given in Definition 31 results in the generation of new internal relations, which do not belong to any of its subconcepts. This is the most important property of concept composition.

Corollary 1. *The composition of multiple concepts is an incremental union operation, where the newly generated internal relations ΔR^c can be determined as:*

$$\Delta R^c = \bigcup_{i=1}^{n} \{(c, c_i), (c_i, c)\} \tag{30}$$

Box 6.

$$\begin{aligned} & c(O, A, R^c, R^i, R^o) \triangleq \biguplus_{i=1}^{n} c_i(O_i, A_i, R_i^c, R_i^i, R_i^o) \\ & = c(O, A, R^c, R^i, R^o \mid O = \bigcup_{i=1}^{n} O_{c_i}, A = \bigcup_{i=1}^{n} A_{c_i}, R^c = \bigcup_{i=1}^{n} R^c{}_{c_i} \cup \{(c, c_i), (c_i, c)\}, \\ & \quad R^i = \bigcup_{i=1}^{n} R^i{}_{c_i} \;\; R^o = \bigcup_{i=1}^{n} R^o{}_{c_i}) \\ & \| \mathop{\mathrm{R}}_{i=1}^{n} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\}) \end{aligned} \tag{29}$$

Box 7.

$$\begin{aligned} & c(O, A, R^c, R^i, R^o) \Cap \mathop{\mathrm{R}}_{i=1}^{n} c_i \\ & \triangleq \mathop{\mathrm{R}}_{i=1}^{n} \{ c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\}) \\ & \quad \| c(O, A, R^{c'}, R^{i'}, R^{o'} \mid R^{c'} = \bigcup_{i=1}^{n} (R^c{}_{c_i} \setminus \{(c, c_i)(c_i, c)\}), \\ & \quad \{R^{i'} = R^i \cup \mathop{\mathrm{R}}_{i=1}^{n} (c_i, c)\}, R^{o'} = R^o \cup \{\mathop{\mathrm{R}}_{i=1}^{n} (c, c_i)\}) \\ & \quad \setminus c_i(O_i, A_i, R^c{}_i, R^i{}_i, R^o{}_i) \\ & \quad \} \end{aligned} \tag{31}$$

A concept decomposition is an inverse operation of concept compositions.

Definition 32. *The decomposition of concept c into n subconcepts c_1, c_2, …, c_n, denoted by ⫛, is a partition of the superconcept into multiple subconcepts; at the same time, it establishes new associations between them, see Box 7.*

As specified in Definition 32, the decomposition operation results in the removal of all internal relations $\Delta R^c = \bigcup_{i=1}^{n} \{(c, c_i), (c_i, c)\}$ that are no longer belong to any of its subconcepts.

Definition 33. *The aggregation of concept c_1 from concept c_2, denoted by ⇚, is a creation of c_1 via abstraction of c_2 with a reduced intension of more generic attributes; at the same time, it establishes new associations between them, see Box 8.*

Concept aggregation is also known as concept *generalization*, *abstraction*, or *elicitation*. Binary aggregations can be extended to *n*-nary parallel or serial aggregations.

Definition 34. *The parallel aggregation of a concept c from a set of n concepts c_1, c_2, …, c_n,*

Box 8.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \Leftarrow c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1 \mid O_1 \supset O_2, A_1 \subset A_2, R^c{}_1 = (O_1 \times A_1) \cup \\
& \qquad \{(c_1, c_2), (c_2, c_1)\}, R^i{}_1 = R^i{}_2 \cup \{(c_2, c_1)\}, R^o{}_1 = R^o{}_2 \cup \{(c_1, c_2)\}) \\
& \| c_2(O_2, A_2, R^c{}_2, R^{i'}{}_2, R^{o'}{}_2 \mid R^{i'}{}_2 = R^i{}_2 \cup \{(c_1, c_2)\}, R^{o'}{}_2 = R^o{}_2 \cup \{(c_2, c_1)\})
\end{aligned}
\tag{32}
$$

Box 9.

$$
\begin{aligned}
& c(O, A, R^c, R^i, R^o) \Leftarrow \mathop{R}_{i=1}^{n} c_i \\
& \triangleq c(O, A, R^c, R^i, R^o \mid O = \bigcup_{i=1}^{n} O_i, A = \bigcap_{i=1}^{n} A_i, R^c = \bigcup_{i=1}^{n} (R^c{}_{c_i} \cup \{(c, c_i), (c_i, c)\}), \\
& \qquad (R^i = \bigcup_{i=1}^{n} R^i{}_{c_i} \cup \{(c_i, c)\}), R^o = \bigcup_{i=1}^{n} (R^o{}_{c_i} \cup \{(c, c_i)\}), \\
& \| \mathop{R}_{i=1}^{n} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\})
\end{aligned}
\tag{33}
$$

denoted by ⇐, is an aggregation of c with the elicitation of all concepts in the set, see Box 9.

Based on Definition 34, a concept may be inductively generalized by a series of aggregations with a smaller set of more abstract (super) attributes.

Definition 35. *The serial aggregation of a concept c from a set of n concepts c_1, c_2, …, c_n, denoted by ⇐, is an aggregation with a total order of a series of decreasing intensions of the concepts by more abstract and generic attributes, see Box 10.*

A concept specification is an inverse operation of concept aggregations.

Definition 36. *The specification of concept c_1 by concept c_2, denoted by ⊢, is a deductive refinement of c_1 by an increasing intension with more specific and precise attributes in c_2; at the same time, it establishes new associations between them, see Box 11.*

Binary specifications can be extended to *n*-nary parallel or serial aggregations.

Definition 37. *The parallel specification of concept c by a set of n concepts c_1, c_2, …, c_n, denoted by ⊢, is a specification of c with the elicitation of all concepts in the set, see Box 12.*

Box 10.

$$
\begin{aligned}
& c(O, A, R^c, R^i, R^o) \Leftarrow (c_1 \Leftarrow c_2 \Leftarrow \ldots \Leftarrow c_n) \\
& \triangleq c(O, A, R^c, R^i, R^o \mid O \supset O_1 \supset O_2 \supset \ldots \supset O_n, A \subset A_1 \subset A_2 \subset \ldots \subset A_n, \\
& \quad R^c = (O \times A) \cup \{(c, c_1), (c_1, c)\}, R^i = R^i{}_1 \cup \{\mathop{R}_{i=1}^{n}(c_i, c)\}, \\
& \quad R^o = R^o{}_1 \cup \{\mathop{R}_{i=1}^{n}(c, c_i)\}) \\
& \| \mathop{R}_{i=1}^{n} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\})
\end{aligned}
\tag{34}
$$

Box 11.

$$
\begin{aligned}
& c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \vdash c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2) \\
& \triangleq c_2(O_2, A_2, R^c{}_2, R^i{}_2, R^o{}_2 \mid O_2 \subset O_1, A_2 \supset A_1, R^c{}_2 = (O_2 \times A_2) \cup \\
& \quad \{(c_2, c_1), (c_1, c_2)\}, R^i{}_2 = R^i{}_1 \cup \{(c_1, c_2)\}, R^o{}_2 = R^o{}_1 \cup \{(c_2, c_1)\}) \\
& \| c_1(O_1, A_1, R^c{}_1, R^{i'}{}_1, R^{o'}{}_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{(c, c_1)\}, R^{o'}{}_1 = R^o{}_1 \cup \{(c_1, c)\})
\end{aligned}
\tag{35}
$$

Box 12.

$$\begin{aligned}
&\mathop{R}_{i=1}^{n} c_i \vdash c(O, A, R^c, R^i, R^o) \\
&\triangleq c(O, A, R^c, R^i, R^o \mid O \subset \bigcup_{i=1}^{n} O_i, A = \bigcap_{i=1}^{n} A_i, R^c = \bigcup_{i=1}^{n} (R^c_{c_i} \cup \{(c, c_i), (c_i, c)\}), \\
&\quad R^i = \bigcup_{i=1}^{n} (R^i_{c_i} \cup \{(c_i, c)\}), R^o = \bigcup_{i=1}^{n} (R^o_{c_i} \cup \{(c, c_i)\})) \\
&\mid\mid \mathop{R}_{i=1}^{n} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\})
\end{aligned} \tag{36}$$

Definition 38. *The serial specification of concept c by a set of concepts c_1, c_2, …, c_n, denoted by $\vdash$, is a specification with a total order of a series of refinements by increasing intensions of the concepts with more specific and precise attributes, see Box 13.*

The *binary, parallel,* and *series* specifications and aggregations of concepts provide a generic means for forming a hierarchical structure of concepts in knowledge engineering.

Theorem 3. *A totally ordered series of decreasing intensions in a serial concept aggregation is reversely proportional to a totally ordered series of increasing extensions, i.e.:*

Box 13.

$$\begin{aligned}
&(c_n \vdash ... \vdash c_2 \vdash c_1) \vdash c(O, A, R^c, R^i, R^o) \\
&\triangleq c(O, A, R^c, R^i, R^o \mid O_n \supset ... \supset O_2 \supset O_1 \supset O, A_n \subset ... \subset A_2 \subset A_1 \subset A, \\
&\quad R^c = (O \times A) \cup \mathop{R}_{i=1}^{n} \{(c, c_i), (c_i, c)\}, \\
&\quad R^i = R^i{}_n \cup \mathop{R}_{i=1}^{n} \{(c_i, c)\}), R^o = R^o{}_n \cup \mathop{R}_{i=1}^{n} \{(c, c_i)\}) \\
&\mid\mid \mathop{R}_{i=1}^{n} c_i(O_i, A_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{(c, c_i)\}, R^{o'}{}_i = R^o{}_i \cup \{(c_i, c)\})
\end{aligned} \tag{37}$$

$$\forall c \Leftarrow c_1 \Leftarrow c_2 \Leftarrow ... \Leftarrow c_n,$$
$$A \subset A_1 \subset A_2 \subset ... \subset A_n \Rightarrow O \supset O_1 \supset O_2 \supset ... \supset O_n \quad (38)$$

Example 3. *The relationships between series of specifications and aggregations on the concept animal can be described as follows:*

$$(animal \vdash mammal \vdash feline \vdash tiger) \Rightarrow$$
$$(animal \Leftarrow mammal \Leftarrow feline \Leftarrow tiger)$$

where, according Theorem 3, it can be obtained:

$$A_{animal} \subset A_{mammal} \subset A_{feline} \subset A_{tiger}, \text{and}$$
$$O_{animal} \supset O_{mammal} \supset O_{feline} \supset O_{tiger}$$

The compositional operations of concepts formally defined so far are those among abstract concepts. The remainder of this section describes a special compositional operation between a given concept and its objects.

Definition 39. *The instantiation of a concept c, denoted by* $\mapsto$*, is an embodiment of its generic semantics onto a specific case or implementation known as an object o, see Box 14.*

It is noteworthy that the output relation of an object is always empty (i.e., $R^o_o \equiv \varnothing$), which means that the object is an end product of a concept where there is no further deduction of meanings in the hierarchy of the inheritance chain.

Definition 40. *The multiple instantiation of a concept c onto n objects, denoted by* $\mapsto$*, is a compound parallel instantiation that creates a set of new objects {o_1, o_2, …, o_n} based on c, and establishes new associations between them, see Box 15.*

Box 14.

$$c(O, A, R^c, R^i, R^o) \mapsto o(A_o, R^c{}_o, R^i{}_o)$$
$$\triangleq o(A_o, R^c{}_o, R^{i'}{}_o \mid o \subset O, A_o = A, R^c{}_o = o \times A_o, R^{i'}{}_o = R^i{}_o \cup \{(c, o)\})$$
$$||\, c(O, A, R^c, R^{i'}, R^{o'} \mid R^{i'} = R^i \cup \{(o, c)\}, R^{o'} = R^o \cup \{(c, o)\}) \quad (39)$$

Box 15.

$$c(O, A, R^c, R^i, R^o) \mapsto \mathop{R}_{i=1}^{n} o_i(A_{o_i}, R^c{}_{o_i}, R^i{}_{o_i})$$
$$\triangleq \mathop{R}_{i=1}^{n} o_i(A_{o_i}, R^c{}_{o_i}, R^{i'}{}_{o_i} \mid o_i \subseteq O, A_{o_i} = A, R^c{}_{o_i} = o_i \times A_{o_i},$$
$$R^{i'}{}_{o_i} = R^i{}_{o_i} \cup (c, o_i)\})$$
$$||\, c(O, A, R^c, R^{i'}, R^{o'} \mid R^{i'} = R^i \cup \{\mathop{R}_{i=1}^{n}(o_i, c)\}, R^{o'} = R^o \cup \{\mathop{R}_{i=1}^{n}(c, o_i)\}) \quad (40)$$

CONCEPT ALGEBRA FOR KNOWLEDGE MANIPULATION

This section describes applications of concept algebra in the manipulation of abstract models of knowledge and the methodology for knowledge representation and manipulation.

The Mathematical Model of Knowledge

In cognitive informatics (Wang, 2002a, 2006a, 2007e), particularly the OAR model (Wang, 2007c; Wang & Wang, 2006) on internal knowledge representation in the brain, human *knowledge* is modeled as a concept network, where concept algebra is applied as a set of rules for knowledge composition in order to construct complex and dynamic concept networks.

Definition 41. *A generic knowledge K is an n-nary relation* $\Re$ *among a set of n concepts and the entire set of concepts C, i.e.:*

$$K = \Re : (\bigtimes_{i=1}^{n} C_i) \to C \tag{41}$$

where $\biguplus_{i=1}^{n} C_i = C$, *and*

$$\Re = \bullet_c = \{\Rightarrow, \overline{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \uplus, \sqcap, \Leftarrow, \vdash, \mapsto\}.$$

According to Definition 41, the most simple knowledge *k* is a binary relation $\Re$ between two concepts in *C*, i.e.:

$$k = \Re : C \times C \to C. \tag{42}$$

Definition 41 indicates that the compositional operations of concept algebra, $\bullet_c$, provide a set of coherent mathematical means and rules for knowledge manipulation. Because the relations between concepts are transitive, the generic topology of knowledge is a hierarchical network as shown in Figures 2 and 3.

Theorem 4. *The generic topology of abstract knowledge systems K is a hierarchical concept network.*

Theorem 4 can be proved by the nine compositional rules in concept algebra, particularly the composition/decomposition and aggregation/specification operations, as defined in the previous section.

Corollary 2. *The property of the hierarchical knowledge architecture K in the form of concept networks is as follows:*

a. **Dynamic:** *The knowledge network may be updated dynamically along with information acquisition and learning without destructing the existing concept nodes and relational links.*
b. **Evolvable:** *The knowledge network may grow adaptively without changing the overall and existing structure of the hierarchical network.*

The Hierarchical Model of Concept Networks

A concept network as a generic knowledge model has been widely studied in linguistics, computing, and cognitive informatics. The notion of the *semantic network* model for knowledge representation is first proposed by Quillian in 1968 (Matlin, 1998; Quillian, 1968; Reisberg, 2001), where the semantic memory is perceived as information represented in network structures with conceptual nodes and interrelations. The meaning of a given concept depends on other concepts to which it is connected in the network. The semantic network has been extended by a number of theories such as the *hypothetical network* (Colins & Loftus, 1975), the *adaptive control of thought - star* (ADT*) model (Anderson, 1983, 1991). The latter proposes that all cognition processes in thought are controlled by unitary network models.

This subsection develops the concept network model based on concept algebra and Wang's OAR model (Wang, 2007c; Wang & Wang, 2006) for knowledge representation, which treat a concept as a basic and adaptive unit for knowledge representation and thinking.

Definition 42. *A concept network CN is a hierarchical network of concepts interlinked by the set of nine composing rules ℜ concept algebra, i.e.:*

$$CN = \Re : \sum_{i=1}^{n} C_i \rightarrow \sum_{j=1}^{n} C_j \tag{43}$$

Theorem 5. *In a concept network CN, the abstract levels of concepts ℓ_c form a partial order of a series of superconcepts, i.e.:*

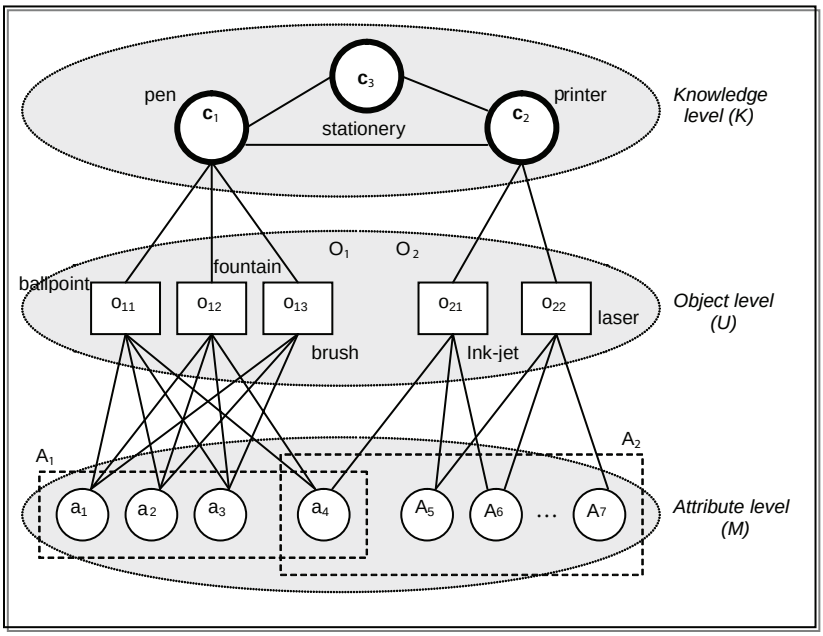
$$\ell_c = (\varnothing \preccurlyeq c_1 \preccurlyeq c_2 \preccurlyeq \ldots \preccurlyeq c_n \preccurlyeq \ldots \preccurlyeq \Omega) \tag{44}$$

where ∅ is the empty concept ∅ = (⊥, ⊥), and Ω the universal concept, Ω = (𝒪, 𝒜).

According to Theorem 5 and Definition 42, a hierarchical structure of concepts in a given semantic environment Θ can be formally described by concept algebra. The algebraic relations and compositional operations of concept algebra enable the construction of hierarchical concept networks in a dynamic process.

Example 4. *A concrete concept network, pen and printer, can be illustrated in Figure 4. The concept network may be dynamically extended along with the development of related knowledge such as to extend it to a more abstract concept network of stationery. In Figure 4, concept c_1(pen) may be formally described in concept algebra, see Box 16.*

Figure 4. A concrete concept network



Box 16.

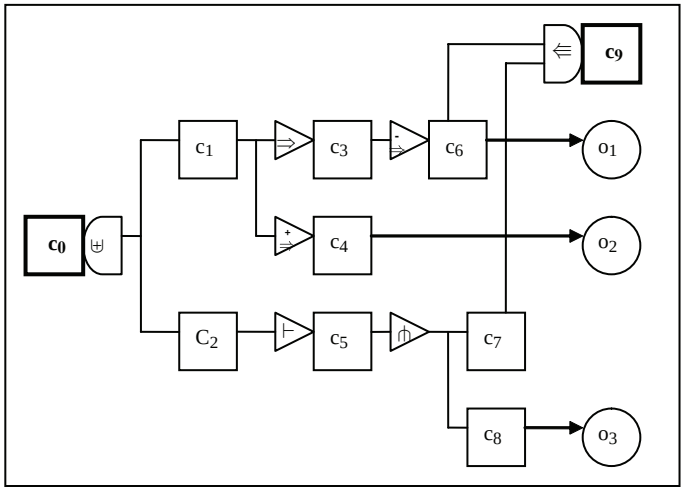
$$
\begin{aligned}
& c_1(pen) = c_1(O_1, A_1, R^c{}_1, R^i{}_1, R^o{}_1) \\
& \text{where } O_1 = \{o_{11}, o_{12}, o_{13}\} = \{ballpoinbt, fountain, brush\}; \\
& \quad A_1 = \{a_1, a_2, a_3\} = \{a\_writing\_tool, using\_ink, having\_a\_nib\}; \\
& \quad A^*{}_1 = \{a_1, a_2, a_3, a_4\} = A \cup \{with\_an\_ink\_container\}; \\
& \quad R^c{}_1 = O_1 \times A_1 = \{(o_{11}, a_1), (o_{11}, a_2), (o_{11}, a_3)\} \cup \\
& \qquad \{(o_{12}, a_1), (o_{12}, a_2), (o_{12}, a_3)\} \cup \{(o_{13}, a_1), (o_{13}, a_2), (o_{13}, a_3)\}
\end{aligned}
\tag{45}
$$

It is noteworthy that, according to Definition 10, the intension of c_1(*pen*) does not include the attribute a_4, because it is not commonly shared by all objects of the given concept. However, A_1* does include a_4 in the closure of attributes of the given concept *pen*.

Example 5. *An abstract concept network that is formed by the composition and aggregation of a set of related concepts c_0 through c_9, as well as objects o_1 through o_3, can be expressed in Figure 5.*

A formal description corresponding to the above concept network can be carried out using concept algebra as given below:

Figure 5. An abstract concept network



$$c_0 \uplus ((c_1 \Rightarrow c_3 \overline{\Rightarrow} c_6 \mapsto o_1) \parallel (c_1 \overset{+}{\Rightarrow} c_4 \mapsto o_2)) \parallel (c_2 \vdash c_5 \pitchfork (c_7 \parallel (c_8 \mapsto o_3)))$$

$$c_9 \Leftarrow (c_6 \parallel c_7) \tag{46}$$

The case studies in Examples 4 and 5 demonstrate that concept algebra and concept network are a generic and formal knowledge manipulation means that are capable to deal with complicated abstract or concrete knowledge structures and their algebraic operations. Further, detailed concept operations of concept algebra may be extended into a set of inference processes, which can be formally described by RTPA (Wang, 2002b, 2003, 2006a, 2007b, 2007d, 2008a, 2008d) as a set of behavioral processes.

CONCLUSION

A new mathematical structure, known as concept algebra, has been presented for abstract concepts and knowledge representation and manipulation. Concepts have been treated as the basic unit of cognition that carries certain meanings in almost all cognitive processes such as thinking, learning, reasoning, and system design. Abstract concepts have been formally modeled as a dynamic mathematical structure with internal attributes, objects, and their relations that possess the adaptive capability to grow in concept networks. The formal methodology for manipulating knowledge has been developed using concept algebra and concept networks. A number of case studies have been used to demonstrate the expressive power of concept algebra in knowledge representation and manipulation. A wide range of applications of concept algebra has been identified for solving common problems in cognitive informatics, logic, linguistics, psychology, knowledge engineering, data mining, software engineering, and intelligence science. One of the important applications of concept algebra is the formalization of object-oriented methodologies and the development of a rigorous semantics of UML as an industrial OO design languages. Autonomic machine learning and searching engines can be developed on the basis of concept algebra.

ACKNOWLEDGMENT

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support of this work. The author would like to thank anonymous reviewers for their valuable comments and suggestions. The author is grateful to Dr. Y. Y. Yao for many insightful discussions.

REFERENCES

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1991). Is human cognition adaptive? *Behavioral and Brain Science, 14,* 71-517.

Chen, Y., & Yao, Y. Y. (2005). Formal concept analysis based on hierarchical class analysis. In *Proceedings of the Fourth IEEE International Conference on Cognitive Informatics (ICCI'05)* (pp. 285-292). Irvin, CA: IEEE CS Press.

Codin, R., Missaoui, R., & Alaoui, H. (1995). Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence, 11*(2), 246-267.

Colins, A. M., & Loftus, E. F. (1975). A Spreading-activation theory of semantic memory. *Psychological Review, 82,* 407-428.

Ganter, B., & Wille, R. (1999). *Formal concept analysis.* Berlin, Germany: Springer.

Hampton, J. A. (1997). *Psychological representation of concepts of memory* (pp. 81-11). Hove, England: Psychology Press

Hurley, P. J. (1997), *A concise introduction to logic* (6th ed.). Belmony, CA: Wadsworth.

Matlin, M. W. (1998). *Cognition* (4th ed.). New York: Harcourt Brace.

Medin, D. L., & Shoben, E. J. (1988). Context and structure in conceptual combination. *Cognitive Psychology, 20,* 158-190.

Murphy, G. L. (1993). Theories and concept formation. In I. V. Mechelen et al. (Eds.), *Categories and concepts, theoretical views and inductive data analysis* (pp. 173-200). New York: Academic Press.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing.* Cambridge, MA: MIT Press.

Reisberg, D. (2001). *Cognition: Exploring the science of the mind* (2nd ed.). New York: Norton.

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Wang, Y. (2002a). Keynote lecture: On cognitive informatics. In *Proceedings of the First IEEE International Conference on Cognitive Informatics (ICCI'02)* (pp. 34-42). Calgary, Canada: IEEE CS Press.

Wang, Y. (2002b). The real-time process algebra (RTPA). *The International Journal of Annals of Software Engineering, 14,* 235-274.

Wang, Y. (2003). Using process algebra to describe human and software system behaviors. *Brain and Mind, 4*(2), 199-213.

Wang, Y. (2006a, July). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation, invited plenary talk. In *Proceedings of the First International Conference on Rough Set and Knowledge Technology (RSKT'06)* (LNAI 4062, pp. 69-78). Chongqing, China: Springer.

Wang, Y. (2006b). On abstract systems and system algebra. In *Proceedinngs of the Fifth IEEE International Conference on Cognitive Informatics (ICCI'06)* (pp. 332-343). Beijing, China: IEEE CS Press.

Wang, Y. (2006c). On concept algebra and knowledge representation. In *Proceedings of the Fifth IEEE International Conference on Cognitive Informatics (ICCI'06)* (pp. 320-331), Beijing, China: IEEE CS Press.

Wang, Y. (2006d, March). On the informatics laws and deductive semantics of software. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 161-171.

Wang, Y. (2007a). The cognitive processes of formal inferences. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(4), 75-86.

Wang, Y. (2007b). Keynote speech: On theoretical foundations of software engineering and denotational mathematics. In *Proceedings of the Fifth Asian Workshop on Foundations of Software* (pp. 99-102). Xiamen, China:.

Wang, Y. (2007c). The OAR model of neural informatics for internal knowledge representation in the brain. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(3), 64-75.

Wang, Y. (2007d, July). Software engineering foundations: A software science perspective. In *CRC Series in Software Engineering: Vol. 2*. CRC Press.

Wang, Y. (2007e). The theoretical framework of cognitive informatics. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(1), 1-27.

Wang, Y. (2008a, April). Deductive semantics of RTPA. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 96-121.

Wang, Y. (2008b, April). On system algebra: A denotational mathematical structure for abstract system modeling. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 20-43.

Wang, Y. (2008c). On the big-R notation for describing iterative and recursive behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(1), 17-18.

Wang, Y. (2008d, April). RTPA: A denotational mathematics for manipulating intelligent and computational behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 44-62.

Wang, Y., & Wang, Y. (2006, March). On cognitive informatics models of the brain. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 203-207.

Wille, R. (1982). Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival (Ed.), *Ordered sets* (pp. 445-470). Reidel, Dordrecht

Wilson, R. A., & Keil, F. C. (Eds.). (1999). *The MIT encyclopedia of the cognitive sciences*. Cambridge, MA: The MIT Press.

Yao, Y. Y. (2004). Concept formation and learning: A cognitive informatics perspective. In *Proceedings of the Third IEEE International Conference on Cognitive Informatics (ICCI'04)* (pp. 42-51). Victoria, British Columbia, Canada: IEEE CS Press.

This work was previously published in the International Journal of Cognitive Informatics and Natural Intelligence, edited by Y. Wang, Volume 2, Issue 2, pp. 1-19, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.26
On System Algebra:
A Denotational Mathematical Structure for Abstract System Modeling

Yingxu Wang
University of Calgary, Canada

ABSTRACT

Systems are the most complicated entities and phenomena in abstract, physical, information, and social worlds across all science and engineering disciplines. System algebra is an abstract mathematical structure for the formal treatment of abstract and general systems as well as their algebraic relations, operations, and associative rules for composing and manipulating complex systems. This article presents a mathematical theory of system algebra and its applications in cognitive informatics, system engineering, software engineering, and cognitive informatics. A rigorous treatment of abstract systems is described, and the algebraic relations and compositional operations of abstract systems are analyzed. System algebra provides a denotational mathematical means that can be used to model, specify, and manipulate generic "to be" and "to have" type problems, particularly system architectures and high-level system designs, in computing, software engineering, system engineering, and cognitive informatics.

INTRODUCTION

Systems are the most complicated entities and phenomena in abstract, physical, information, and social worlds across all science and engineering disciplines. Systems are needed because the physical and/or cognitive power of an individual component or a person is not enough to carry out a work or solve a problem. System philosophy intends to treat everything as a system, and it perceives that a system always belongs to other super system(s) and contains more subsystems.

The system concept can be traced back to the 17th century, when René Descartes (1596-1650) noticed the interrelationships among scientific disciplines as a system. The general system notion was then proposed by Ludwig von Bertalanffy in the 1920s (von Bertalanffy, 1952; Ellis & Fred,

1962). The theories of system science have evolved from classic theories (Ashby, 1958, 1962; Ellis & Fred, 1962; Heylighen, 1989; G. J. Klir, 1992; R. G. Klir, 1988; Rapoport, 1962) to contemporary theories in the mid-20th century, such as I. Prigogine's dissipative structure theory (Prigogine et al., 1972), H. Haken's synergetics (Haken, 1977), and Eigen's hypercycle theory (Eigen & Schuster, 1979). Then, during the late part of the last century, there are proposals of complex systems theories (G. J. Klir, 1992; Zadeh, 1973), fuzzy theories (Zadeh, 1965, 1973), and chaos theories (Ford, 1986; Skarda & Freeman, 1987).

System algebra is an abstract mathematical structure for the formal treatment of abstract and general systems as well as their algebraic relations, operations, and associative rules for composing and manipulating complex systems. System algebra (Wang, 2005, 2006a, 2006b, 2007a, 2007c) presented in this article is the latest attempt to provide a formal and rigorous treatment of abstract systems and their properties. This article treats systems as a mathematic entity and it studies the generic rules and theories of abstract systems. A new mathematical structure of abstract systems as the most complicated mathematical entities beyond sets, functions, and processes is presented. Properties of abstract systems are modeled and analyzed. System algebra is introduced as a set of relational and compositional operations for manipulating abstract systems and their composing rules. The relational operations of system algebra are described encompassing *independent, related, overlapped, equivalent, subsystem,* and *supersystem.* The compositional operations of system algebra are explored encompassing *inheritance, tailoring, extension, substitute, difference, composition, decomposition, aggregation,* and *specification*. A wide range of applications of system algebra are identified in cognitive informatics, system science, system engineering, computing, software engineering, and intelligent systems.

THE ABSTRACT SYSTEM THEORY

This section demonstrates that systems may be treated rigorously as a new mathematical structure beyond conventional mathematical entities. Based on this view, the concept of abstract systems and their mathematical models are introduced.

Definition 1. *An abstract system is a collection of coherent and interactive entities that has stable functions and a clear boundary with the external environment.*

An abstract system forms the generic model of various real-world systems and represents the most common characteristics and properties of them.

Lemma 1. *The generality principle of system abstraction states that a system can be represented as a whole in a given level k of reasoning, $1 \le k \le n$, without knowing the details at levels below k.*

Definition 2. *Let $\mathcal{C}$ be a finite or infinite nonempty set of components, and $\mathcal{B}$ a finite or infinite nonempty set of behaviors, then the universal system environment U is denoted as a triple, i.e.:*

$$\begin{aligned} \mathfrak{U} &\triangleq (\mathcal{C}, \mathcal{B}, \mathcal{R}) \\ &= \mathcal{R} : \mathcal{C} \to \mathcal{C} \mid \mathcal{C} \to \mathcal{B} \mid \mathcal{B} \to \mathcal{C} \mid \mathcal{B} \to \mathcal{B} \end{aligned} \tag{1}$$

where $\mathcal{R}$ is a set of relations between $\mathcal{C}$ and $\mathcal{B}$, and | demotes alternative relations.

Abstract systems can be classified into two categories known as the *closed* and *open* systems. Most practical and useful systems in nature are open systems in which there are interactions between the system and its environment. However, in order to develop the theoretical framework of abstract systems, the closed systems in which there is no interaction with the external environment will be modeled first in the following subsection.

The Mathematical Model of Closed Systems

The axiom of the abstract system theory is based on the Object-Attribute-Relation (OAR) model (Wang, 2007b, 2007d; Wang & Wang, 2006c), in which the architecture of a system object O_s can be modeled by a set of attributes A and a set of binary relations R among A and O_s, i.e.:

$$O_s \triangleq (A, R) \tag{2}$$

Encompassing both architectures and behaviors of a system on the basis of Equation 2, an abstract closed system without interactions with the environment can be formally described as follows.

Definition 3. *A closed system $\hat{S}$ on $\mathfrak{U}$ is a 4-tuple, i.e.:*

$$\hat{S} \triangleq (C, R, B, \Omega) \tag{3}$$

where

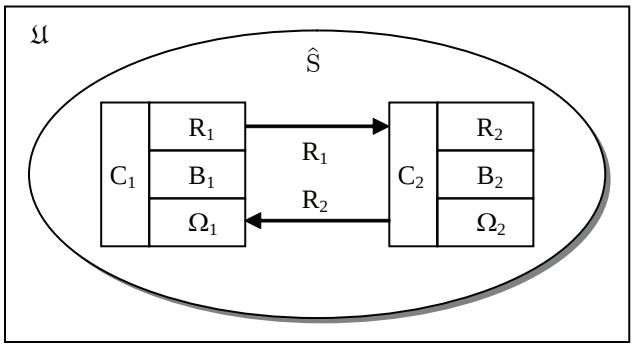
- C is a nonempty set of components of the system, $C = \{c_1, c_2, \ldots, c_n\} \subseteq \wp\mathcal{C} \sqsubseteq \mathfrak{U}$.
- R is a nonempty set of relations between pairs of the components in the system, $R = \{r_1, r_2, \ldots, r_m\} \subseteq C \times C$.
- B is a set of behaviors (or functions), $B = \{b_1, b_2, \ldots, b_p\} \subseteq \wp\mathcal{B} \sqsubseteq \mathfrak{U}$.
- Ω is a set of constraints on the memberships of components, the conditions of relations, and the scopes of behaviors, $\Omega = \{\omega_1, \omega_2, \ldots, \omega_q\}$.

According to Definition 3, a closed system $\hat{S} = (C, R, B, \Omega)$ on $\mathfrak{U}$ can be illustrated in Figure 1.

It is noteworthy that system behaviors B is the most broad set of system actions implemented or embodied on the given layout of the systems, including any kind of system functions, interactions, and communications. This is the major difference that distinguishes an abstract system from other mathematical structures such as a set, lattice, group, or concept (Wang, 2008b).

Lemma 2. *A closed system $\hat{S} = (C, R, B, \Omega)$ is an asymmetric (directed) and reflective system because the relations R in it are constrained by the following rules:*

Figure 1. The abstract model of a closed system



a. *Asymmetric*:

$$\forall a, b \in C \wedge a \neq b \wedge r \in R, r(a, b) \nRightarrow r(b, a) \quad (4)$$

b. *Reflective*:

$$\forall c \in C, r(c, c) \in R \quad (5)$$

Corollary 1. *The maximum number of binary relations n_r between all pairs of the n_c components in a closed system $\widehat{S} = (C, R, B, \Omega)$ can be determined as follows:*

$$n_r = \#R = \#(C \times C) = n_c^{\,2} \quad (6)$$

if all reflective self-relations are not considered, n_r becomes the maximum number of binary relations of fully connected systems, n'_r, i.e.:

$$n'_r = n'_r - n_c = n_c(n_c - 1) \quad (7)$$

Example 1. *According to Corollary 1, the creation of relations in a closed system is solely determined by the number of components possessed in the system. This property can be illustrated in Figure 2, where $\uplus$ denotes a system composition for both closed or open systems, which will be formally explained later. The relations of the three closed systems $\widehat{S_1}$, $\widehat{S_2}$, and $\widehat{S}$ are as follows, observing that all pairwise relations are asymmetric or different:*

$$n_{r_1}(\widehat{S_1}) = n_{c_1}^{\,2} = 3^2 = 9;\ n_{r_1}(\widehat{S_2}) = 2^2 = 4;$$
$$n_r(\widehat{S}) = 5^2 = 25$$

and

$$n'_{r_1}(\widehat{S_1}) = n_{r_1} - n_{c_1} = 9 - 3 = 6;$$
$$n'_{r_2}(\widehat{S_2}) = 4 - 2 = 2;\ n'_r(\widehat{S}) = 25 - 5 = 20$$

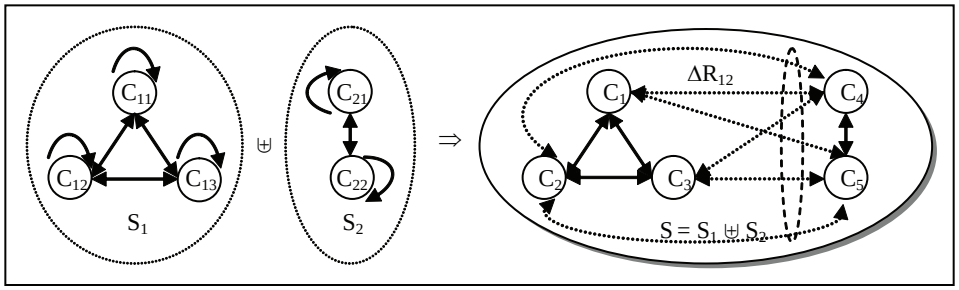
The Mathematical Model of Open Systems

Most practical systems in the real world are not closed. That is, they need to interact with external world known as the *environment* Θ in order to exchange energy, matter, and/or information. Such systems are called "open systems." Typical interactions between an open system and the environment are inputs and outputs.

Contrary to that the relations of a closed system are defined on the Cartesian product of internal components, the set of relations R of an open system needs to be extended to include both internal relations R^c and external (input/output) relations R^i and R^o, i.e.:

$$R = \{R^c \cup R^i \cup R^o\} \quad (8)$$

Figure 2. Creation of relations in open and closed systems



Definition 4. *An open system S on U is a 7-tuple, i.e.:*

$$S \triangleq (C, R, B, \Omega, \Theta),\ R = \{R^c, R^i, R^o\} = (C, R^c, R^i, R^o, B, \Omega, \Theta) \quad (9)$$

where the extensions of entities beyond the closed system as given in Definition 3 are as follows:

- Θ is the environment of S with a nonempty set of components C_Θ outside C, i.e., $\Theta = C_\Theta \sqsubseteq \mathfrak{U}$.
- $R^c = C \times C$ is a set of internal relations.
- $R^i \subseteq C_\Theta \times C$ is a set of external input relations.
- $R^o \subseteq C \times C_\Theta$ is a set of external output relations.

An open system $S = (C, R^c, R^i, R^o, B, \Omega, \Theta)$ can be illustrated in Figure 3.

Lemma 3. *An open system S(C, R^c, R^i, R^o, B, Ω, Θ) on $\mathfrak{U}$ is an asymmetric and reflective system because its relations R^c, R^i, and R^o are constrained by the following rules:*

a. *Internally asymmetric*:

$$\forall a, b \in C \wedge a \neq b \wedge r \in R^c,\ r(a,b) \Rightarrow r(b,a) \quad (10)$$

b *Externally asymmetric*:

$$\forall a \in C \wedge \forall x \in C_\Theta \wedge r \in R^i \vee r \in R^o,\ r(x,a) \Rightarrow r(a,x) \quad (11)$$

c. *Reflective*:

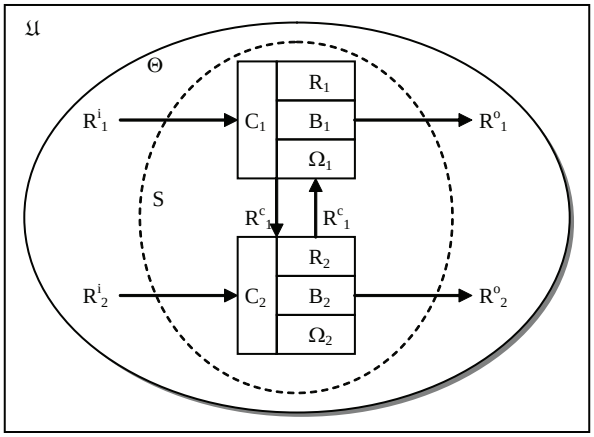
$$\forall c \in C,\ r(c, c) \in R^c \quad (12)$$

Corollary 2. *The maximum number of binary relations n_r and n'_r in an open system S(C, R^c, R^i, R^o, B, Ω, Θ) is determined by the numbers of internal relations R^c as well as external relations R^i and R^o, i.e.:*

$$n_r(S) = \#R^c + \#R^i + \#R^o = {n_c}^2 + 2(n_c \bullet n_{c_\Theta}) = n_c(n_c + 2n_{c_\Theta}) \quad (13)$$

$$n'_r(S) = n_r(S) - n_c = n_c \bullet (n_c + 2n_{c_\Theta}) - n_c = n_c \bullet (n_c + 2n_{c_\Theta} - 1) \quad (14)$$

Figure 3. The abstract model of an open system



Example 2. *According to Corollary 2, the creation of relations in an open system is solely determined by the number of components possessed in the system and its environment. This property can also be illustrated in Figure 2, where S_1 is treated as the open system and S_2 as the environment* Θ*:*

$$n_r(S) = n_c(n_c + 2n_{c_\Theta}) = 3(3 + 2\bullet 2) = 21$$

and

$$n'_r(S) = n_r(S) - n_c = 21 - 3 = 18$$

According to Corollaries 1 and 2, as well as Examples 1 and 2, it is apparent that either a closed or an open system may result in a huge number of relations n_r and the exponential increases of complexity, when the number of components possessed in them is considerably large. Therefore, system algebra is introduced to formally and efficiently manipulate abstract and general systems.

Definition 5. *A system algebra SA on a given universal system environment* $\mathfrak{U}$ *is a triple, i.e.:*

$$SA \triangleq (S, OP, \Theta) = (\{C, R^c, R^i, R^o, B, \Omega\}, \{\bullet_r, \bullet_c\}, \Theta) \quad (15)$$

where OP = {$\bullet_r$, $\bullet_c$}are the sets of relational and compositional operations on abstract systems.

System algebra provides a denotational mathematical means for algebraic manipulations of abstract systems. System algebra can be used to model, specify, and manipulate generic "to be" and "to have" type problems, particularly system architectures and high-level system designs, in computing, software engineering, system engineering, and cognitive informatics. The *relational* and *compositional* operations on abstract systems will be formally described in the following sections.

PROPERTIES OF ABSTRACT SYSTEMS

Taxonomy of Systems

Systems as the most complex entities in the physical and abstract worlds may be classified into various categories according to the key characteristics of their components (*C*), relations (*R*), behaviors (*B*), constraints (Ω), and/or environments (Θ). A summary of the system taxonomy is shown in Table 1, according to Definitions 3 and 4.

Table 1 shows that all types of systems fit the unified framework of system taxonomy. There are hybrid systems that may fall in two or more categories, such as a dynamic nonlinear system and a discrete fuzzy social system. The types of systems may also classified by their magnitudes as described in the following subsection.

Magnitude of Systems

Abstract and real-world systems may be very small or extremely large (Qian et al, 1990; Rosen, 1977). Therefore, a formal model of system magnitudes is needed to classify the size properties of systems and their relationship with other basic system attributes. In order to derive such a model, a set of measures on system sizes, magnitudes, and complexities is introduced next.

Definition 6. *The size of a system S_s is the number of components encompassed in the system, i.e.:*

$$S_s = \#C = n_c \quad (16)$$

Definition 7. *The magnitude of a system M_s is the number of asymmetric binary relations among the n_c components of the system including the reflexive relations, i.e.:*

$$M_s = \#R = n_r = \#(C \times C) = n_c^2 \quad (17)$$

Table 1. Taxonomy of systems

	System	Key Characteristics			
		Components (C)	**Relations (R)**	**Behaviors (B)**	**Environment (Θ)**
1	Concrete	Natural or real entities			
2	Abstract	Mathematical or virtual entities			
3	Physical	Natural entities			
4	Social	Humans			
5	Finite	$\#C \neq \infty$			
6	Infinite	$\#C = \infty$			
7	Closed		$R^i = \varnothing \wedge R^o = \varnothing$		
8	Open		$R^i \neq \varnothing \wedge R^o \neq \varnothing$		
9	Static			Invariable	
10	Dynamic			Variable	
11	Linear			Linear functions	
12	Nonlinear			Nonlinear functions	
13	Continuous			Continuous functions	
14	Discrete			Discrete functions	
15	Precise			Precise functions	
16	Fuzzy			Fuzzy functions	
17	Determinate			Response predictable to same stimulates	
18	Indeterminate			Response unpredictable to same stimulates	
19	White-box	Observable	Transparent	Fully observable	
20	Black-box	Unobservable	Non-transparent	Partially observable	
21	Intelligent			Autonomic	Adaptive
22	Non-intelligent			Imperative	Nonadaptive
23	Maintainable	Fixable		Recoverable	
24	Non-maintainable	Nonfixable		Nonrecoverable	

If all self-reflective relations are ruled out in n_r, the pure number of binary relations M'_s in the given system is determined as follows:

$$M'_s = M_s - n_c = n_c^2 - n_c = n_c(n_c - 1) \quad (18)$$

Lemma 4. *The pure number of binary relations M'_s equals to exactly two times of the number of pairwise combinations among n_c, i.e.:*

$$M'_s = n_c(n_c - 1) = 2 \bullet \frac{n_c(n_c - 1)}{2} = 2 \bullet \mathrm{C}_{n_c}^2 \quad (19)$$

where the factor 2 represents the asymmetric binary relation r, i.e., arb ≠ bra.

The magnitude of a system determines its complexity. The complexities of systems can be classified based on if they are fully or partially connected. The former is the theoretical upper-bound complexity of systems in which all components are potentially interconnected with each other in all *n*-nary ways, $1 \le n \le n_c = \#C$. The latter is the more typical complexity of systems where components are only pairwisely connected.

Definition 8. *The complexity of a fully connected system C_{max} is a closure of all possible n-nary relations R*, $1 \le n \le n_c$, among all components of the given system $n_c = \#C$, i.e.:*

$$C_{max} = R^* = 2\sum_{k=0}^{n_r} \mathrm{C}_{n_r}^k \approx 2 \bullet 2^{n_r} = 2^{n_r+1} = 2^{n_c^2+1} = 2^{M_s+1} \quad (20)$$

where C_{max} is also called the maximum complexity of systems.

According to Definition 8, the closure of all possible *n*-nary relations *R** may easily result in an extremely huge degree of complexity for a system with few components. For example, when $n_c = 10$, $C_{max} = 2^{101}$. This explains why most of the real-world systems are really too hard to be modeled and handled in conventional models and techniques.

It is noteworthy that almost all functioning systems are partially connected, because a fully connected system may not represent or provide anything meaningful. Therefore, the complexity of partially connected systems can be simplified as follows on the basis of Definition 8.

Definition 9. *The complexity of a partially connected system C_r is determined by the number of asymmetric binary relations M'_s of the system, i.e.:*

$$C_r = M'_s = 2 \bullet \mathrm{C}_{n_c}^2 = n_c(n_c - 1) \quad (21)$$

where C_r can be referred to the relational complexity of systems.

The extent of system magnitudes (Wang, 2006d, 2007c) can be classified at seven levels known as the *empty, small, medium, large, giant, immense,* and *infinite* systems from the bottom up. A summary of the relationships between system magnitudes, sizes, internal relations, and complexities can be described in the *system magnitude model,* shown in Table 2.

Table 2 indicates that the complexity of a small system may easily be out of control of human cognitive manageability. This leads to the following theorem.

Theorem 1. *The holism complexity of systems states that within the 7-level scale of system magnitudes, known as the empty, small, medium, large, giant, immense, and infinite systems, almost all systems are too complicated to be cognitively understood or mentally handled as a whole, except small systems or those that can be decomposed into small systems.*

According to Theorem 1, the basic principle for dealing with complicated systems is system

Table 2. The system magnitude model

Level	Category	Size of systems ($S_s = n_c$)	Magnitude of systems ($M_s = n_r = n_c^2$)	Relational complexity of systems ($C_r = n_c(n_c - 1)$)	Maximum complexity of systems ($C_{max} = 2^{n_c^2}$)
1	The empty system ($\mathfrak{O}$)	0	0	0	-
2	Small system	[1, 10]	[1, 10^2]	[0, 90]	[2, 2^{100}]
3	Medium system	(10, 10^2]	(10^2, 10^4]	(90, $0.99 \bullet 10^4$]	(2^{100}, $2^{10,000}$]
4	Large system	(10^2, 10^3]	(10^4, 10^6]	($0.99 \bullet 10^4$, $0.999 \bullet 10^6$]	∞
5	Giant system	(10^3, 10^4]	(10^6, 10^8]	($0.999 \bullet 10^6$, $0.9999 \bullet 10^8$]	∞
6	Immense system	(10^4, 10^5]	(10^8, 10^{10}]	($0.9999 \bullet 10^8$, $0.99999 \bullet 10^{10}$]	∞
7	The infinite system ($\mathfrak{U}$)	∞	∞	∞	∞

decomposition or modularity, in which the complexity of a lower level subsystem must be small enough to be cognitively manageable. Details of system decomposition theories and the art of system architectures will be developed in the following sections.

RELATIONAL OPERATIONS ON SYSTEMS

The relational operations of abstract systems are static and comparative operations that do not change the systems involved. The relational operations on abstract systems are described below.

Lemma 5. *The relational operations $\bullet_r$ in system algebra encompasses 6 comparative operators for manipulating the algebraic relations between abstract systems, i.e.:*

$$\bullet_r \triangleq \{\nleftrightarrow, \leftrightarrow, \prod, =, \sqsubset, \sqsupset\} \tag{22}$$

where the relational operators stand for independent, related, overlapped, equivalent, subsystem, and supersystem, respectively.

Algebraic Relations of Closed Systems

Relationships between two closed systems can be independent, equivalent, being subsystem, and being super system. The four relational operations of closed systems are defined as follows.

Definition 10. *Two closed systems $\widehat{S_1}$ and $\widehat{S_2}$ are independent, denoted by $\nleftrightarrow$, if both their component sets and relation sets are disjoint, i.e.:*

$$\widehat{S}_1(C_1, R_1, B_1, \Omega_1) \nleftrightarrow (\widehat{S}_2\ C_2, R_2, B_2, \Omega_2)$$
$$\triangleq C_1 \cap C_2 = \varnothing \wedge R_1 \cap R_2 = \varnothing \quad (23)$$

Definition 11. *Two closed systems $\widehat{S_1}$ and $\widehat{S_2}$ are equivalent, denoted by =, if all sets of components, relations, behaviors, and constraints are identical, i.e.:*

$$\widehat{S}_1(C_1, R_1, B_1, \Omega_1) = \widehat{S}_2(C_2, R_2, B_2, \Omega_2)$$
$$\triangleq C_1 = C_2 \wedge R_1 = R_2 \wedge B_1 = B_2 \wedge \Omega_1 = \Omega_2 \quad (24)$$

Definition 12. *A subsystem $\widehat{S}'$ is a system that is implicated in another system $\widehat{S}$, denoted by $\sqsubseteq$, i.e.:*

$$\widehat{S}'(C, R, B, \Omega) \sqsubseteq \widehat{S}(C, R, B, \Omega)$$
$$\triangleq C' \subseteq C \wedge R' \subseteq R \wedge B' \subseteq B \wedge \Omega' \subseteq \Omega \quad (25)$$

The aforementioned definition indicates that a subsystem of a closed system is a coherent part with all integrated components, internal/input/output relations, behaviors, constraints, and the environment.

Definition 13. *A supersystem $\widehat{S}$ is a system that consists of one or more subsystems S', denoted by $\sqsupseteq$, i.e.:*

$$\widehat{S}(C, R, B, \Omega) \sqsupseteq \widehat{S}'(C', R', B', \Omega')$$
$$\triangleq C' \subseteq C \wedge R' \subseteq R \wedge B' \subseteq B \wedge \Omega' \subseteq \Omega \quad (26)$$

Algebraic Relations of Open Systems

Relationships between two open systems can be independent, overlapped, related, equivalent, being subsystem, and being supersystem. The six compositional operations of open systems are defined as follows.

Definition 14. *Two open systems S_1 and S_2 are independent, denoted by $\nleftrightarrow$, if both their component sets and external relation sets are disjoint, i.e.:*

$$S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) \nleftrightarrow$$
$$S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2)$$
$$\triangleq C_1 \cap C_2 = \varnothing \wedge R_1^i \cap R_2^i =$$
$$\varnothing \wedge R_1^o \cap R_2^o = \varnothing \quad (27)$$

Definition 15. *Two open systems S_1 and S_2 are overlapped, denoted by Π, if their component sets are joint, i.e.:*

$$S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) \prod$$
$$S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2)$$
$$\triangleq C_1 \cap C_2 \neq \varnothing \quad (28)$$

Definition 16. *Two open systems S_1 and S_2 are related, denoted by $\leftrightarrow$, if either the sets of their input relations or output relations are overlapped, i.e.:*

$$S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) \leftrightarrow$$
$$S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2)$$
$$\triangleq R_1^i \cap (R_2^0)^{-1} \neq \varnothing \vee R_2^i \cap (R_1^0)^{-1} \neq \varnothing \quad (29)$$

where $(R^0_1)^{-1}$ or $(R^0_2)^{-1}$ denotes an inverse relation, i.e., $\forall a \in C_1 \wedge b \in C_2$, $r(a, b) \in R^0_1 \Rightarrow r(b, a) \in (R^i_2) = (R^0_1)^{-1}$.

It is noteworthy that, by definition, there is no closed system that is related or overlapped.

Definition 17. *Two open systems S_1 and S_2 are equivalent, denoted by =, if all sets of components, relations, behaviors, constraints, and environments are identical, i.e.:*

$$S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) =$$
$$S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2) \triangleq$$
$$C_1 = C_2 \wedge R_1^c = R_2^c \wedge R_1^i = R_2^i \wedge R_1^o =$$
$$R_2^o \wedge B_1 = B_2 \wedge \Omega_1 = \Omega_2 \wedge \Theta_1 = \Theta_2 \quad (30)$$

Definition 18. *A subsystem S' is a system that is implicated in another system S, denoted by* $\sqsubseteq$, *i.e.:*

$$S'(C', R^{c'}, R^{i'}, R^{o'}, B', \Omega', \Theta') \sqsubseteq S(C, R^c, R^i, R^o, B, \Omega, \Theta) \triangleq C' \subseteq C \wedge R^{c'} \subseteq R^c \wedge R^{i'} \subseteq R^i \wedge R^{o'} \subseteq R^o \wedge B' \subseteq B \wedge \Omega' \subseteq \Omega \wedge \Theta' = \Theta \quad (31)$$

The above definition indicates that a subsystem of an open system is a coherent part of it with all integrated components, internal/input/output relations, behaviors, and constraints. However, they share the same environment.

Definition 19. *A supersystem S is a system that consists of one or more subsystems S', denoted by* $\sqsupseteq$, *i.e.:*

$$S(C, R^c, R^i, R^o, B, \Omega, \Theta) \sqsupseteq S'(C', R^{c'}, R^{i'}, R^{o'}, B', \Omega', \Theta') \triangleq C' \subseteq C \wedge R^{c'} \subseteq R^c \wedge R^{i'} \subseteq R^i \wedge R^{o'} \subseteq R^o \wedge B' \subseteq B \wedge \Omega' \subseteq \Omega \wedge \Theta' = \Theta \quad (32)$$

Relations between Open and Closed Systems

Although, the previous subsections analyze the relations of closed and open systems separately, it is noteworthy that closed and open systems are transformable, when the environment of them is treated as a supersystem as well. This notion can be described in the following theorem and corollaries on the basis of Definitions 3 and 4.

Theorem 2. *The equivalence between open and closed systems states that an open system S is equivalent to a closed system* $\widehat{S}$, *or vice versa, when its environment* Θ_S *or* $\Theta_{\widehat{S}}$ *is conjoined, respectively, i.e.:*

$$\begin{cases} \widehat{S} = \mathrm{S} \sqcup \Theta_{\mathrm{S}} \\ \mathrm{S} = \widehat{S} \sqcup \Theta_{\widehat{\mathrm{S}}} \end{cases} \quad (33)$$

Theorem 2 can be proved by observing the embedded relation of close and open systems as illustrated in Figure 3. According to Theorem 2, the following properties of equivalence between closed and open systems can be derived.

Corollary 3. *Any subsystem* $\widehat{S_k}$ *of a closed system* $\widehat{S}$ *is an open system, i.e.:*

$$\forall \widehat{S_k} \subseteq \widehat{S} \Rightarrow R^i{}_k \neq \varnothing \wedge R^o{}_k \neq \varnothing \wedge \Theta_k = C_s \neq \varnothing \quad (34)$$

Corollary 4. *Any supersystem S of a given set of n open systems* S_k, *conjoining with their environments* Θ_k, $1 \le k \le n$, *is a closed systems, i.e.:*

$$\forall S_k, \forall \Theta_k, \ \widehat{S} = \mathop{R}_{k=1}^{n} (S_k \sqcup \Theta_k) \Rightarrow R_S{}^i = \varnothing \wedge R_S{}^o = \varnothing \wedge \Theta_S = \varnothing \quad (35)$$

where $\mathop{R}_{k=1}^{n} s_k$ is an operator known as the *big-R notation* (Wang, 2002, 2008a, 2008b, 2008c) that denotes a repetitive behavior or recurrent structure as defined in real-time process algebra (RTPA) (Wang, 2002, 2003, 2006a, 2006c, 2007a, 2007c, 2008a, 2008d).

COMPOSITIONAL OPERATIONS ON SYSTEMS

This section describes how abstract systems and their relations as modeled in previous sections may be manipulated by an algebraic system. The compositional operations of system algebra are dynamic and integrative operations that manipulate all systems involved in parallel. Compositional operations on abstract systems provide a set of fundamental mathematical means to construct complex systems on the basis of simple ones or to derive new systems on the basis of exiting ones.

Lemma 6. *The compositional operations* $\bullet_c$ *in system algebra encompasses 9 associative opera-*

tors for manipulating the algebraic compositions among abstract systems, i.e.:

$$\bullet_c \triangleq \{\Rightarrow, \underset{-}{\Rightarrow}, \underset{+}{\Rightarrow}, \underset{\sim}{\Rightarrow}, \boxminus, \uplus, \pitchfork, \Leftarrow, \vdash\} \tag{36}$$

where the compositional operators stand for system inheritance, tailoring, extension, substitute, difference, composition, decomposition, aggregation, and specification, respectively.

System Inheritance

Definition 20. *The inheritance of a closed system $\widehat{S_2}$ from a given system $\widehat{S_1}$, denoted by $\Rightarrow$, is the creation of the new system $\widehat{S_2}$ by reproducing $\widehat{S_1}$, i.e.:*

$$\begin{aligned}&\widehat{S_1}(C_1, R_1, B_1, \Omega_1) \Rightarrow \widehat{S_2}(C_2, R_2, B_2, \Omega_2)\\ &\triangleq \widehat{S_2}(C_2, R_2, B_2, \Omega_2 \mid C_2 = C_1, R_2 = R_1, B_2 = B_1, \Omega_2 = \Omega_1)\end{aligned} \tag{37}$$

where $\widehat{S_1}$ is called the parent system, $\widehat{S_2}$ the child system.

Similarly, the inheritance of open systems can be defined as follows.

Box 1.

$$\begin{aligned}&S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) \Rightarrow S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2)\\ &\triangleq S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2 \mid C_2 = C_1, R_2^c = R_1^c,\\ &\quad R^i{}_2 = R^i{}_1 \cup \{C_1 \times C_2\}, R^o{}_2 = R^o{}_1 \cup \{C_2 \times C_1\},\\ &\quad B_2 = B_1\ \Omega_2 = \Omega_1, \Theta_2 = \Theta_1)\\ &\|\, S_1(C_1, R_1^c, R_1^{i'}, R_1^{o'}, B_1, \Omega_1, \Theta'_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{C_2 \times C_1\},\\ &\quad R^{o'}{}_1 = R^o{}_1 \cup \{C_1 \times C_2\}, \Theta'{}_1 = \Theta_1 \cup C_2)\end{aligned} \tag{38}$$

Box 2.

$$\begin{aligned}&\mathop{\|}_{i=1}^{n} S_i \Rightarrow (S\ C, R^c, R^i, R^o, B, \Omega, \Theta)\\ &\triangleq S(C, R^c, R^i, R^o, B, \Omega, \Theta \mid C = \bigcup_{i=1}^{n} C_i, R^c = \bigcup_{i=1}^{n} R^c{}_i, R^i = \bigcup_{i=1}^{n} R^i{}_i \cup \{\mathop{\|}_{i=1}^{n}(C_i \times C)\},\\ &\quad R^o = \bigcup_{i=1}^{n} R^o{}_i \cup \{\mathop{\|}_{i=1}^{n}(C \times C_i)\}, B = \bigcup_{i=1}^{n} B_i, \Omega = \bigcup_{i=1}^{n} \Omega_i, \Theta = \bigcup_{i=1}^{n} \Theta_i)\\ &\|\, \mathop{\|}_{i=1}^{n} S_i(C_i, R_i^c, R_i^i, R_i^o, B_i, \Omega_i, \Theta'_i \mid R^{i'}{}_i = R^i{}_i \cup \{C \times C_i\},\\ &\quad R^{o'}{}_i = R^o{}_i \cup \{C_i \times C\}, \Theta'{}_i = \Theta_i \cup C)\end{aligned} \tag{39}$$

Definition 21. *The inheritance of an open system S_2 from the parent system S_1, denoted by $\Rightarrow$, is the creation of the new system S_2 by reproducing S_1, and the establishment of new associations between them, see Box 1, where $||$ denotes that an open system inheritance creates new associations between S_1 and S_2 in parallel via (R^o_1, R^i_2) and (R^o_2, R^i_1).*

Definition 22. *The multiple inheritance of an open system S from n parent systems $S_1, S_2, \ldots, S_n$, denoted by $\Rightarrow$, is an inheritance that creates the new system S via a set of n conjoint systems and establishes new associations among them, see Box 2.*

System Tailoring

Definition 23. *The tailoring of a closed system $\widehat{S_2}$ from the parent system $\widehat{S_1}$, denoted by $\bar{\Rightarrow}$, is a special system inheritance that creates the new system $\widehat{S_2}$ based on $\widehat{S_1}$ with the removal of specific subsets of components C', behaviors B', and constraints Ω', see Box 3.*

Similarly, the tailoring of open systems can be defined as follows.

Definition 24. *The tailoring of an open system S_2 from the parent system S_1, denoted by $\bar{\Rightarrow}$, is a special system inheritance that creates the new system S_2 based on S_1 with the removal of specific subsets of components C', behaviors B', and constraints Ω'; and at the same time, it establishes new associations between them, see Box 4.*

System Extension

Definition 25. *The extension of a closed system $\widehat{S_2}$ from the parent system $\widehat{S_1}$, denoted by $\overset{+}{\Rightarrow}$, is a special*

Box 3.

$$\begin{aligned}&\widehat{S_1}(C_1, R_1, B_1, \Omega_1) \bar{\Rightarrow} \widehat{S_2}(C_2, R_2, B_2, \Omega_2)\\ &\triangleq \widehat{S_2}(C_2, R_2, B_2, \Omega_2 \mid C_2 = C_1 \setminus C', {R_2}^c = {R_1}^c \setminus \{C_1 \times C'\}\\ &\qquad B_2 = B_1 \setminus B', \Omega_2 = \Omega_1 \setminus \Omega')\end{aligned} \tag{40}$$

Box 4.

$$\begin{aligned}&S_1(C_1, {R_1}^c, {R_1}^i, {R_1}^o, B_1, \Omega_1, \Theta_1) \bar{\Rightarrow} S_2(C_2, {R_2}^c, {R_2}^i, {R_2}^o, B_2, \Omega_2, \Theta_2)\\ &\triangleq S_2(C_2, {R_2}^c, {R_2}^i, {R_2}^o, B_2, \Omega_2, \Theta_2 \mid C_2 = C_1 \setminus C', {R_2}^c = {R_1}^c \setminus \{C_1 \times C'\},\\ &\qquad {R^i}_2 = {R^i}_1 \cup \{C_1 \times C_2\}, {R^o}_2 = {R^o}_1 \cup \{C_2 \times C_1\},\\ &\qquad B_2 = B_1 \setminus B', \Omega_2 = \Omega_1 \setminus \Omega', \Theta_2 = \Theta_1)\\ &\mid\mid S_1(C_1, {R_1}^c, {R_1}^{i'}, {R_1}^{o'}, B_1, \Omega_1, {\Theta'}_1 \mid {R^{i'}}_1 = {R^i}_1 \cup \{C_2 \times C_1\},\\ &\qquad {R^{o'}}_1 = {R^o}_1 \cup \{C_1 \times C_2\}, {\Theta'}_1 = \Theta_1 \cup C_2)\end{aligned} \tag{41}$$

system inheritance that creates the new system $\widehat{S_2}$ based $\widehat{S_1}$ with additional subsets of components C′, behaviors B′, and constraints Ω′, see Box 5.

Similarly, the extension of open systems can be defined as follows.

Definition 26. *The extension of an open system S_2 from the parent system S_1, denoted by $\overset{+}{\Rightarrow}$, is a special system inheritance that creates the new system S_2 based S_1 with additional subsets of components C′, behaviors B′, and constraints Ω′; and at the same time, it establishes new associations between the two systems, see Box 6.*

System Substitution

Definition 27. *The substitute of a closed system $\widehat{S_2}$ from the parent system $\widehat{S_1}$, denoted by $\tilde{\Rightarrow}$, is a flexible system inheritance that creates the new system $\widehat{S_2}$ based on $\widehat{S_1}$ with the new subsets of components C'_{c2}, behaviors B'_{c2}, and constraints Ω'_{c2} to replace the corresponding inherited ones C'_{c1}, B'_{c1}, and Ω'_{c2} that share the same identifiers, see Box 7. Where $C'_{S1} \subset C_1 \wedge C'_{S2} \subset C_2 \wedge \#C'_{S1} = \#C'_{S2}$; $B'_{S1} \subset B_1 \wedge B'_{S2} \subset B_2 \wedge \#B'_{S1} = \#B'_{S2}$; and $\Omega'_{S1} \subset \Omega_1 \wedge \Omega'_{S2} \subset \Omega_2 \wedge \#\Omega'_{S1} = \#\Omega'_{S2}$. Similarly, the substitute of open systems can be defined as follows.*

Box 5.

$$\begin{aligned}
&\widehat{S_1}(C_1, R_1, B_1, \Omega_1) \overset{+}{\Rightarrow} \widehat{S_2}(C_2, R_2, B_2, \Omega_2) \\
&\triangleq \widehat{S_2}(C_2, R_2, B_2, \Omega_2 \mid C_2 = C_1 \cup C', {R_2}^c = {R_1}^c \cup \{C_1 \times C'\} \\
&\quad B_2 = B_1 \cup B', \Omega_2 = \Omega_1 \cup \Omega')
\end{aligned} \tag{42}$$

Box 6.

$$\begin{aligned}
&S_1(C_1, {R_1}^c, {R_1}^i, {R_1}^o, B_1, \Omega_1, \Theta_1) \overset{+}{\Rightarrow} (S_2\ C_2, {R_2}^c, {R_2}^i, {R_2}^o, B_2, \Omega_2, \Theta_2) \\
&\triangleq\ S_2(C_2, {R_2}^c, {R_2}^i, {R_2}^o, B_2, \Omega_2, \Theta_2 \mid C_2 = C_1 \cup C', {R_2}^c = {R_1}^c \cup \{C_1 \times C'\} \\
&\quad {R^i}_2 = {R^i}_1 \cup \{C_1 \times C_2\}, {R^o}_2 = {R^o}_1 \cup \{C_2 \times C_1\}, \\
&\quad B_2 = B_1 \cup B', \Omega_2 = \Omega_1 \cup \Omega', \Theta_2 = \Theta_1) \\
&\|\ S_1(C_1, {R_1}^c, {R_1}^{i'}, {R_1}^{o'}, B_1, \Omega_1, {\Theta'}_1 \mid {R^{i'}}_1 = {R^i}_1 \cup \{C_2 \times C_1\}, \\
&\quad {R^{o'}}_1 = {R^o}_1 \cup \{C_1 \times C_2\}, {\Theta'}_1 = \Theta_1 \cup C_2)
\end{aligned} \tag{43}$$

Definition 28. *The substitute of an open system S_2 from the parent system S_1, denoted by $\tilde{\Rightarrow}$, is a flexible system inheritance that creates the new system S_2 based on S_1 with the new subsets of components C'_{c2}, behaviors B'_{c2}, and constraints attributes Ω'_{c2} to replace the corresponding inherited ones C'_{c1}, B'_{c1}, and Ω'_{c2} that share the same identifiers; and at the same time, it establishes new associations between the two concepts, see Box 8. Where $C'_{S1} \subset C_1 \wedge C'_{S2} \subset C_2 \wedge \#C'_{S1} = \#C'_{S2}$; $B'_{S1} \subset B_1 \wedge B'_{S2} \subset B_2 \wedge \#B'_{S1} = \#B'_{S2}$; and $\Omega'_{S1} \subset \Omega_1 \wedge \Omega'_{S2} \subset \Omega_2 \wedge \#\Omega'_{S1} = \#\Omega'_{S2}$.*

Binary tailoring, extension, and substitution can also be extended to corresponding *n*-nary operations, similar to that of inheritance as given in Definitions 22.

System Composition

As a preparation to describe the important property of system relations, the mathematical calculus of incremental union between sets of relations is introduced below (Wang, 2006c, 2007c).

Definition 29. *An incremental union of two sets of relations R_1 and R_2, denoted by $\boxplus$, are a union of R_1 and R_2 plus a newly generated incremental set of relations ΔR_{12}, i.e.:*

$$R_1 \boxplus R_2 \triangleq R_1 \cup R_2 \cup \Delta R_{12} \tag{46}$$

where

$\Delta R_{12} \not\subseteq R_1 \wedge \Delta R_{12} \not\subseteq R_2$, *but* $\Delta R_{12} \subseteq R_1 \boxplus R_2$.

Box 7.

$$\begin{aligned}
&\widehat{S}_1(C_1, R_1, B_1, \Omega_1) \tilde{\Rightarrow} \widehat{S}_2(C_2, R_2, B_2, \Omega_2) \\
&\triangleq \widehat{S}_2(C_2, R_2, B_2, \Omega_2 \mid C_2 = (C_1 \setminus C'_{S_1}) \cup C'_{S_2}, R_2^c = (R_1^c \setminus (C_1 \times C'_{S_1}))\ (\cup\ C_1 \times C'_{S_2}), \\
&\qquad (B_2 = B_1 \setminus B'_{S_1}) \cup B'_{S_2}, \Omega_2 = (\Omega_1 \setminus \Omega'_{S_1}) \cup \Omega'_{S_2})
\end{aligned} \tag{44}$$

Box 8.

$$\begin{aligned}
&S_1(C_1, R_1^c, R_1^i, R_1^o, B_1, \Omega_1, \Theta_1) \tilde{\Rightarrow} S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2) \\
&\triangleq S_2(C_2, R_2^c, R_2^i, R_2^o, B_2, \Omega_2, \Theta_2 \mid C_2 = (C_1 \setminus C'_{S_1}) \cup C'_{S_2}, \\
&\qquad R_2^c = (R_1^c \setminus \{C_1 \times C'_{S_1}\})\ \cup \{C_1 \times C'_{S_2}\}, R^i{}_2 = R^i{}_1 \cup \{C_1 \times C_2\}, \\
&\qquad R^o{}_2 = R^o{}_1 \cup \{C_2 \times C_1\}, B_2 = (B_1 \setminus B'_{S_1}) \cup B'_{S_2}, \Omega_2 = (\Omega_1 \setminus \Omega'_{S_1}) \cup \Omega'_{S_2}, \Theta_2 = \Theta_1) \\
&\mid\mid S_1(C_1, R_1^c, R_1^{i'}, R_1^{o'}, B_1, \Omega_1, \Theta'_1 \mid R^{i'}{}_1 = R^i{}_1 \cup \{C_2 \times C_1\}, R^{o'}{}_1 = R^o{}_1 \cup \{C_1 \times C_2\}, \\
&\qquad \Theta'_1 = \Theta_1 \cup C_2)
\end{aligned} \tag{45}$$

The number of the incremental relations ΔR_{12} generated in the incremental union can be determined as stated in the following theorem.

Theorem 3. *The maximum number of newly gained relations ΔR_{12} obtained during the incremental union of two sets of relations is a product of the numbers of elements of the two sets $\#C_1$ and $\#C_2$, i.e.:*

$$\Delta R_{12} = 2(\#C_1 \bullet \#C_2) = 2n_{c_1} n_{c_2}) \tag{47}$$

Proof: *Because ΔR_{12} is the difference between the relations of the newly generated entire system $\#R$ and those of the independent systems $\#R_1$ and $\#R_2$, according to Corollaries 1 and 2, Theorem 3 is proved by the following inference process:*

$$\begin{aligned}\Delta R_{12} &= \#R - (\#R_1 + \#R_2) \\ &= (n_{c_1} + n_{c_2})^2 - ({n_{c_1}}^2 + {n_{c_2}}^2) \\ &= ({n_{c_1}}^2 + 2n_{c_1} n_{c_2} + {n_{c_2}}^2) - ({n_{c_1}}^2 + {n_{c_2}}^2) \\ &= 2n_{c_1} n_{c_2}\end{aligned} \tag{48}$$

The incremental union of relations reveals an important property of systems, which indicates that the merge of two systems results in new relations, behaviors, functions, and/or constraints that are not belong to any original individual subsystems. Theorem 3 can be used to predict the maximum numbers of newly established relations, behaviors, and/or constraints in a composition of two systems. According to Theorem 3, the maximum *incremental system gain* equals to the number of by-directly interconnection between all components in both S_1 and S_2, i.e., $2(\#C_1 \bullet \#C_2)$.

Definition 29 can be extended to *n*-nary incremental unions for multiple sets of relations.

Definition 30. *The n-nary incremental union for multiple sets of relations,*

$$\biguplus_{i=1}^{n} R_i,$$

is a series of cumulative binary incremental unions as in Box 9.

Based on the calculus of incremental union of sets of relations, system compositions can be defined as follows.

Definition 31. *The composition of two closed systems $\widehat{S_1}$ and $\widehat{S_2}$, denoted by $\uplus$, results in a super system $\widehat{S}$ that is formed by union of sets of components and environments, as well as incremental union of sets of relations, behaviors, and constraints, respectively, see Box 10.*

Theorem 4. *The system fusion principle states that new relations ΔR_{12}, new behaviors (functions) ΔB_{12}, and new constraints $\Delta\Omega_{12}$, generated in system compositions are solely a property of the new super system $\widehat{S}$, but not belong to any of the independent subsystems, i.e.:*

Box 9.

$$\begin{aligned}\biguplus_{i=1}^{n} R_i &\triangleq \mathop{\mathbf{R}}_{i=1}^{n-1}(R_i \cup R_j \cup \Delta R_{ij}),\ j = i+1 \\ &= (...((R_1 \cup R_2 \cup \Delta R_{12}) \cup R_3 \cup \Delta R_{2'3})\ \cup ...) \cup R_n \cup \Delta R_{n-1',n}\end{aligned} \tag{49}$$

Box 10.

$$\begin{aligned}
&\widehat{S_1}(C_1, R_1, B_1, \Omega_1) \uplus \widehat{S_2}(C_2, R_2, B_2, \Omega_2) \\
&\triangleq \widehat{S}(C, R, B, \Omega \mid C = C_1 \cup C_2, R = R_1 \boxplus R_2, B = B_1 \boxplus B_2, \Omega = \Omega_1 \boxplus \Omega_2) \\
&= \widehat{S}(C, R, B, \Omega \mid C = C_1 \cup C_2, R = R_1 \cup R_2 \cup \Delta R_{12}, \\
&\qquad\qquad B = B_1 \cup B_2 \cup \Delta B_{12},\ \Omega = \Omega_1 \cup \Omega_2 \cup \Delta\Omega_{12})
\end{aligned} \tag{50}$$

$$\Delta R_{12} \in \widehat{S} \wedge (\Delta R_{12} \notin \widehat{S_1} \wedge \Delta R_{12} \notin \widehat{S_2}) \tag{51.a}$$

$$\Delta B_{12} \in \widehat{S} \wedge (\Delta B_{12} \notin \widehat{S_1} \wedge \Delta B_{12} \notin \widehat{S_2}) \tag{51.b}$$

$$\Delta\Omega_{12} \in \widehat{S} \wedge (\Delta\Omega_{12} \notin \widehat{S_1} \wedge \Delta\Omega_{12} \notin \widehat{S_2}) \tag{51.c}$$

where E denote a membership relation of a given set in a system.

The discovery in Theorems 3 and 4 reveal that the nature of system utilities can be rigorously explained as the newly gained relations ΔR_{12}, as well as behaviors ΔB_{12} and constraints $\Delta\Omega_{12}$, during the composition of two or more systems. The empirical awareness of this key system property has been intuitively or qualitatively described in the literature of system science (Ellis & Fred, 1962; G. J. Klir, 1992). However, Theorems 3 and 4 are the first mathematical explanation of the mechanism of system gains during system compositions (Wang, 2006b, 2007c).

More generally, Definition 31 and Theorem 4 can be extended to open systems.

Definition 32. *The composition of two open systems S_1 and S_2, denoted by $\uplus$, results in a super system S that is formed by simple conjunctions both of sets of components and environments, as well as incremental unions of sets of relations, behaviors, and constraints, respectively, see Box 11.*

The operation of open system composition is illustrated in Figure 4, where the generation of the new relations ΔR^c_1 and ΔR^c_2 in S after the composition of S_1 and S_2 can be observed.

System compositions as modeled in Definitions 31 and 32 can be extended to *n*-nary compositions as follows.

Definition 33. *The composition of multiple open systems, denoted by*

$$\biguplus_{i=1}^{n} S_i,$$

is an iterative integration of a pair of systems, which cumulatively creates the new supersystem S, see Box 12.

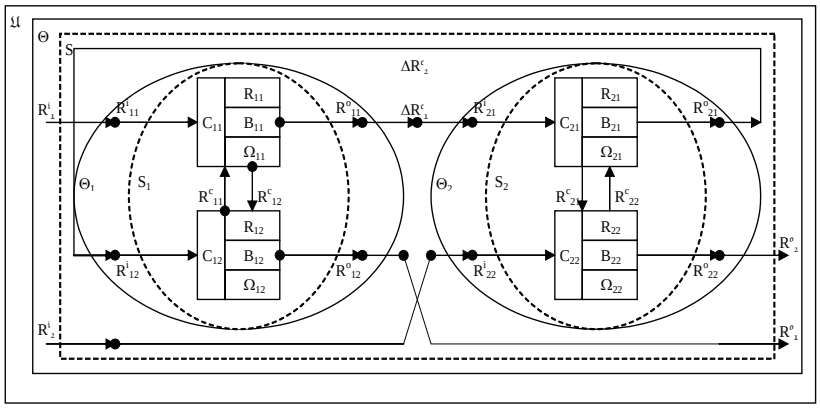
The composition of multiple closed systems is similar to Definition 33, which can be tailored from Equation 53.

System composition at the top level is a complicated algebraic operation that integrates two or

Box 11.

$$
\begin{aligned}
& S_1(C_1, R_1^{\,c}, R_1^{\,i}, R_1^{\,o}, B_1, \Omega_1, \Theta_1) \sqcup S_2(C_2, R_2^{\,c}, R_2^{\,i}, R_2^{\,o}, B_2, \Omega_2, \Theta_2) \\
& \triangleq S(C, R^c, R^i, R^o, B, \Omega, \Theta) \mid C = C_1 \cup C_2, R^c = R_1^{\,c} \boxplus R_2^{\,c}, R^i = R_1^{\,i} \boxplus R_2^{\,i}, \\
& \quad R^o = R_1^{\,o} \boxplus R_2^{\,o}, B = B_1 \boxplus B_2, \Omega = \Omega_1 \boxplus \Omega_2, \Theta = \Theta_1 \cup \Theta_2) \\
& \| \underset{i=1}{\overset{2}{R}} S_i(C_i, R_i^{\,c}, R_i^{\,i'}, R_i^{\,o'}, B_i, \Omega_i, \Theta'_i \mid R^{i'}{}_i = R^i{}_i \cup \{C \times C_i\}, \\
& \quad R^{o'}{}_i = R^o{}_i \cup \{C_i \times C\}, \Theta'{}_i = \Theta_i \cup C) \\
= \; & S(C, R^c, R^i, R^o, B, \Omega\ \Theta \mid C = C_1 \cup C_2, R^c = R_1^{\,c} \cup R_2^{\,c} \cup \Delta R_{12}{}^c, \\
& \quad R^i = R_1^{\,i} \cup R_2^{\,i} \cup \Delta R_{12}{}^i, R^o = R_1^{\,o} \cup R_2^{\,o} \cup \Delta R_{12}{}^o, \\
& \quad B = B_1 \cup B_2 \cup \Delta B_{12}, \Omega = \Omega_1 \cup \Omega_2 \cup \Delta\Omega_{12}, \Theta = \Theta_1 \cup \Theta_2) \\
& \| \underset{i=1}{\overset{2}{R}} S_i(C_i, R_i^{\,c}, R_i^{\,i'}, R_i^{\,o'}, B_i, \Omega_i, \Theta'_i \mid R^{i'}{}_i = R^i{}_i \cup \{C \times C_i\}, \\
& \quad R^{o'}{}_i = R^o{}_i \cup \{C_i \times C\}, \Theta'{}_i = \Theta_i \cup C)
\end{aligned}
\tag{52}
$$

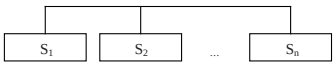
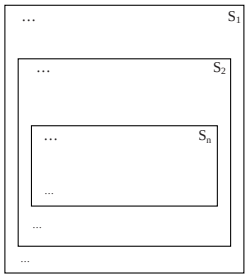
Figure 4. The composition of two open systems



Box 12.

$$
\begin{aligned}
& S(C, R^c, R^i, R^o, B, \Omega, \Theta) \triangleq \underset{i=1}{\overset{n}{\biguplus}} S_i(C_i, R_i^{\,c}, R_i^{\,i}, R_i^{\,o}, B_i, \Omega_i, \Theta_i) \\
& = S(C, R^c, R^i, R^o, B, \Omega, \Theta \mid C = \bigcup_{i=1}^{n} C_i, R^c = \underset{i=1}{\overset{n}{\boxplus}} R^c{}_i, R^i = \underset{i=1}{\overset{n}{\boxplus}} R^i{}_i, \\
& \quad R^o = \underset{i=1}{\overset{n}{\boxplus}} R^o{}_i, B = \underset{i=1}{\overset{n}{\boxplus}} B_i,\ \Omega = \underset{i=1}{\overset{n}{\boxplus}} \Omega_i, \Theta = \bigcup_{i=1}^{n} \Theta_i) \\
& \| \underset{i=1}{\overset{n}{R}} S_i(C_i, R_i^{\,c}, R_i^{\,i'}, R_i^{\,o'}, B_i, \Omega_i, \Theta'_i \mid R^{i'}{}_i = R^i{}_i \cup \{C \times C_i\}, \\
& \quad R^{o'}{}_i = R^o{}_i \cup \{C_i \times C\}, \Theta'{}_i = \Theta_i \cup C)
\end{aligned}
\tag{53}
$$

Figure 5. Basic types of system structural relations in system compositions

No.	Type of composition	Syntax	Example
1	Parallel	$S_1 \parallel S_2$	$S \triangleq S_1 \parallel S_2 \parallel \ldots \parallel S_n$ 
2	Serial	$S_1 \rightarrow S_2$	$S \triangleq S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$ 
3	Nested	$S_1 \rightarrowtail S_2$	$S \triangleq S_1 \rightarrowtail S_2 \rightarrowtail \ldots \rightarrowtail S_n$ 

more systems into a supersystem with a hierarchical architecture. There are three basic types of system structural relations in system composition known as *parallel* (||), *serial* (→), and *nested* (↣) as shown in Figure 5. Complex system compositions can be represented by a combination of these three meta-architectural relations between subsystems. The syntaxes and semantics of these three system relations in system compositions can be referred to related definitions in RTPA (Wang, 2002, 2003, 2006c, 2007c, 2008a, 2008d).

According to Definition 33, a system can be integrated from the bottom up by a series of compositions, level-by-level, in a system hierarchy.

Example 3. *A composed system S(C, R, B, Ω ,Θ) as given in Figure 6 can be formally described as follows:*

$$S(C, R, B, \Omega\ \Theta) \triangleq S_1 \parallel S_2 \parallel \ldots \parallel S_x$$

in which the subsystems of S can be refined as in Box 13.

System Difference

Definition 34. *The difference between a closed systems $\widehat{S}$ and its subsystem $\widehat{S_1}$, denoted by ⊟, results in a closed subsystem $\widehat{S_2}$ that is formed by the difference of sets of components (C_1), and the differences of the internal relations, behaviors, and constraints (R_1, B_1, Ω_1) with their incremental counterparts (ΔR_{12}, ΔB_{12}, $\Delta\Omega_{12}$), see Box 14.*

According to Definition 34, a difference of a subsystem from a system $\widehat{S}$, will result in the removal of not only the given subsystem but also all

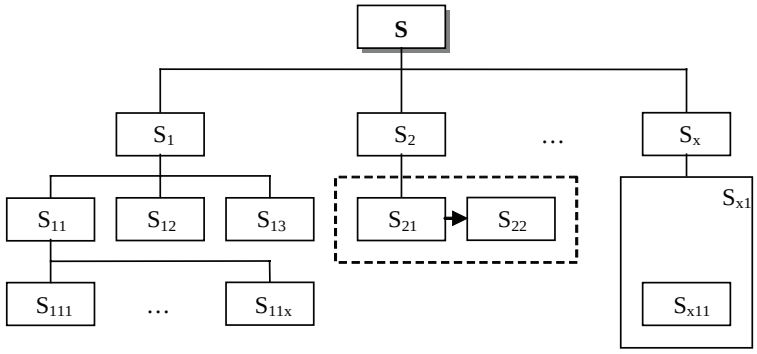
Box 13.

$$S_1(C_1, R_1, B_1, \Omega_1, \Theta_1) \triangleq S_{11} \| S_{12} \| S_{13} = (S_{111} \| .. \| S_{11x}) \| S_{12} \| S_{13}$$

$$S_2(C_2, R_2, B_2, \Omega_2, \Theta_2) \triangleq S_{21} \rightarrow S_{22}$$

$$S_x(C_x, R_x, B_x, \Omega_x, \Theta_x) \triangleq S_{x1} \rightarrowtail S_{x11}$$

Figure 6. The hierarchical structure of system compositions



Box 14.

$$\begin{aligned}&\widehat{S}(C, R, B, \Omega) \boxminus \widehat{S_1}(C_1, R_1, B_1, \Omega_1)\\ &\triangleq \widehat{S_2}(C_2, R_2, B_2, \Omega_2 \mid C_2 = C \setminus C_1, R_2 = R \setminus (R_1 \cup \Delta R_{12}),\\ &\qquad B_2 = B \setminus (B_1 \cup \Delta B_{12}), \Omega_2 = \Omega \setminus (\Omega_1 \cup \Delta \Omega_{12}))\end{aligned} \tag{54}$$

interrelations and incremental behaviors between the subsystem and other subsystem in it.

Similarly, the difference of open systems can be defined as follows.

Definition 35. *The difference between an open systems S and its subsystem S_1, denoted by ⊟, results in an open subsystem S_2 that is formed by the difference of sets of components and I/O relations*

(C_1, R^i_1, R^o_1), and the differences of the internal relations, behaviors, and constraints (R^c_1, B_1, Ω_1) with their incremental counterparts (ΔR^c_{12}, ΔB_{12}, $\Delta\Omega_{12}$), see Box 15.

The operation of open system difference can also be illustrated by Figure 4, where S_1 and all related I/O relations should be removed in the operation $S_2 = S \boxminus S_1$,.

System Decomposition

A system decomposition is an inverse operation of system composition that breaks up a system into two or more subsystems. System decomposition can be described based on the concept of system difference, which is an inversed operation of the incremental union of sets.

Definition 36. *The decomposition of an open systems S, denoted by* $\pitchfork$*, is to break up S into two or more subsystems at a given level of the system hierarchy by one of the compositional relations* $R_c = \{||, \rightarrow, \rightarrowtail\}$*, see Box16.*

As specified in Definition 36, the decomposition operation results in the removal of all internal relations $\Delta R_{ij}{}^c = C_i \times C_j, 1 \le i, j \le n$ that are no longer belong to any of its subsystems.

Box 15.

$$
\begin{aligned}
&S(C, R^c, R^i, R^o, B, \Omega, \Theta) \boxminus S_1(C_1, R_1{}^c, R_1{}^i, R_1{}^o, B_1, \Omega_1, \Theta_1) \\
&\triangleq S_2(C_2, R_2{}^c, R_2{}^i, R_2{}^o, B_2, \Omega_2, \Theta_2 \mid C_2 = C \setminus C_1, R_2{}^c = R^c \setminus (R_1{}^c \cup \Delta R_{12}{}^c), \\
&\quad R_2{}^i = R^i \setminus R_1{}^i, R_2{}^o = R^o \setminus R_1{}^o, B_2 = B \setminus (B_1 \cup \Delta B_{12}), \\
&\quad \Omega_2 = \Omega \setminus (\Omega_1 \cup \Delta\Omega_{12}), \Theta_2 = \Theta_1)
\end{aligned}
\tag{55}
$$

Box 16.

$$
\begin{aligned}
&S(C, R^c, R^i, R^o, B, \Omega, \Theta) \pitchfork_{i=1}^{n} \S_i \ C_i, R_i{}^c, R_i{}^i, R_i{}^o, B_i, \Omega_i, \Theta_i) \\
&\triangleq \mathop{\mathrm{R}}_{i=1}^{n} \{ \ S_i(C_i, R^c{}_i, R^{i'}{}_i, R^{o'}{}_i, B_i, \Omega_i, \Theta'{}_i \mid R^{i'}{}_i = R^i{}_i \cup \{C \times C_i\}, \\
&\qquad R^{o'}{}_i = R^o{}_i \cup \{C_i \times C\}, \Theta'{}_i = \Theta_i \cup C) \\
&\quad \| \ S(C, R^c, R^i, R^o, B, \Omega, \Theta) \boxminus S_i(C_i, R_i{}^c, R_i{}^i, R_i{}^o, B_i, \Omega_i, \Theta_i) \\
&\quad \}
\end{aligned}
\tag{56}
$$

Similarly, the decomposition of a closed system into multiple subsystems can be defined as follows.

Definition 37. *The decomposition of a closed system $\widehat{S}$, denoted by $\pitchfork$, is to break up $\widehat{S}$ into two or more subsystems at a given level of the system hierarchy by one of the compositional relations $R_c = \{||, \rightarrow, \rightarrowtail\}$, see Box17.*

Definitions 36 and 37 indicate that either an open or a closed system can be resolved from the top down by a series of decompositions, level-by-level, in the system hierarchy. It is noteworthy that both open and closed system decompositions result in a set of open subsystems.

System Aggregation

Definition 38. *The aggregation of a closed system $\widehat{S}$ from a set of n peer systems $\widehat{S}_i$, $1 \le i \le n$, denoted by $\Leftarrow$, is an aggregation of $\widehat{S}$ with the elicitation of interested subsets of components C'_i, behaviors B'_i, and constraints Ω'_i, see Box 18. Similarly, the aggregation of open systems can be defined as follows.*

Definition 39. *The aggregation of an open system S from a set of n peer systems S_i, $1 \le i \le n$, denoted by $\Leftarrow$, is an aggregation of S with the elicitation of interested subsets of components C'_i, behaviors B'_i, and constraints Ω'_i; and at the same time, it establishes new associations among all aggregated*

Box 17.

$$\begin{aligned}
&\hat{S}(C,R,B,\Omega) \pitchfork_{i=1}^{n} \widehat{S}_i(C_i,R_i,B_i,\Omega_i) \\
&\triangleq \mathop{R}_{i=1}^{n}\{\ S_i(C_i,R^c{}_i,R^i{}_i,R^o{}_i,B_i,\Omega_i,\Theta_i \mid R^i{}_i = (C\times C_i), R^o{}_i = (C_i\times C), \Theta_i = C) \\
&\quad \| \ \hat{S}(C,R,B,\Omega) \boxminus \widehat{S}_i(C_i,R_i,B_i,\Omega_i) \\
&\quad \}
\end{aligned} \tag{57}$$

Box 18.

$$\begin{aligned}
&S(C,R,B,\Omega) \Leftarrow \mathop{R}_{i=1}^{n}(S_i\ C_i,R_i,B_i,\Omega_i) \\
&\triangleq S(C,R,B,\Omega \mid C = \bigcup_{i=1}^{n} C'_i \subseteq C_i, R = C\times C, \\
&\qquad\qquad B = \bigcup_{i=1}^{n} B_i \subseteq B_i,\ \Omega = \bigcup_{i=1}^{n} C'_i \subseteq C_i)
\end{aligned} \tag{58}$$

Box 19.

$$S(C, R^c, R^i, R^o, B, \Omega, \Theta) \Leftarrow \mathop{R}_{i=1}^{n} S_i(C_i, R_i^c, R_i^i, R_i^o, B_i, \Omega_i, \Theta_i)$$

$$\hat{=} S(C, R^c, R^i, R^o, B, \Omega, \Theta \mid C = \bigcup_{i=1}^{n} C'_i \subseteq C_i, R^c = \{C \times C\},$$

$$R^i = \bigcup_{i=1}^{n} (R^i_i \cup \{C_i \times C\}), R^o = \bigcup_{i=1}^{n} (R^o_i \cup \{C \times C_i\}),$$

$$B = \bigcup_{i=1}^{n} B'_i \subseteq B_i,\ \Omega = \bigcup_{i=1}^{n} C'_i \subseteq C_i,\ \Theta \subseteq C_\Theta \cup \bigcup_{i=1}^{n} \Theta_i)$$

$$|| \mathop{R}_{i=1}^{n} S_i(C_i, R_i^c, R_i^{i'}, R_i^{o'}, B_i, \Omega_i, \Theta'_i \mid R^{i'}_i = R^i_i \cup \{C \times C_i\},$$

$$R^{o'}_i = R^o_i \cup \{C_i \times C\}, \Theta'_i = \Theta_i \cup C) \quad (59)$$

systems, see Box 19. System aggregation is also known as system elicitation.

According to Definitions 32 and 39, the difference between system composition and aggregation is that the former constructs a new system by integrating a set of entire systems as subsystems while the latter constructs a new system by eliciting interested subsets of components, behaviors, and/or constraints from a set of individual and independent systems.

System Specification

A system specification is an inverse operation of system aggregation. System specification is usually operated in a series of refinements.

Definition 40. *The specification of a closed system $\widehat{S}_0$ by a set of n refined systems $\widehat{S}_i$, $1 \le i \le n$, denoted by $\vdash$, is a specification of $\widehat{S}_0$ with a total order of a series of refinements by increasingly specific and detailed components C_i, behaviors B_i, and constraints Ω_i, see Box 20.*

Similarly, the specification of open systems can be defined as follows.

Definition 41. *The specification of an open system S_0 by a set of n refined systems S_i, $1 \le i \le n$, denoted by $\vdash$, is a specification of S_0 with a total order of a series of refinements by increasingly specific and detailed components C_i, behaviors B_i, and constraints Ω_{ij}; and at the same time, it establishes new associations among all refining systems, see Box 21.*

System specification is a refinement process where more specific and detailed components, behaviors, and constraints are developed in a consistent and coherent top-down hierarchy. The major tasks of system specifications are system architecture (component) and behavior specifications, which can be further refined by the Com-

Box 20.

$$
\begin{aligned}
&(\widehat{S}_n \vdash ... \vdash \widehat{S}_2 \vdash \widehat{S}_1) \vdash \widehat{S}_0(C_0, R_0, B_0, \Omega_0) \\
&\triangleq \widehat{S}_0(C_0, R_0, B_0, \Omega_0 \mid C_0 = \overset{n}{\underset{i=1}{R}}(C_{i-1} \subset C_i), R_0 = (C_0 \times C_0), \\
&\quad B_0 = (\overset{n}{\underset{i=1}{R}} B_{i-1} \subset B_i),\ \Omega_0 = (\overset{n}{\underset{i=1}{R}} B_{i-1} \subset B_i))
\end{aligned}
\tag{60}
$$

Box 21.

$$
\begin{aligned}
&(S_n \vdash ... \vdash S_2 \vdash S_1) \vdash S_0(C_0, R_0^c, R_0^i, R_0^o, B_0, \Omega_0, \Theta_0) \\
&\triangleq S_0(C_0, R_0^c, R_0^i, R_0^o, B_0, \Omega_0, \Theta_0 \mid C_0 = \overset{n}{\underset{i=1}{R}}(C_{i-1} \subset C_i), R_0^c = \{C_0 \times C_0\}, \\
&\quad R_0^i = \bigcup_{i=1}^{n}(R^i{}_i \cup \{C_i \times C_0\}), R_0^o = \bigcup_{i=1}^{n}(R^o{}_i \cup \{C_0 \times C_i\}), \\
&\quad B_0 = \overset{n}{\underset{i=1}{R}}(B_{i-1} \subset B_i),\ \Omega_0 = \overset{n}{\underset{i=1}{R}}(B_{i-1} \subset B_i),\ \Theta_0 = \Theta_1 = \Theta_2 = .. = \Theta_n) \\
&\mid\mid \overset{n}{\underset{i=1}{R}} S_i(C_i, R_i^c, R_i^{i'}, R_i^{o'}, B_i, \Omega_i, \Theta'_i \mid R_i^{i'} = R_i^i \cup \{C_0 \times C_i\}, \\
&\quad R_i^{o'} = R_i^o \cup \{C_i \times C_o\}, \Theta'_i = \Theta_i \cup C_0))
\end{aligned}
\tag{61}
$$

ponent Logical Models (CLMs) and processes as provided in RTPA (Wang, 2002, 2003, 2006a, 2006c, 2007c, 2008a, 2008d).

CONCLUSION

A new mathematical structure of abstract systems has been presented as the most complicated mathematical entities beyond sets, functions, concepts, and processes. A formal and rigorous treatment of abstract systems as well as their taxonomy and properties has been described. System algebra has been introduced as a set of relational and compositional operations for manipulating abstract systems and their composing rules. The former have been elicited as the algebraic operations of *independent*, *related*, *overlapped*, *equivalent*,

subsystem, and *supersystem*. The latter have been identified as the algebraic operations of *inheritance*, *tailoring*, *extension*, *substitute*, *difference*, *composition*, *decomposition*, *aggregation*, and *specification*. A wide range of applications of system algebra has been recognized in cognitive informatics, system science, system engineering, computing, and software engineering. System algebra has formed a fundamental theory for denoting the rigorous semantics of conventional object-oriented design notations and methodologies such as UML in software and intelligent system engineering.

ACKNOWLEDGMENT

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The author would like to thank anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Ashby, W. R. (1958). Requisite variety and implications for control of complex systems. *Cybernetica, 1,* 83-99.

Ashby, W. R. (1962) Principles of the self-organizing system. In von H. Foerster & G. Zopf (Eds.), *Principles of self-organization* (pp. 255-278). Oxford, England: Pergamon.

Eigen, M., & Schuster, P. (1979). *The hypercycle: A principle of natural self-organization.* Berlin, Germany: Springer.

Ellis, D. O., & Fred, J. L. (1962). *Systems philosophy.* Prentice Hall.

Ford, J. (1986). Chaos: Solving the Unsolvable, predicting the unpredictable. In *Chaotic dynamics and fractals.* Academic Press.

Haken, H. (1977). *Synergetics*. New York: Springer-Verlag.

Heylighen, F. (1989). Self-organization, emergence and the architecture of complexity. In *Proceedings of the First European Conference on System Science (AFCET)* (pp. 23-32). Paris.

Klir, G. J. (1992). *Facets of systems science*. New York: Plenum Press.

Klir, R. G. (1988). Systems profile: the emergence of systems science. *Systems Research, 5*(2), 145-156.

Prigogine, I., & Nicolis, G. (1972). Thermodynamics of evolution. *Physics Today, 25,* 23-28.

Rapoport, A. (1962). Mathematical aspects of general systems theory. *General Systems Yearbook, 11,* 3-11.

Skarda, C. A., & Freeman, W. J. (1987). How brains make chaos into order. *Behavioral and Brain Sciences, 10.*

von Bertalanffy, L. (1952). *Problems of life: An evolution of modern biological and scientific thought.* London: C. A. Watts.

Wang, Y. (2002). The real-time process algebra (RTPA). *The International Journal of Annals of Software Engineering, 14,* 235-274.

Wang, Y. (2003). Using process algebra to describe human and software system behaviors. *Brain and Mind, 4*(2), 199-213.

Wang, Y. (2005). System science models of software engineering. In *Proceedings of the Eighteenth Canadian Conference on Electrical and Computer Engineering (CCECE'05)* (pp. 1802-1805). Saskatoon, Saskatchewan, Canada: IEEE CS Press.

Wang, Y. (2006a). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation (Invited plenary talk). In *Proceedings of the First International Confer-*

ence on Rough Set and Knowledge Technology (RSKT'06) (LNAI 4062, pp. 69-78). Chongqing, China: Springer.

Wang, Y. (2006b). On abstract systems and system algebra. In *Proceedings of the Fifth IEEE International Conference on Cognitive Informatics (ICCI'06)* (pp. 332-343), Beijing, China: IEEE CS Press.

Wang, Y. (2006c, March). On the informatics laws and deductive semantics of software. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 161-171.

Wang, Y. (2007a). Keynote speech, on theoretical foundations of software engineering and denotational mathematics. In *Proceedings of the Fifth Asian Workshop on Foundations of Software* (pp. 99-102). Xiamen, China: BHU Press.

Wang, Y. (2007b). The OAR model of neural informatics for internal knowledge representation in the brain. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(3), 64-75.

Wang, Y. (2007c). Software engineering foundations: A software science perspective. In *CRC Series in Software Engineering: Vol. 2*. CRC Press.

Wang, Y. (2007d). The theoretical framework of cognitive informatics. *The International Journal of Cognitive Informatics and Natural Intelligence, 1*(1), 1-27.

Wang, Y. (2008a, April). Deductive semantics of RTPA. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 95-121.

Wang, Y. (2008b, April). On concept algebra: A denotational mathematical structure for knowledge and software modeling. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(2), 1-19.

Wang, Y. (2008c). On the big-R notation for describing iterative and recursive behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence, 2*(1), 17-28.

Wang, Y. (2008d, April). RTPA: A denotational mathematics for manipulating intelligent and computing behaviors. *The International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), 2*(2), 44-62.

Wang, Y., & Wang, Y. (2006). On cognitive informatics models of the brain. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 203-207.

Zadeh, L. A. (1965). Fuzzy sets and systems. In J. Fox (Ed.), *Systems theory* (pp. 29-37). Brooklyn, NY: Polytechnic Press.

Zadeh, L. A. (1973). Outline of a new approach to analysis of complex systems. *IEEE Trans. on Sys., Man and Cyb., 1*(1), 28-44.

This work was previously published in the International Journal of Cognitive Informatics and Natural Intelligence, edited by Y. Wang, Volume 2, Issue 2, pp. 20-43, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 7.27
A Cognitive Informatics Reference Model of Autonomous Agent Systems (AAS)

Yingxu Wang
University of Calgary, Canada

ABSTRACT

Despite the fact that the origin of software agent systems has been rooted in autonomous artificial intelligence and cognitive psychology, their implementations are still based on conventional imperative computing techniques rather than autonomous computational intelligence. This paper presents a cognitive informatics perspective on autonomous agent systems (AAS's). A hierarchical reference model of AAS's is developed, which reveals that an autonomous agent possesses intelligent behaviors at three layers known as those of imperative, autonomic, and autonomous from the bottom up. The theoretical framework of AAS's is described from the facets of cognitive informatics, computational intelligence, and denotational mathematics. According to Wang's abstract intelligence theory, an autonomous software agent is supposed to be called as an intelligent-ware, shortly, an intelware, parallel to hardware and software in computing, information science, and artificial intelligence.

INTRODUCTION

A *software agent* is an intelligent software system that autonomously carries out robotic and interactive applications based on goal-driven cognitive mechanisms. The studies on software agent are rooted in the essences of computing science and cognitive science such as automata theory (von Neumann, 1946, 1958, 1963, 1966; Shannon, 1956), Turing machines (Turing, 1950), cognitive psychology (Newell, 1990; Sternberg, 1997; Anderson and Rosenfeld, 1998; Matlin, 1998), artificial intelligence (*McCarthy,* 1955,

1963; McCulloch, 1943, 1965; Barr and Feigenbaum, 1981), computational intelligence (Poole et al., 1997; Wang, 2008a), and decision theories (Wald, 1950; Newell and Simon, 1972; Berger et al., 1990; Bronson and Naadimuthu, 1997; Wang and Ruhe, 2007; Wang, 2008b).

The history towards software agents may be traced back to the work as early as in the 1940s. J. *McCarthy*, W. McCulloch, *M.L. Minsky, N. Rochester, and C.E. Shannon* proposed the term *Artificial Intelligence* (AI) (*McCarthy,* 1955, 1963; McCulloch, 1943, 1965). S.C. Kleene analyzed the relations of *automata* and nerve nets (Kleene, 1956). Then, Bernard Widrow developed the technology of *artificial neural networks* in the 1950s (Widrow and Lehr, 1990). The concepts of *robotics* (Brooks, 1970) and *expert systems* (Giarrantans and Riley, 1989) were developed in the 1970s and 1980s, respectively. In 1992, the notion of *genetic algorithms* was proposed by J.H. Holland (Holland, 1992). Then, *distributed artificial intelligence* and *intelligent system* technologies emerged since late 1980s (Bond and Gasser, 1988; Kurzweil, 1990; Chaib-Draa et al., 1992; Meystel and Albus, 2002, Meystel and Albus, 2002).

The origin of the term *autonomous agent* is based on Carl Hewitt and his colleagues' *artificial intelligence actor models* proposed in 1973 (Hewitt et al., 1973, 1991). Then, as a novel approach of artificial intelligence, agent technologies have been proliferated since the early 1990s (Foner, 1993; Genesereth and Ketchpel, 1994; Hayes-Roth, 1995; Axelrod, 1997; Huhns and Singh, 1997; Wooldridge and Jenings, 1995; Wooldridge, 2002, Wang, 2003b). Pattie Maes perceived that a software agent is a process that lives in the world of computers and networks and that can operate autonomously to fulfill a set of tasks (Maes, 1991). Dimitris N. Chorafas described a software agent as a new software paradigm of things that think (Chorafas, 1998). Software agents are characterized by knowledge, learning, reasoning, and adaptation, which are rational to the extent that their behaviors are predictable by given goals and the solution environment (Russell and Norvig 1995; Poole, Mackworth, and Goebel 1997; Nilsson 1998).

Multi-agent systems are proposed in (Wittig, 1992; Wellman, 1999) as distributed intelligent systems (Bond and Gasser, 1988) in which each node is an autonomous software agent. The key technology of autonomous agent systems is how a variety of heterogeneous agents allocate their roles, coordinate their behaviors, share their resources, and communicate their information, beliefs, and needs (Maes, 1991). The interaction mechanisms of multi-agent systems, such as cooperation, negotiation, belief reconciliation, information sharing, and distributed decision making, are identified as important issues in the design and implementation of multi-agent systems.

Autonomic computing is one of the fundamental technologies of software agents, which is a mimicry and simulation of the natural intelligence possessed by the brain using general computers. Autonomic computing was first proposed by IBM in 2001, where it is perceived that "Autonomic computing is an approach to self-managed computing systems with a minimum of human interference. The term derives from the body's autonomous nervous system, which controls key functions without conscious awareness or involvement (IBM, 2006)." Various studies on autonomic computing have been reported following the IBM initiative (Kephart and Chess, 2003; Murch, 2004; Wang, 2004).

According to Wang's *abstract intelligence* theory (Wang, 2008a, 2009), software agents are a paradigm of abstract and computational intelligence, which is a subset of or an application-specific virtual brain. Behaviors of a software agent are mirrored human behaviors. Therefore, a software agent may be more accurately named as an *intelligent-ware*, shortly, an *intelware*, parallel to hardware and software in computing,

information science, and artificial intelligence. In this notion, intelware will be treated as a synonym of an autonomous agent system.

This paper presents a coherent theoretical framework of autonomous agent systems (AAS's) or *intelware* from the facets of cognitive informatics, computational intelligence, and denotational mathematics. The nature of software agents and intelware is elaborated. A reference model of AAS with intelligent behaviors at three layers known as those of imperative, autonomic, and autonomous is developed from the bottom up. The theoretical framework of AAS's/intelware is presented on the basis of cognitive informatics and computational intelligence theories. A set of denotational mathematics is introduced in order to provide a fundamental mathematical means for formally and rigorously dealing with the highly complicated architectures and intricate behaviors of AAS's and intelware.

THE NATURE OF SOFTWARE AGENTS AND INTELWARE

Definition 1. *A software agent, or more actually an intelware, is an intelligent software system that autonomously carries out robotistic and interactive applications based on goal-driven cognitive mechanisms.*

On the basis of Definition 1, an autonomous agent is a software agent that possesses high-level autonomous ability and behaviors beyond conventional imperative computing technologies.

Definition 2. *An Autonomous Agent System (AAS) is a composition of distributed agents that possesses autonomous computing and decision making abilities as well as interactive communication capability to peers and the environment.*

The classification of agent/intelware technologies can be described in Table 1, where *I* and *O* denote the inputs/outputs of a given AAS. When both input event (I) and output behavior (O) are constant, it denotes a *routine* intelware; while when both *I/O* are variable, it represents the most complicated *autonomous* intelware. Otherwise, the combinations of variable event/constant behavior and constant event/variable behaviors indicate an *algorithmic* or *autonomic* intelware, respectively.

In Table 1, the routine and algorithmic AAS's may be implemented by computational imperative behaviors. However, the autonomic AAS's should be implemented by autonomic computing, as that of the autonomous AAS's by autonomous mechanisms and behaviors.

Table 1. Classification of intelware / AAS's

		Behavior (O)	
		Constant	Variable
Event (I)	Constant	*Routine*	*Autonomic*
	Variable	*Algorithmic*	*Autonomous*

THE REFERENCE MODEL OF INTELWARE/AUTONOMOUS AGENT SYSTEMS

The reference model of intelware/AAS's is a hierarchical model with three layers known as those of imperative, autonomic, and autonomous behaviors. This section elaborates the mathematical models of the imperative and intelligent behaviors of intelware/AAS's in the layered reference model of agent intelligence.

The Hierarchical Behavioral Model of Intelware/AAS's

Behaviorism is a doctrine of psychology and cognitive informatics that describes the association between a given stimulus and an observed response of human brains and AAS's. Cognitive informatics reveals that human and AAS behaviors may be classified into four categories known as the *perceptive*, *cognitive*, *instructive*, and *reflective* behaviors (Wang, 2007b).

The reference model of AAS's (RMAAS) is a hierarchical behavioral model of agent intelligence as illustrated in Figure 1. In the RMAAS model, the hierarchy of agent behaviors can be divided into the imperative, autonomic, and autonomous layers. Conventional computing machines are implemented only by imperative behaviors. However, the autonomic computing systems and AAS's are implemented by advanced cognitive behaviors. *Imperative computing* is an enclosure of instructive and passive behaviors. The *autonomic computing* is an enclosure of internally motivated behaviors beyond those of the imperative space. The *autonomous computing* is an enclosure of perceptive- and inference-driven behaviors beyond those of both imperative and autonomic computing. More formal descriptions of the three types of behaviors of AAS's will be presented in the following subsections.

The Imperative Behavioral Layer of Intelware/AAS's

According to the RMAAS model as illustrated in Figure 1, the imperative behavioral intelligence of intelware and AAS's can be formally modeled and elaborated in this subsection.

Definition 3. *The imperative behavioral layer of AAS's, B_I, is a set of instruction-based behaviors such as the event-driven behaviors (B_e), time-driven behaviors (B_t), and interrupt-driven behaviors (B_{int}), i.e.:*

$$B_I \triangleq \{B_e, B_t, B_{int}\} \quad (1)$$

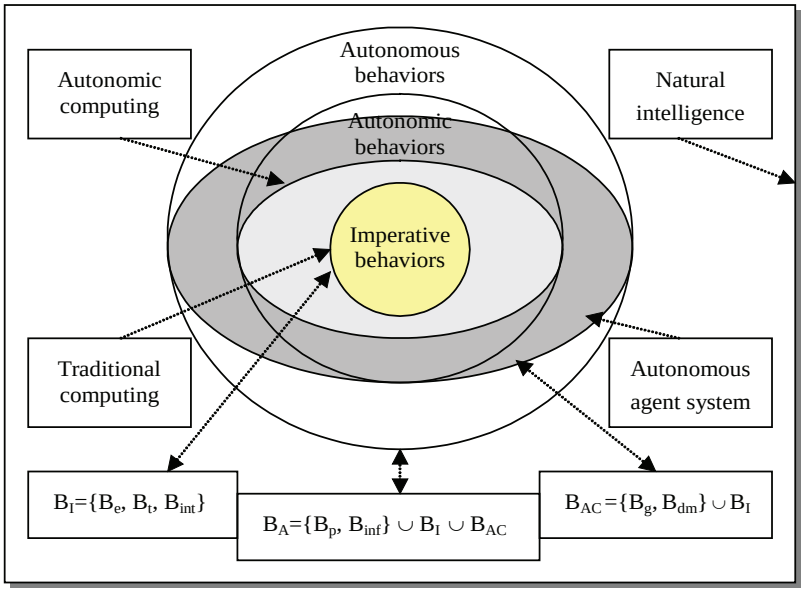
An imperative system implemented with B_I may do nothing unless a specific program is loaded, in which the stored program transfers a general-purpose computer to a specific intelligent application. The imperative system is a passive system that implements deterministic, context-free, and stored-program controlled behaviors.

Definition 4. *An event is an abstract variable that represents an external stimulus to a system or the occurring of an internal change of status, such as an action of users, an updating of the environment, and a change of the value of a control variable.*

The types of events that may trigger a behavior can be classified into operational (@*e*S), time (@*t*TM), and interrupt (@*int*⦿) events, where @ is the *event prefix*, and S, TM, and □ the type suffixes, respectively. The *interrupt event* is a kind of special event that models the interruption of an executing process, the temporal handover of controls to an Interrupt Service Routine (ISR), and the return of control after its completion.

Definition 5. *An interrupt, denoted by ↯, is a parallel process relation in which a running process*

Figure 1. The hierarchical reference model of autonomous agent systems (RMAAS)



P is temporarily held by another higher priority process Q via an interrupt event @int⊙ at the interrupt point ⊙, and the interrupted process will be resumed when the high priority process has been completed, i.e.:

$$P \lightning Q \triangleq P \,||\, (@int\odot \nearrow Q \searrow \odot) \qquad (2)$$

where ↗ and ↘ denote an interrupt service and an interrupt return, respectively.

In general, all types of events, including the operational events, timing events, and interrupt events, are captured by the system in order to dispatch a designated behavior.

Definition 6. *An event-driven behavior B_e, denoted by $\hookrightarrow_e$, is an imperative process in which the ith behavior in term of a designated process P_i is triggered by a predefined event $@e_i$S, i.e.:*

$$B_e \triangleq \mathop{\mathrm{R}}_{i=1}^{n} (@e_i\mathrm{S} \hookrightarrow_e P_i) \qquad (3)$$

where the big-R notation is a mathematical calculus that denotes a sequence of repetitive/iterative behaviors or a set of recurring structures (Wang, 2007a).

Definition 7. *A time-driven behavior B_t, denoted by $\hookrightarrow_t$, is an imperative process in which the ith behavior in term of process P_i is triggered by a predefined point of time $@t_i$TM, i.e.:*

$$B_t \triangleq \mathop{\mathrm{R}}_{i=1}^{n} (@t_i\mathrm{TM} \hookrightarrow_t P_i) \qquad (4)$$

where $@t_i$TM may be a system timing or external timing event.

Definition 8. *An interrupt-driven behavior B_{int}, denoted by $\hookrightarrow_{int}$, is an imperative process in which the ith behavior in term of process P_i is triggered by a predefined system interrupt $@int_i\odot$, i.e.:*

$$B_{int} \triangleq \mathop{\mathrm{R}}_{i=1}^{n}(@int_i \odot \hookrightarrow_{int} P_i) \quad (5)$$

As a summary, an imperative computing system can be described as follows.

Definition 9. *An Imperative Computing (IC) system is a passive system that implements deterministic, context-free, and stored-program controlled behaviors.*

The Autonomic Behavioral Layer of Intelware/AAS's

According to the RMAAS model as illustrated in Figure 1, the autonomic behavioral intelligence of intelware and AAS's can be formally modeled and elaborated in this subsection.

Definition 10. *The autonomic behavioral layer of AAS's, B_C, is a set of internally motivated and self- generated behaviors such as the goal-driven behaviors (B_g) and decision-driven behaviors (B_d) on the basis of the imperative layer B_I , i.e.:*

$$B_C \triangleq \{B_g, B_d\} \cup B_I = \{B_e, B_r, B_{int}, B_g, B_d\} \quad (6)$$

Definition 11. *A goal-driven behavior B_g, denoted by $\hookrightarrow_g$, is an autonomic process in which the ith behavior in term of process P_i is generated by the system itself, rather than be given, corresponding to the goal $@g_i$ST, i.e.:*

$$B_g \triangleq \mathop{\mathrm{R}}_{i=1}^{n}(@g_i\mathrm{ST} \hookrightarrow_{\mathrm{g}} P_i) \quad (7)$$

*where the goal, denoted by g*ST*, is a triple, i.e.:*

$$g\mathrm{ST} = (P, \Omega, \Theta) \quad (8)$$

in which that $P = \{p_1, p_2, \ldots, p_n\}$ is a finite nonempty set of purposes or motivations, Ω is a finite set of constraints for the goal, and Θ is the environment of the goal.

Definition 12. *A decision-driven behavior B_d, denoted by $\hookrightarrow_d$, is an autonomic process in which the ith behavior in term of process P_i is generated by a given decision $@d_i$ST, i.e.:*

$$B_d \triangleq \mathop{\mathrm{R}}_{i=1}^{n}(@\mathrm{d}_i\mathrm{ST} \hookrightarrow_{\mathrm{d}} P_i) \quad (9)$$

*where the decision, denoted by d*ST*, is a selected alternative $a \in \mathcal{A}$ from a nonempty set of alternatives $\mathcal{A}$, based on a given set of criteria C, i.e.:*

$$d = f(\mathcal{A}, C) = f: \mathcal{A} \times C \to \mathcal{A}, \ \mathcal{A} \neq \varnothing \quad (10)$$

Definition 13. *An Autonomic Computing (AC) system is an intelligent system that implements nondeterministic, context-dependent, and adaptive behaviors based on goal- and decision-driven mechanisms.*

The autonomic systems do not rely on instructive and procedural information, but are dependent on internal status and willingness that formed by long-term historical events and current rational or emotional goals (Wang, 2007d).

The Autonomous Behavioral Layer of Intelware/AAS's

According to the RMAAS model as illustrated in Figure 1, the autonomous behavioral intelligence of intelware and AAS's can be formally modeled and elaborated in this subsection.

Definition 14. *The autonomous behavioral layer of AAS's, B_A, is a set of autonomously generated*

behaviors by internal cognitive processes such as the perception-driven behaviors (B_p) and inference-driven behaviors (B_{inf}) on the basis of the imperative space B_I and the autonomic space B_C, i.e.:

$$B_A \triangleq \{B_p, B_{inf}\} \cup B_I \cup B_C = \{B_e, B_t, B_{int}, B_g, B_d, B_p, B_{inf}\} \quad (11)$$

The new forms of behaviors covered in the autonomous layer can be elaborated as follows.

Definition 15. *A perception-driven behavior B_p, denoted by $\hookrightarrow_p$, is a cognitive process in which the ith behavior in term of process P_i is generated by the result of a perceptive process $@p_i$PC, i.e.:*

$$B_p \triangleq \mathop{\mathrm{R}}_{i=1}^{n}(@p_i\mathrm{PC} \hookrightarrow_p P_i) \quad (12)$$

where PC *stands for a type of process, and the perception result* pPC *is an outcome of the cognitive process of perception that an AAS may generate.*

Inferences are cognitive processes that reason about a possible causality from given premises based on known causal relations between a pair of cause and effect proven true by empirical arguments, theoretical inferences, or statistical regulations.

Definition 16. *An inference-driven behavior B_{inf}, denoted by $\hookrightarrow_{inf}$, is a cognitive process in which the ith behavior in term of process P_i is generated by the result of an inference process $@inf_i$PC, i.e.:*

$$B_{inf} \triangleq \mathop{\mathrm{R}}_{i=1}^{n}(@inf_i\mathrm{PC} \hookrightarrow_{inf} P_i) \quad (13)$$

where formal inferences can be classified into the deductive, inductive, abductive, and analogical categories, as well as modal, probabilistic, and belief theories (Wang, 2007e).

As shown in Definition 16 and Figure 1, an AAS implemented on B_A extends the conventional behaviors B_I and B_C to more powerful and intelligent behaviors, which are generated by internal and autonomous processes such as the perception and inference processes. With the possession of all the seven forms of intelligent behaviors in B_A, the AAS may advance closer to the intelligent power of human brains.

Relationships between the Agent Behaviors of Intelware/AAS's at the Three Layers of RMAAS

Contrasting Definitions 3, 10, and 14, the following relationships among the three-layer agent intelligent behaviors can be established on the basis of the RMAAS model as illustrated in Figure 1.

Theorem 1. *The relationships of the imperative behaviors B_I, autonomic behaviors B_C, and cognitive behaviors B_A of intelware or AAS's are hierarchical and inclusive, i.e.:*

$$B_I \subseteq B_C \subseteq B_A \quad (14)$$

Theorem 1 and Definition 14 indicate that any lower layer behavior of an intelware or AAS is a subset of those of a higher layer. In other words, any higher layer behavior is a natural extension of those of lower layers as shown in Figure 1. Therefore, the *necessary and sufficient conditions of AAS's, C_{AAS}*, are the possession of all behaviors at the three layers.

Corollary 1. *The behavioral model of intelware or* AAS, *§*AASST, *can be logically modeled by a set of parallel processes that encompasses the imperative behaviors B_I, autonomic behaviors B_C, and autonomous behaviors B_A from the bottom-up, i.e.:*

$$
\begin{aligned}
\S AASST &\triangleq (B_I, B_C, B_A) \\
&= \{ (\ B_e, B_t, B_{int}) && // B_I \\
&\quad || (B_e, B_t, B_{int}, B_g, B_d) && // B_C \\
&\quad || (B_e, B_t, B_{int}, B_g, B_d, B_p, B_{\inf}) && // B_A \\
&\quad \}
\end{aligned} \tag{15}
$$

where || denotes a parallel relation in RTPA.

THEORETICAL FOUNDATIONS OF INTELWARE/AAS'S

Recent research reveals that the foundations of agent technologies root in cognitive informatics, denotational mathematics, and computational intelligence (Wang, 2002a, 2003b, 2008a). Along with the latest advances in cognitive informatics, non-imperative autonomous agent systems known as *intelware* and *cognitive computers* are emerging. This section explores the theoretical foundations of AAS's and intelware. The latest development of fundamental theories and technologies underpinning AAS's and intelware are highlighted.

Denotational Mathematics for AAS's

Applied mathematics can be classified into two categories known as *analytic* and *denotational* mathematics (Wang, 2002b, 2007a, 2008a, 2008c). The former are mathematical structures that deal with functions of variables as well as their operations and behaviors; while the latter are mathematical structures that formalize rigorous expressions and inferences of system architectures and behaviors with abstract concepts, complex relations, and dynamic processes. The denotational and expressive needs in cognitive informatics, computational intelligence, software engineering, and knowledge engineering have led to new forms of mathematics collectively known as denotational mathematics.

Definition 17. *Denotational mathematics is a category of expressive mathematical structures that deals with high-level mathematical entities beyond numbers and simple sets, such as abstract objects, complex relations, behavioral information, concepts, knowledge, processes, intelligence, and systems.*

The term denotational mathematics is first introduced by Yingxu Wang in the emerging discipline of cognitive informatics (Wang, 2002a, 2007a, 2008c). Typical paradigms of denotational mathematics are comparatively presented in Table 1, where their structures, mathematical entities, algebraic operations, and usages are contrasted. The paradigms of denotational mathematics as shown in Table 1 are *concept algebra* (Wang, 2008d), *system algebra* (Wang, 2008e), and *Real-Time Process Algebra* (RTPA) (Wang, 2002b, 2008f).

The emergence of denotational mathematics is driven by the practical needs in cognitive informatics, computational intelligence, computing science, software science, and knowledge engineering, because all these modern disciplines study complex human and machine behaviors and their rigorous treatments. Among the new forms of denotational mathematics, *concept algebra* is designed to deal with the abstract mathematical structure of concepts and their representation and manipulation in knowledge engineering. *System algebra* is created to the rigorous treatment of abstract systems and their algebraic relations and operations. RTPA is developed to deal with series of behavioral processes and architectures of human and systems.

Denotational mathematics provides a powerful mathematical means for modeling and formalizing AAS's. Not only the architectures of AAS's, but also their dynamic behaviors can be rigorously and systematically manipulated by denotational mathematics. Applications of denotational mathe-

Table 2. Paradigms of denotational mathematics

No.	Paradigm	Structure	Mathematical entities	Algebraic operations	Usage
1	Concept algebra	$CA \triangleq (C, OP, \Theta) = (\{O, A, R^c, R^i, R^o\}, \{\bullet_r, \bullet_c\}, \Theta_C)$	$c \triangleq (O, A, R^c, R^i, R^o)$	$\bullet_r \triangleq \{\leftrightarrow, \nleftrightarrow, \prec, \succ, =, \cong, \sim, \triangleq\}$ $\bullet_c \triangleq \{\Rightarrow, \overset{-}{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \uplus, \Cap, \Lsh, \vdash, \mapsto\}$	Algebraic manipulations on abstract concepts
2	System algebra	$SA \triangleq (\mathcal{C}, OP, \Theta) = (\{C, R^c, R^i, R^o, B, \Omega\}, \{\bullet_r, \bullet_c\}, \Theta)$	$S \triangleq (\mathcal{C}, R^c, R^i, R^o, B, \Omega, \Theta)$	$\bullet_r \triangleq \{\nleftrightarrow, \leftrightarrow, \prod, =, \sqsubseteq, \sqsupseteq\}$ $\bullet_c \triangleq \{\Rightarrow, \overset{-}{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \sqcup, \uplus, \Cap, \Lsh, \vdash\}$	Algebraic manipulations on abstract systems
3	Real-time process algebra (RTPA)	$RTPA \triangleq (\mathfrak{T}, \mathfrak{P}, \mathfrak{R})$	$\mathfrak{P} \triangleq \{:=, \blacklozenge, \Rightarrow, \Leftarrow, \nRightarrow, \gg, \ll, \mid \gg, \mid \ll, @, \triangleq, \uparrow, \downarrow, !, \otimes, \boxtimes, \S\}$ $\mathfrak{T} \triangleq \{$N, Z, R, S, BL, B, H, P, TI, D, DT, RT, ST, @*eS*, @*tTM*, @*int*$\odot$, @*sBL*$\}$	$\mathfrak{R} \triangleq \{\mapsto, \curvearrowright, \mid, \mid, \ldots, \mid, \ldots \mid \ldots, R^*, R^+, R^i, R^i, \circlearrowleft, \rightarrowtail, \mid\mid, \Cap, \gg, \lightning, \hookrightarrow, \hookrightarrow, \hookrightarrow, \hookrightarrow, \hookrightarrow, \hookrightarrow\}$	Algebraic manipulations on abstract processes

matics in cognitive informatics and computational intelligence have been elaborated with a wide range of real-world case studies (Wang, 2008a, 2008c), which demonstrate that denotational mathematics is an ideal mathematical means for dealing with concepts, knowledge, behavioral processes, and human/machine intelligence in ASS's and intelware.

Cognitive Informatics Theories of AAS's

Cognitive informatics is the transdisciplinary enquiry of cognitive and information sciences that investigates into the internal information processing mechanisms and processes of the brain and natural intelligence, and their engineering applications via an interdisciplinary approach (Wang, 2002a, 2003a , 2003b, 2006, 2007a, 2007b, 2007c, 2007d). According to the abstract intelligence theory (Wang, 2008a, 2009), because cognitive informatics investigates the internal information processing mechanisms and processes of the brain and natural intelligence, its research results underlie the engineering applications of AAS's. Cognitive informatics reveals that artificial intelligence (AI) is a subset of natural intelligence (NI) (Wang, 2007a, 2007b). Therefore, AAS's may be referred to the natural intelligence and behavioral mechanisms of human beings.

A Layered Reference Model of the Brain (LRMB) is developed (Wang, et al., 2006) that reveals the logical model of NI and a coherent set of cognitive mechanisms. LRMB presents a systematical view toward the formal description and modeling of architectures and behaviors of AAS's, which are created to extend human capability, reachability, and/or memory capacity. The LRMB model explains the functional mechanisms and cognitive processes of the natural intelligence with 39 cognitive processes at seven layers known as the *sensation*, *memory*, *perception*, *action*, *meta-cognitive*, *meta-inference*, and *higher cognitive layers* from the bottom up. LRMB elicits the core and highly repetitive recurrent cognitive processes from a huge variety of life functions, which may shed light on the study of the fundamental mechanisms and interactions of complicated mental processes as well as AAS's, particularly the relationships and interactions between the inherited and the acquired life functions as well as those of the subconscious and conscious cognitive processes. The cognitive model of the brain can be used as a reference model for goal- and inference-driven technologies in AAS's.

Definition 18. *The cognitive model of the kernel of an AAS or intelware, AAS_k, can be described as a real-time intelligent system with an inherited Agent Operating System AOS and a set of Agent Intelligent Behaviors AIB in parallel, i.e.:*

$$AAS_k \triangleq AOS \,||\, AIB \tag{16}$$

Definition 19. *The Cognitive Models of Memory (CMM) states that the architecture of human memory is parallel configured by the Sensory Buffer Memory (SBM), Short-Term Memory (STM), Long-Term Memory (LTM), Conscious Status Memory (CSM), and Action-Buffer Memory (ABM), i.e.:*

$$\begin{aligned} CMM \triangleq (\ & LTM \\ & ||\, STM \\ & ||\, CSM \\ & ||\, SBM \\ & ||\, ABM \\ &) \end{aligned} \tag{17}$$

The CMM model provides a neural informatics foundation of natural intelligence. With the CMM model, the broad sense of an AAS, AAS', can be described by mimicking the abstract architecture and mechanisms of the brain.

Definition 20. *The cognitive model of AAS's, AAS, is represented by a real-time intelligent system that encompasses the intelware and the CMM as well as their interactions, i.e.:*

$$
\begin{aligned}
AAS &\triangleq Intelware \\
&\;||\, CMM \\
&= (\; AOS \\
&\quad ||\, AIB \\
&\quad) \\
&\;||\, (\; LTM \\
&\quad ||\, STM \\
&\quad ||\, CSM \\
&\quad ||\, SBM \\
&\quad ||\, ABM \\
&\quad)
\end{aligned}
\tag{18}
$$

Eq. 18 indicates that although *intelware* is considered the center of AAS's, the memories are essential to enable it to properly function, and to keep temporary and permanent results physiologically retained and retrievable.

Computational Intelligence Theories of AAS's

According to the abstract intelligence theory (Wang, 2008a, 2009), intelligence is perceived as the driving force or the ability to acquire and use knowledge and skills, or to reason in problem solving. It was conventionally perceived that only human beings possess higher-level intelligence. However, the development of computers, robots, intelligent systems, and AAS's indicates that intelligence may also be created or implemented by machines and man-made systems.

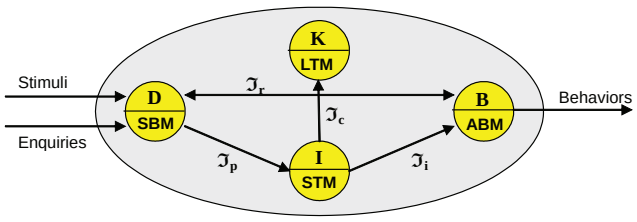
Definition 21. *Intelligence, in the narrow sense, is a human or a system ability that transforms information into behaviors; and in a broad sense, it is any human or system ability that autonomously transfers the forms of abstract information between data, information, knowledge, and behaviors in the brain.*

Definition 22. *The Generic Abstract Intelligence Model (GAIM), as shown in Figure 2, represents abstract intelligence in four forms known as the perceptive, cognitive, instructive, and reflective intelligence, corresponding to the specific forms of cognitive information and their memories.*

The GAIM indicates that different forms of intelligence are the driving force that transfers between a pair of abstract objects in the brain such as *data* (*D*), *information* (*I*), *knowledge* (*K*), and *behavior* (*B*). It is noteworthy that each abstract object is physiologically retained in a particular type of memories as given in the CMM model. This is the neural informatics foundation of natural intelligence, and the physiological evidences of why natural intelligence can be classified into four forms as shown in Figure 2.

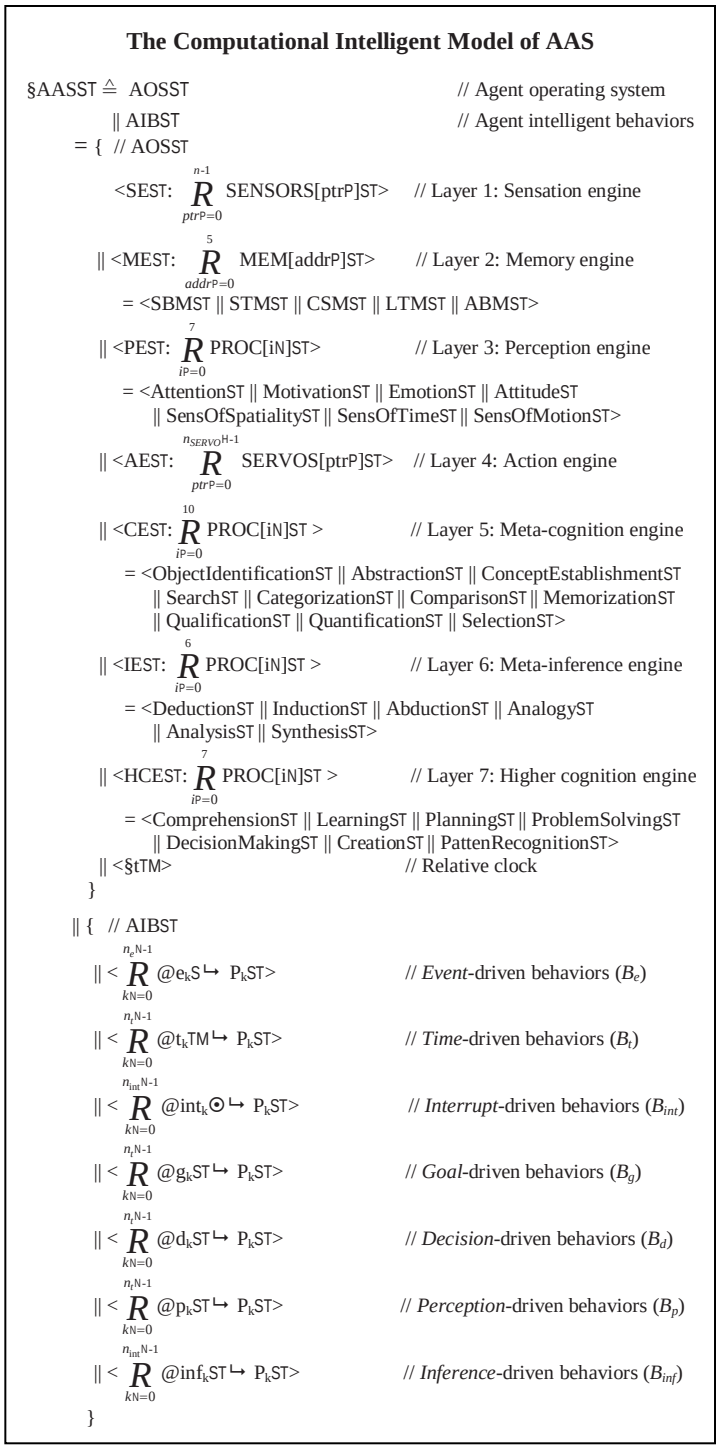
According to Definitions 21 and 22, *computational intelligence* is a paradigm of abstract

Figure 2. The generic abstract intelligence model (GAIM)



$\mathfrak{I}_p$ – *Perceptive* intelligence
$\mathfrak{I}_i$ – *Instructive* intelligence
$\mathfrak{I}_c$ – *Cognitive* intelligence
$\mathfrak{I}_i$ – *Reflective* intelligence

Figure 3. The computational intelligence model of AAS



The Computational Intelligent Model of AAS

§AASST ≜ AOSST // Agent operating system

|| AIBST // Agent intelligent behaviors

= { // AOSST

<SEST: $\overset{n-1}{\underset{ptrP=0}{R}}$ SENSORS[ptrP]ST> // Layer 1: Sensation engine

|| <MEST: $\overset{5}{\underset{addrP=0}{R}}$ MEM[addrP]ST> // Layer 2: Memory engine

= <SBMST || STMST || CSMST || LTMST || ABMST>

|| <PEST: $\overset{7}{\underset{iP=0}{R}}$ PROC[iN]ST> // Layer 3: Perception engine

= <AttentionST || MotivationST || EmotionST || AttitudeST

|| SensOfSpatialityST || SensOfTimeST || SensOfMotionST>

|| <AEST: $\overset{n_{SERVO}H-1}{\underset{ptrP=0}{R}}$ SERVOS[ptrP]ST> // Layer 4: Action engine

|| <CEST: $\overset{10}{\underset{iP=0}{R}}$ PROC[iN]ST > // Layer 5: Meta-cognition engine

= <ObjectIdentificationST || AbstractionST || ConceptEstablishmentST

|| SearchST || CategorizationST || ComparisonST || MemorizationST

|| QualificationST || QuantificationST || SelectionST>

|| <IEST: $\overset{6}{\underset{iP=0}{R}}$ PROC[iN]ST > // Layer 6: Meta-inference engine

= <DeductionST || InductionST || AbductionST || AnalogyST

|| AnalysisST || SynthesisST>

|| <HCEST: $\overset{7}{\underset{iP=0}{R}}$ PROC[iN]ST > // Layer 7: Higher cognition engine

= <ComprehensionST || LearningST || PlanningST || ProblemSolvingST

|| DecisionMakingST || CreationST || PattenRecognitionST>

|| <§tTM> // Relative clock

}

|| { // AIBST

|| < $\overset{n_eN-1}{\underset{kN=0}{R}}$ @e_kS↦ P_kST> // *Event*-driven behaviors (B_e)

|| < $\overset{n_tN-1}{\underset{kN=0}{R}}$ @t_kTM↦ P_kST> // *Time*-driven behaviors (B_t)

|| < $\overset{n_{int}N-1}{\underset{kN=0}{R}}$ @int_k⊙↦ P_kST> // *Interrupt*-driven behaviors (B_{int})

|| < $\overset{n_tN-1}{\underset{kN=0}{R}}$ @g_kST↦ P_kST> // *Goal*-driven behaviors (B_g)

|| < $\overset{n_tN-1}{\underset{kN=0}{R}}$ @d_kST↦ P_kST> // *Decision*-driven behaviors (B_d)

|| < $\overset{n_tN-1}{\underset{kN=0}{R}}$ @p_kST↦ P_kST> // *Perception*-driven behaviors (B_p)

|| < $\overset{n_{int}N-1}{\underset{kN=0}{R}}$ @inf_kST↦ P_kST> // *Inference*-driven behaviors (B_{inf})

}

intelligence. Computational intelligence models human intelligence by computational methodologies and cognitively inspired models.

Definition 23. *The computational intelligence model of AAS's and intelware, §AAS*ST*, is a parallel structure represented by the Agent Operating System (AOS*ST*) and a set of agent intelligence represented by the Agent Intelligent Behaviors (AIB*ST*), as shown in Figure 3.*

The GAIM and §AASST model reveal that NI and AI share the same cognitive informatics foundations on the basis of abstract intelligence. The *compatible intelligent capability* states that NI, AI, AAS's, and intelware are compatible by sharing the same mechanisms of intelligent capability and behaviors. In other words, at the logical level, NI of the brain shares the same mechanisms as those of AI and computational intelligence. The differences between NI and AI are only distinguishable by the means of implementation and the extent of intelligent ability. Therefore, the studies on NI and AI in general, and intelware and AAS's in particular, may be unified into a coherent framework based on cognitive informatics and computational intelligence, which are formalized by denotational mathematics.

CONCLUSION

This paper has presented a coherent theoretical framework of Autonomous Agent Systems (AAS), known as *intelware*, from the facets of cognitive informatics, computational intelligence, and denotational mathematics. A reference model of AAS has been developed with three-layer intelligent behaviors known as the imperative, autonomic, and autonomous agent intelligence from the bottom up. It has been recognized that the characteristics of an AAS is its perception-driven and inference-driven behaviors beyond the imperative and autonomic ones as provided by conventional imperative and autonomic computing.

In order to formally and rigorously deal with the highly complicated architectures and intricate behaviors of intelware and AAS's, a new mathematical means known as denotational mathematics has been developed. Typical paradigms of denotational mathematics have been introduced such as concept algebra, system algebra, and RTPA. The findings of this work, particularly the necessary and sufficient conditions of imperative and autonomous computing, and the abstract intelligence model of natural and artificial intelligence, have formed a solid foundation for explaining and developing advanced autonomous computing systems and their engineering applications.

ACKNOWLEDGMENT

The author would like to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) to this work. The author would like to thank the valuable comments and suggestions of the reviewers and colleagues.

REFERENCES

Anderson, J.A. and E. Rosenfeld, eds. (1988). *Neurocomputing: Foundations of Research*, Cambridge.

Axelrod, R. (1977), *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*, Princeton Univ. Press, Princeton, NJ.

Barr, A. and E. A. Feigenbaum, eds. (1981), *The Handbook of Artificial Intelligence*, Vol. 1. Stanford and Los Altos, CA: HeurisTech Press and Kaufmann.

Berger, J. (1990), *Statistical Decision Theory – Foundations, Concepts,* and Methods, Springer-Verlag.

Bond, A.H. and L. Gasser (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.

Bronson, R. and G. Naadimuthu (1997), *Schaum's Outline of Theory and Problems of Operations Research*, 2nd ed., McGraw-Hill, NY.

Brooks, R.A. (1970), New Approaches to Robotics, *American Elsevier*, NY, 5, 3-23.

Chaib-Draa, B. Moulin, R. Mandiau, and P. Millot, (1992), Trends in Distributed Artificial Intelligence, *Artificial Intelligence Review* 6, 35-66.

Chorafas, D.N. (1998), *Agent Technology Handbook*, McGraw-Hill, NY.

Foner, L. (1993), What ís an Agent, Anyway? A Sociological Case Study, *Agents Memo 93-01*, MIT Media Lab, Cambridge, MA.

Genesereth, M.R. and S.P. Ketchpel (1994), Software Agents, *Communications of the ACM*, 37 (7), 48-53.

Giarrantans, J. and G. Riley (1989), *Expert Systems: Principles and Programming*, PWS-KENT Pub. Co., Boston.

Hayes-Roth, B. (1995), An Architecture for Adaptive Intelligent Systems, *Artificial Intelligence*, 72(1-2), 329-365.

Hewitt, C., R. Bishop, and R. Steiger (1973), A Universal Modular Actor Formalism for Artificial Intelligence, *Proc. 3rd Int. Joint Conf. on Artificial Intelligence*, Stanford, CA, Aug.

Hewitt, C. and J. Inman (1991), DAI Betwixt and Between: From Intelligent Agents to Open Systems Science, IEEE Trans. on System, Man, and Cybernetics, Nov/Dec.

Holland, J.H. (1992), Genetic Algorithms, *Scientific American*, 267, 66-72.

Huhns, M., and M. Singh, eds. (1997). *Readings in Agents*, Kaufmann, San Francisco.

IBM (2006), *Autonomous Computing White Paper: An Architectural Blueprint for Autonomous Computing*, 4th ed., June, 1-37.

Jennings, N.R. (2000), On Agent-Based Software Engineering, *Artificial Intelligence*, 17(2), 277-296.

Kephart, J. and D. Chess (2003), The Vision of Autonomic Computing, IEEE *Computer*, 26(1), Jan, 41-50.

Kleene, S.C. (1956), Representation of Events by Nerve Nets, in C.E. Shannon and J. McCarthy eds., *Automata Studies*, Princeton Univ. Press, 3-42.

Kurzweil, R. (1990). *The Age of Intelligent Machines.* Cambridge, MA, MIT Press.

Maes, P. ed. (1991), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, London, The MIT press.

Matlin, M.W. (1998), *Cognition*, 4th ed., Harcourt Brace College Publishers, Orlando, FL.

McCarthy, J., M.L. Minsky, N. Rochester, and C.E. Shannon (1955), *Proposal for the 1956 Dartmouth Summer Research Project on Artificial Intelligence*, Dartmouth College, Hanover, NH, USA, http://www.formal.stanford.edu/jmc/history/dartmouth/*dartmouth.html*.

McCarthy, J. (1963), *Situations, Actions, and Causal Laws*, Memo 2, Stanford University Artificial Intelligence Project, Stanford, CA.

McCulloch, W. S., and W. Pitts. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics* 5, 115–137.

McCulloch, W.S. (1965), *Embodiments of Mind*, MIT Press, Cambridge, MA.

Meystel, A.M. and J.S. Albus (2002), *Intelligent Systems, Architecture, Design, and Control*, John Wiley & Sons.

Murch, R. (2004), *Autonomic Computing*, Person Education, London.

Newell, A. (1990), *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA.

Newell, A., and H.A. Simon (1972), *Human Problem Solving*, Prentice-Hall Englewood Cliffs, NJ.

Nilsson, N. J. (1998), *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann, San Mateo, CA.

Poole, D., A. Mackworth, and R. Goebel. (1997). *Computational Intelligence: A Logical Approach*. Oxford: Oxford University Press, Oxford, UK.

Russell, S.J., and P. Norvig. (1995), *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Shannon, C.E. *ed*. (1956), Automata Studies, Princeton University Press, Princeton.

Sternberg, R.J. (1997), The Concept of Intelligence and the its Role in Lifelong Learning and Success, *American Psychologist*, 52(10), 1030-1037.

Turing, A.M. (1950), Computing Machinery and Intelligence, *Mind*, 59, 433-460.

von Neumann, J. (1946), The Principles of Large-Scale Computing Machines, reprinted in *Annals of History of Computers*, 3(3), 263-273.

von Neumann, J. (1958), The Computer and the Brain, Yale Univ. Press, New Haven.

von Neumann, J. (1963), *General and Logical Theory of Automata*, A.H. Taub ed., Collected Works, Vol. 5, Pergamon, 288-328.

von Neumann, J. and A.W. Burks (1966), Theory of Self-Reproducing Automata, Univ. of Illinois Press, Urbana IL.

Wald, A. (1950), *Statistical Decision Functions*, John Wiley & Sons.

Wang, Y. (2002a), Keynote: On Cognitive Informatics, *Proc. 1st IEEE International Conference on Cognitive Informatics* (ICCI'02), Calgary, Canada, IEEE CS Press, August, 34-42.

Wang, Y. (2002b), The Real-Time Process Algebra (RTPA), *Annals of Software Engineering: An International Journal*, 14, USA, 235-274.

Wang, Y. (2003a), Cognitive Informatics: A New Transdisciplinary Research Field, *Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy*, 4(2), 115-127.

Wang, Y. (2003b), Keynote: Cognitive Informatics Models of Software Agent Systems, Proc. 1st International Conference on Agent-Based Technologies and Systems (ATS'03), Univ. of Calgary Press, Calgary, Canada, August, 25.

Wang, Y. (2004), Keynote: On Autonomic Computing and Cognitive Processes, *Proc. 3rd IEEE International Conference on Cognitive Informatics* (ICCI'04), Victoria, Canada, IEEE CS Press, August, 3-4.

Wang, Y. (2006), Keynote: *Cognitive Informatics - Towards the Future Generation Computers that Think and Feel*, *Proc. 5th IEEE International Conference on Cognitive Informatics* (ICCI'06), Beijing, China, IEEE CS Press, July, 3-7.

Wang, Y. (2007a), *Software Engineering Foundations: A Software Science Perspective*, CRC Book Series in Software Engineering, Vol. II, Aurebach Publications, NY., USA.

Wang, Y. (2007b), Keynote: Cognitive Informatics Foundations of Nature and Machine Intelligence,

Proc. 6th International Conference on Cognitive Informatics (ICCI'07), IEEE CS Press, Lake Tahoe, CA., Aug., 3-12.

Wang, Y. (2007c), The Theoretical Framework of Cognitive Informatics, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, 1(1), Jan., 1-27.

Wang, Y. (2007d), Exploring Machine Cognition Mechanisms for Autonomic Computing, *International Journal on Cognitive Informatics and Natural Intelligence*, March, 1(2), i - v.

Wang, Y. (2007e), The Cognitive Processes of Formal Inferences, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, Dec., 1(4), 75-86.

Wang, Y. (2008a), Keynote: On Abstract Intelligence and Its Denotational Mathematics Foundations, *Proc. 7th IEEE International Conference on Cognitive Informatics* (ICCI'08), Stanford University, CA., USA, IEEE CS Press, August, 5-15.

Wang, Y. (2008b), Toward a Generic Mathematical Model of Abstract Game Theories, *Transactions of Computational Science, 2*, Springer, June, 205-223.

Wang, Y. (2008c), On Contemporary Denotational Mathematics for Computational Intelligence, *Transactions of Computational Science, 2*, Springer, June, 6-29.

Wang, Y. (2008d), On Concept Algebra: A Denotational Mathematical Structure for Knowledge and Software Modeling, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, April, 2(2), 1-19.

Wang, Y. (2008e), On System Algebra: A Denotational Mathematical Structure for Abstract System modeling, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, April, 2(2), 20-42.

Wang, Y. (2008f), RTPA: A Denotational Mathematics for Manipulating Intelligent and Computational Behaviors, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, April, 2(2), 44-62.

Wang, Y. (2009). On Abstract Intelligence: Toward a Unified Theory of Natural, Artificial, Machinable, and Computational Intelligence, *International Journal of Software Science and Computational Intelligence*, IGI, USA, Jan., 1(1), 1-18.

Wang, Y. and G. Ruhe (2007), The Cognitive Process of Decision Making, *International Journal of Cognitive Informatics and Natural Intelligence*, IGI, USA, March, 1(2), 73-85.

Wang, Y., Y. Wang, S. Patel, and D. Patel (2006), A Layered Reference Model of the Brain (LRMB), *IEEE Trans. on Systems, Man, and Cybernetics (C)*, March, 36(2), 124-133.

Wellman, M.P. (1999), Multiagent Systems, in R.A. Wilson and C.K. Frank eds., *The MIT Encyclopedia of the Cognitive Sciences*, MIT Press, MA.

Widrow, B. and M.A. Lehr (1990), 30 Years of Adaptive Neural Networks: Perception, Madeline, and Backpropagation, *Proc. of the IEEE*, Sept., 78(9), 1415-1442.

Wittig, T. ed. (1992), *ARCHON: An Architecture for Multi-Agent Systems*, Ellis Horwood, London.

Wooldridge, M. and N. Jennings (1995), Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review* 10(2), 115-152.

Wooldridge, M. (2002), *An Introduction to Multiagent Systems*, John Wiley & Sons.

This work was previously published in the International Journal of Cognitive Informatics and Natural Intelligence, edited by Y. Wang, Volume 3, Issue 1, pp. 1-16, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 7.28
A Genetic Algorithm–Based QoS Analysis Tool for Reconfigurable Service-Oriented Systems

I-Ling Yen
University of Texas at Dallas, USA

Tong Gao
University of Texas at Dallas, USA

Hui Ma
University of Texas at Dallas, USA

ABSTRACT

Reconfigurability is an important requirement in many application systems. Many approaches have been proposed to achieve static/dynamic reconfigurability. Service-oriented architecture offers a certain degree of reconfigurability due to its support in dynamic composition. When system requirements change, new composition of services can be determined to satisfy the new requirements. However, analysis, especially QoS based analysis, is generally required to make appropriate service selections and service configurations. In this chapter, we discuss the development of QoS-based composition analysis techniques and propose a QoS specification model. The specification model facilitates QoS-based specification of the properties of the Web services and the requirements of the application systems. The composition analysis techniques can be used to analyze QoS tradeoffs and determine the best selections and configurations of the Web services. We develop a composition analysis framework and use the genetic algorithm in the framework for composition decision making. The framework currently supports SOA performance

analysis. The details of the genetic algorithm for the framework and the performance analysis techniques are discussed in this chapter.

INTRODUCTION

Reconfigurability is an important feature that is required in many modern application systems (Aksit & Choukair, 2003). For example, avionics systems, intelligent vehicle control systems, and remote monitoring systems frequently require dynamic adaptability so that the system can adapt to changes due to the failure of system components, reduced power level, unexpected operating conditions, and so forth. Also, some systems are multiple-mission and mission-specific, that is, they have to handle a class of missions that have similar system requirements but they also require adaptations to satisfy mission-specific needs.

Service-oriented architecture (SOA) is an ideal vehicle for achieving reconfigurability (Tsai, Song, Paul, Cao, & Huang, 2004). Desired functionalities can be achieved in SOA by dynamically composing appropriate services. Many research works focus on Web service composition (BEA et al., 2002; Hamadi & Benatallah, 2003; Sirin, Parsia, & Hendler, 2004). Most of these investigate various language issues. Business process execution language (BPEL) is an XML-based language for specifying the business processes and interaction protocols. Web services are then composed together to meet the specified functional requirements. The concept of semantic Webs extends the current Web with well-defined meanings for each Web service to facilitate their compositions. Based on semantic Webs, Sirin et al. (2004) have proposed a service composition tool including two main modules, namely, a composer and an inference engine. A user can interact with the composer to generate the composition and filter the results. The inference engine is a Web ontology language (OWL; W3C, 2004a) reasoner. The OWL reasoner searches the related services for the generated composition. Petri net-based algebra has also been used to help the composition by modeling the control flow. These composition methods can be used for adapting a system to new functional requirements (by composing a new set of services to achieve the modified functionalities).

Service composition generally focuses on composition of Web services to achieve some desired functionality. To achieve reconfiguration, new compositions can be derived for the modified functionalities. Actually, many adaptive systems require reconfigurability in terms of quality of service (QoS) behaviors. For example, some unmanned systems allow degraded QoS when some system failures occur. Many systems tradeoff the quality of their outputs versus the execution time to cope with periods of heavy loads or contentions for resources. In SOA-based systems, QoS reconfiguration can be achieved by, for example, selecting different services among those providing the same functionalities but, perhaps, having different QoS behaviors. Thus, systems can be dynamically assembled to fit the changing functional as well as QoS requirements.

Though SOA provides a convenient framework for reconfigurability, advanced techniques are still required to achieve actual adaptation (dynamic or static). For example, when the functional requirements of a system are modified, it is necessary to decompose the new requirements and then find the matching services in order to compose the new system. Similarly, when the QoS requirements of a system are modified due to changes in the execution environment, it is necessary to reconfigure the current services or choose new services to satisfy the new QoS requirements. To determine the correct service selections and configurations, it is necessary to analyze the QoS behaviors of the composed system.

The focus of this chapter is on QoS-based system reconfiguration. The goal is to develop analysis techniques and tools to facilitate static and dynamic system QoS behavior analysis such

that correct decisions regarding service selections and configurations can be made. The next section, *SOA Background*, introduces the SOA concept. The *QoS Specification Model* section presents the QoS specification model, and the *Analysis of Adaptive SOA Systems* section describes the analysis process of adaptive SOA systems. Then the composition analysis framework and genetic algorithm-based decision support are presented in the *Composition Analysis Framework* and the *Genetic-Algorithm-Based Decision Support* sections, respectively. And, the associated performance analysis method for SOA system is introduced the *Performance Analysis* section. The next section after that is *Case Study*, and finally, the *Conclusion* section summarizes this chapter.

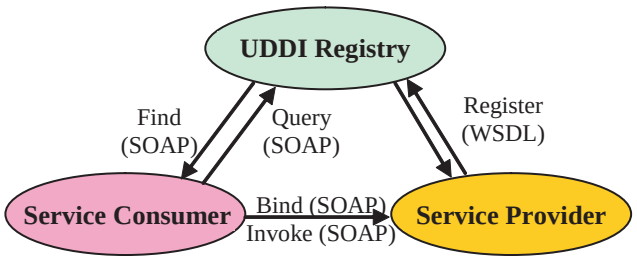
SOA BACKGROUND

The SOA concept is probably one of the most significant software engineering concepts after the currently widely adopted object-oriented technologies. The SOA term expresses the software architectural concept and essentially a collection of services. In SOA, each service encapsulates certain information and data to implement the functionality, and such service can then be consumed by clients in other applications through the exposed interfaces. These services communicate with each other to achieve the system requirements. SOA is an architecture style which is distributed, loosely coupling, highly interoperable between the services, hence, will maximize system flexibility and independent evolvability. Also, due to the potential of reuse and dynamic composition of core services, SOA offers high system reusability and reconfigurability. Legacy systems can be wrapped and reused without significant changes, while new applications can be incorporated into the architecture easily.

Web service-based SOA is a specialized realization of SOA (W3C, 2002). In this specific SOA, Web service (WS) takes the concept of the service and delivers the service over the Web using the technologies such as XML, simple object access protocol (SOAP), Web services description language (WSDL), and universal description, discovery, and integration (UDDI). SOAP provides a way to communicate between the services which are implemented in different programming languages, different technologies, and run on different operating systems. WSDL provides a standard format to describe the Web services by using XML. UDDI is a repository which stores the information of the Web services and facilitates publishing and finding of the Web services.

Web service-based SOA adopts register-find-bind-invoke paradigm which is shown in Figure 1. In this paradigm, service provider registers the

Figure 1. Web services system model



Web services in the public UDDI registry and publishs the Web services. Service consumer searches in the public UDDI registry and finds the desired Web services, then binds and invokes it.

Web service composition enable user to rapidly build a system or create a process by assembling individual Web services. Through specifying some details of the composition, such as when to invoke the appropriate Web services, the calling sequence of the Web services, the developer can implement the needed tasks. To facilitate the composition, several Web service composition languages are proposed, including business process execution language for Web services (BPEL4WS, also called BPEL; BEA Systems et al., 2002) and OWL-based Web Service Ontology (OWL-S; W3C, 2004b).

BPEL provides an abstract business protocol to facilitate the specification of compositions. A BPEL process specifies a set of partners and their interactions and invocation sequence. Each partner is a unit in the process and it is linked to the process via a partner link. The BPEL process coordinates with all involved clients and Web services and interacts with them, coordinates the execution sequence to achieve the desired goal. The process specification is kept at the abstract level, and it only specifies the relations and exchanged messages between partners without considering the internal details of the partner. With certain extensions, the BPEL process can be used to flexibly specify service composition and allows each partner to be associated with alternate Web services.

OWL-S provides a set of markup language constructs for describing the properties and capabilities of the Web services. It facilitates the automated Web service discovery, composition, and execution. It supports similar features as BPEL does for composition specification. OWL-S is widely used in semantic Web community while BPEL is used commonly in business sectors.

QOS SPECIFICATION MODEL

The QoS behavior of a WS-based system can be adapted due to three factors, including: (1) resource allocation, (2) selection of different services that offer the same functionalities, and (3) tuning QoS control parameters within the services. First, for resource allocation, we consider allocating available physical platforms to involved services. Services with higher access rates in the system may be replicated on multiple platforms while multiple services with lower access rates may share a single platform. Second, different services implementing the same functionality may yield different QoS tradeoffs. Thus, appropriate selection of the services can help with achieving the QoS goals of the applications. Third, each service may be reconfigurable in terms of its QoS behavior. Some QoS control parameters within a service unit can be tuned to control the tradeoffs of the QoS behavior. For example, a search program may obtain better results when given more time and the tradeoffs between time and output quality can be controlled by certain parameters within the program. Here, all three types of adaptivity for QoS analysis and reconfiguration in SOA systems are considered.

The goal of QoS-based composition analysis for reconfigurable SOA systems is to determine the best configuration for the system based on system QoS requirements. The configuration should consider the three adaptation factors, resource allocation, Web service selection, and QoS control parameter determination. To achieve QoS-based composition analysis, it is necessary to first establish the model for QoS related specifications. For example, the QoS behavior of the application system is derived from the QoS behavior of individual Web services. Thus, QoS behavior of the Web services to be considered for composing the application systems should be known a priori to facilitate the analysis. Also, it is necessary to specify the QoS requirements of

the application systems and based on which the satisfatory system configuration can be determined. In this section, we step-by-step introduce the model for QoS specification.

QoS Attribute Vector

To support all QoS-related specifications, the first step is to define the set of QoS attributes that need to be measured. Some standard QoS attributes include time and resource consumption metrics. Some application-specific QoS attributes include, for example, picture quality for image compression algorithms, precision of a relaxation based computation, and so forth. Consider a system S. Let $Q_S = (q_1, q_2, ..., q_N)$ denote the quality attribute vector of S, where Q_S includes N quality attributes. For different Web services, different quality attribute vectors may be considered. When specifying the QoS requirements of S, Q_S should be defined first. Note that the quality attribute vectors of the Web services that compose S may not have exactly the same elements, and they may be different from Q_S. When performing composition analysis, the quality attributes that are of interests are the overall system quality attributes (which is Q_S). If a Web service has additional quality attributes, they can simply be ignored. If a Web service does not contribute to a certain quality attribute in Q_S, then, a *null* value can be assigned to it.

System QoS Requirements

Let Γ_S denote the system QoS requirements for system S. Γ_S is defined on Q_S. Generally, system QoS requirements can be in the form of constraints that has to be satisfied and/or objectives that should be optimized. Let $\Gamma_S = (O, R)$, where $O = (o_1, o_2, \ldots)$ is the set of objectives, and $R = (r_1, r_2, \ldots)$ is the set of constraints. An objective o_i is an optimization function defined on one or more QoS attributes in Q_S. For example, maximize(q_1) and minimize ($aq_2 + bq_3$) are possible QoS objectives. A constraint r_i specifies a specific QoS constraint that need to be satisfied, and it is also defined on one or more QoS attributes in Q_S. For example, r_i may specify that the power consumption of the system should be within 100 walts per unit time, denoted as r_i: $q_3 \leq 100$ walts/sec, where q_3 is the power consumption quality attribute.

The goal of the system is to satisfy all the QoS constraints while obtaining optimal solutions in terms of the QoS objectives. Since there may be multiple objectives, the Pareto-optimal solutions can be considered.

Service Composition

The application system S is composed of Web services. We can use BPEL to specify the composition of services. In order to specify the functional requirements and associated candidate Web services of the partner, we extend the definition of the "partner" in BPEL by adding the tags "functional_req" and "associated_ws" respectively. Let Ω_S denote the composition specification for system S. Ω_S consists of a set of partners, $\{\omega_i \mid 1 \leq i \leq M\}$. The functionality of each partner ω_i is specified using the "functional_req" tag. From the functional specification, the actual Web services can be selected to instantiate a partner. Partner ω_i can be instantiated by m_i Web services, $ws_{i,j} \mid 1 \leq j \leq m_i$, which are included in the "associated_ws" tag of ω_i.

QoS Properties of the Individual Web Services

To facilitate QoS analysis and system configuration decision making, it is necessary to know the QoS properties of individual Web services and how to compose them to derive the QoS behavior of the composed system. The QoS property of a Web service is generally a function of the input and the operation environment parameters. For a reconfigurable Web service, the control parameters

in the Web service program also impact its QoS property. Consider the candidate Web services, $ws_{i,j}$, for all i and j, for composing the system S. Let $I_{i,j}^{ws}$ denote the set of input parameters, $E_{i,j}^{ws}$ denote the set of operation environment parameters, and $X_{i,j}^{ws}$ denote the set of configurable parameters for $ws_{i,j}$. $I_{i,j}^{ws}$ is specified by the parameters in client requests. $E_{i,j}^{ws}$ can be, for example, the platform capability. For a given set of $I_{i,j}^{ws}$, $E_{i,j}^{ws}$, and $X_{i,j}^{ws}$ values, the properties of $ws_{i,j}$ in terms of quality attribute q_i can be measured. Let ${}^{*}V_{i,j}^{ws} = ({}^{1}V_{i,j}^{ws}, {}^{2}V_{i,j}^{ws}, \ldots, {}^{N}V_{i,j}^{ws})$ denote the QoS properties of $ws_{i,j}$, where ${}^{k}V_{i,j}^{ws}$ is the QoS property of $ws_{i,j}$ in terms quality attribute q_k. ${}^{k}V_{i,j}^{ws}$ is a function of $I_{i,j}^{ws}$, $E_{i,j}^{ws}$, and $X_{i,j}^{ws}$, i.e., ${}^{k}V_{i,j}^{ws}$ can be expressed as ${}^{k}V_{i,j}^{ws}(I_{i,j}^{ws}, E_{i,j}^{ws}, X_{i,j}^{ws})$. ${}^{k}V_{i,j}^{ws}$ is actually a set of QoS measurement data collected based on various $I_{i,j}^{ws}$, $E_{i,j}^{ws}$, and $X_{i,j}^{ws}$ or a function representing the QoS measurement data. In this chapter, we assume that the measurement data can be fitted to a function.

A configurable parameter is a parameter in a Web service that, when adjusted, can impact the measurements of one or more of the QoS attributes. Different QoS properties of a Web service and the composed system can be obtained by tuning the configurable parameters. For example, the iteration number in a relaxation algorithm can be the control parameter to balance the tradeoffs between execution time and computation precision. The set of configurable parameters $X_{i,j} = (x_{i,j}^{1}, x_{i,j}^{2}, \ldots)$ is the tunable parameters for Web service $ws_{i,j}$ and it includes $\chi_{i,j}$ parameters.

QoS Property Composition

As discussed earlier, how to compose QoS properties of the individual Web services to derive QoS behavior of the composed system is also a necessary entity in QoS-based composition analysis and service configuration decision making. However, the model for QoS property composition depends on the quality attributes and a specific property composition algorithm needs to be defined for each individual quality attribute. The details regarding the QoS property composition algorithms will be discussed in details in the *Genetic Algorithm-Based Decision Support* section.

Resource Specification

Resource allocation can have a significant impact on QoS behavior of the system. QoS properties of each Web service depends on the operation environment $E_{i,j}^{ws}$, which, in turn, depends on the resources allocated to the Web service. In this chapter, the resource allocation problem only considers the allocation of Web services to hardware platforms. Also, heterogeneous platforms are assumed. Multiple platforms may be allocated to a heavily loaded Web service. Let $RS = \{rs_1, rs_2, \ldots, rs_L\}$ denote the set of L resources (platforms) in the system. The configuration decision maker needs to allocate the L platforms to the M Web services that are selected for the M partners.

ANALYSIS OF ADAPTIVE SOA SYSTEMS

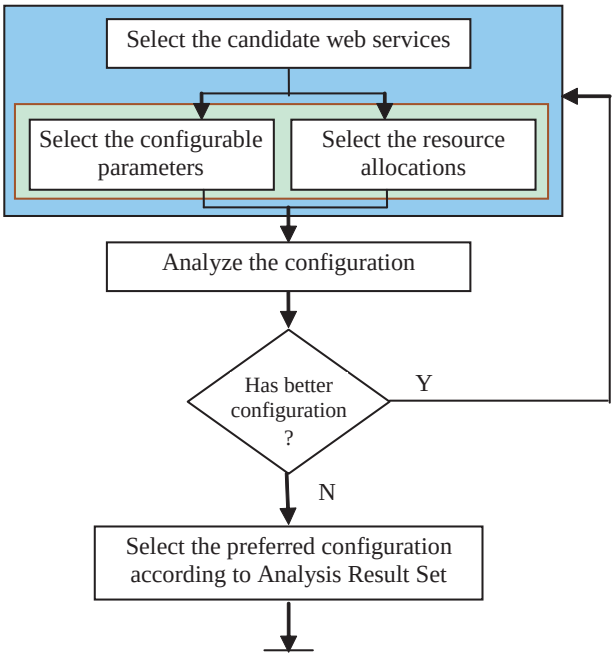
The major goal for QoS-based composition analysis process is to determine the best composition of Web services, their configurations, and mapping of the services to hardware platforms. A general analysis process is illustrated in Figure 2.

In this QoS-based composition analysis process, the candidate configuration is first determined and then analyzed. After analysis, either some potentially better configurations are further considered or the best candidate configuration is selected. More details of each of the steps is given in the following.

Select the Candidate Web Services

Each partner ω_i may be instantiated by multiple candidate Web services $ws_{i,j}$, $1 \le j \le m_i$. In this

Figure 2. QoS analysis process for adaptive SOA systems



step, one of the Web services need to be selected for each partner in the system.

Select the Configurable Parameters

Some of the Web services $ws_{i,j}$ may be reconfigurable and has a set of configurable parameters $X_{i,j}^{ws} = (x_{i,j}^{1}, x_{i,j}^{2}, \ldots)$. Each configurable parameter has a value range. In this step, a specific value in the value range is selected for each configurable parameter.

Select the Resource Allocations

In this step, a specific resource allocation that maps the Web services to the platfoms in *RS* is determined.

Analyze the Configuration

After selecting the specific Web services for each partner, setting each of the configurable parameters, and allocating of the platforms to Web services, the configuration of the system is fixed. Based on the configuration and the component properties of the selected Web services, a QoS property composition algorithm is applied to derive the overall system behavior. The result is then evaluated to determine whether it is satisfatory.

Select the Preferred Configurations

If the QoS behavior of the composed system is not satisfactory, then new configurations are

generated and evaluated repeatedly. The process terminates either after a number of iterations or after a satisfactory solution is selected. The final solution is then used for system configuration.

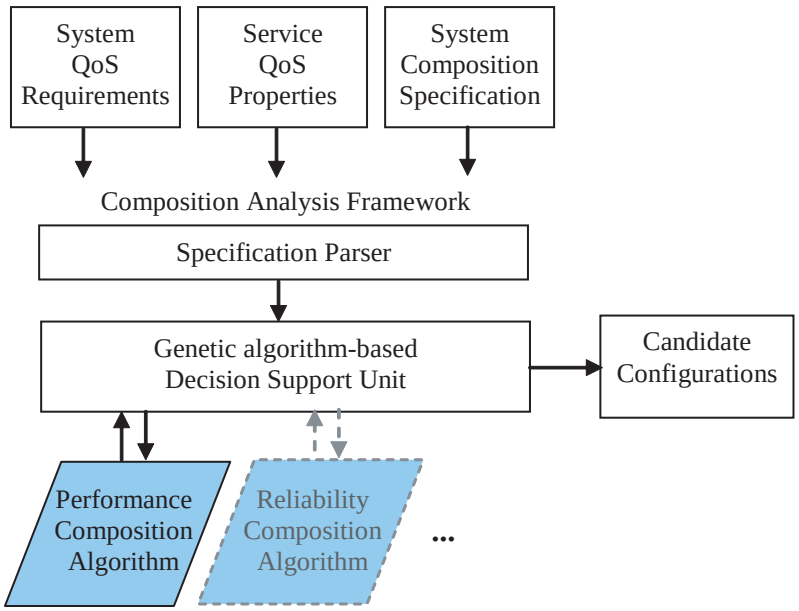
THE COMPOSITION ANALYSIS FRAMEWORK

The goal of the composition analysis process is to find Pareto-optimal configurations for the given SOA system specifications. A composition analysis framework has been developed to realize the QoS analysis process. The architecture of the framework is shown in Figure 3. The framework includes a specfication parser, a genetic-algorithm based decision support unit, and various composition analysis algorithms. The specification parser parses the "system composition specification", "system QoS requirements", and "service QoS properties". The parsed information is integrated and sent to the genetic-algorithm based decision support unit. The genetic-algorithm based decision support unit provides efficient composition decision making.

The genetic algorithm-based decision support unit is responsible for configuration selection, including the steps "select the candidate Web services", "select the configurable parameters", and "select the resource allocations". The selected configurations are passed to the framework. Then the framework invokes the appropriate QoS composition analysis algorithm and pass the selected configuration to it for analysis. The analysis results are then passed back to the decision support unit and the decision support unit either continues to generate new candidate configurations or selects the configuration as the final decision.

QoS composition analysis process may involve the analysis of various QoS attributes, such as

Figure 3. The architecture of the composition analysis framework

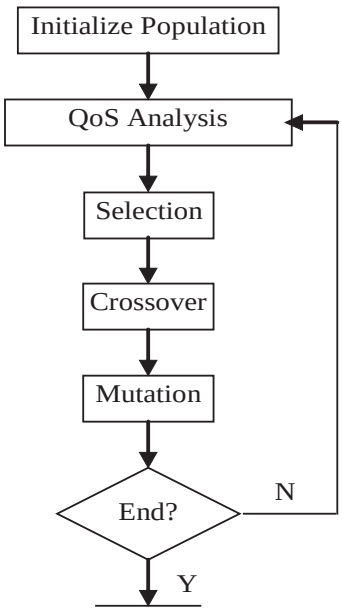


performance, reliability, and so forth. Since the analysis of different QoS attributes requires different methods and the QoS analysis algorithms are highly application-dependent, the composition analysis framework provides a registry table to allow users to plug-in application-specific analysis algorithms. The user can register a QoS property composition algorithm to the framework through the publicly exposed interface. Currently, the algorithm for performance analysis of the SOA is available.

GENETIC ALGORITHM-BASED DECISION SUPPORT

We use a genetic algorithm-based process for the decision support unit in the composition analysis framework. The genetic algorithm is responsible for candidate configuration generation and optimal decision making. An outline of the genetic algorithms is shown in Figure 4. It begins with the initialization phase which randomly selects a set of individuals to form the initial population. An individual represents a solution or, for our specific problem, a configuration. A population, in turn, is a group of individuals. Each individual is composed of a sequence of genes. For our problem, the genes can represent a configurable parameter, selection of a Web service for a partner, and a specific allocation of platforms. The initial population is called the first generation. The individuals in the population are evaluated using a fitness function. For example, in the model discussed in the *QoS Specification Model* section, the performance and/or other quality attributes are measured to evaluate the fitness of the configurations. Based on the evaluation results, the generic "selection" operator is used to select the better individuals as parents. From the parents,

Figure 4. Genetic algorithm outline



the "crossover" and "mutation" operators are applied to generate a new generation of population. The process is repeated until a satisfactory set of solutions are obtained.

In the next sub-section, we introduce the background for multi-objective evolutionary algorithms. The mapping of the composition analysis process to the evolutionary algorithm paradigm is discussed in the *Mapping the Composition Analysis Problem to the Genetic Algorithm* sub-section. The detailed algorithms are discussed in the *Genetic Algorithm for Composition Analysis* sub-section.

Multi-Objective Evolutionary Algorithms (MOEA)

The composition analysis process has some difficulties, including the conflicting objectives, the exponential search space, and a mix of continuous and discrete configurable parameters of the Web services. The classical search algorithms, such as linear programming and gradient search, are not efficient for the multi-objective problems (Deb, 1995). Randomized algorithm can be used to deal with the problem such as local minimal trap but it is very inefficient. The genetic algorithm is a partly randomized exploratory procedure based on biological evolution. Specifically, the multi-objective evolutionary algorithms (MOEA) is a specialized genetic algorithm commonly used to find the Pareto-optimal solutions.

The main objective of the MOEAs is to quickly converge to the true Pareto-optimal front with a widely spread solution set. Initial ones, such as MOGA (Fonseca & Fleming, 1993), non-dominated sorting genetic algorithm (NSGA) (Srinivas & Deb, 1994), and niched Pareto genetic algorithm (NPGA) (Hamadi et al., 2003) realize this objective using non-dominated sorting along with a niching mechanism. These algorithms have shown some successes in finding Pareto optimal solutions. In Zitzler, Deb, and Thiele (2000), the elitism concept is introduced and has been shown to have significant impact on performance improvement. Elitism is a strategy to preserve the better solutions for new population generation. Many subsequent algorithms, for example, NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002), Pareto envelope-based selection (PESA) (Corne, Knowles, & Oates, 2000), and strength Pareto evolutionary algorithm 2 (SPEA2) (Zitzler, Laumanns, & Thiele, 2002) apply the elitism strategy and demonstrate substantial performance improvements.

In Deb et al. (2002), an elitism-based algorithm, NSGA-II, along with improvements in diversity preservation and constraint handling, has been proposed. It has been shown that the algorithm outperforms Pareto archived evolution strategy (PAES) and SPEA (Zitzler & Thiele, 1999) in terms of finding a diverse set of solutions and converging toward the true Pareto-optimal set. Khare, Yao, and Deb (2003) compared the scalability of NSGA-II, PESA, and SPEA2 with respect to the number of objectives (2 to 8). PESA was shown to be the best in terms of converging to the Pareto-optimal front, but it does not have good diversity maintenance. Both SPEA2 and NSGA-II have good performance in terms of convergence and diversity maintenance, but NSGA-II runs much faster than SPEA2. Thus, NSGA-II is chosen to be the algorithm for the composition analysis process, more specifically, to generate and select promising system configurations. Some modifications are made to adapt NSGA-II to the composition analysis problem and to achieve problem-specific improvements.

Mapping the Composition Analysis Problem to the Genetic Algorithm

To map the composition analysis problem to the genetic algorithm flow, it is necessary to first determine the individuals and genes. In the *Genes and Individuals for Configuration Representation* sub-section, the design of individuals and genes for the composition analysis problem is given. To evaluate the individuals, the fitness function

should be defined next. The fitness function is discussed in the *QoS Analysis Algorithms* subsection.

Genes and Individuals for Configuration Representation

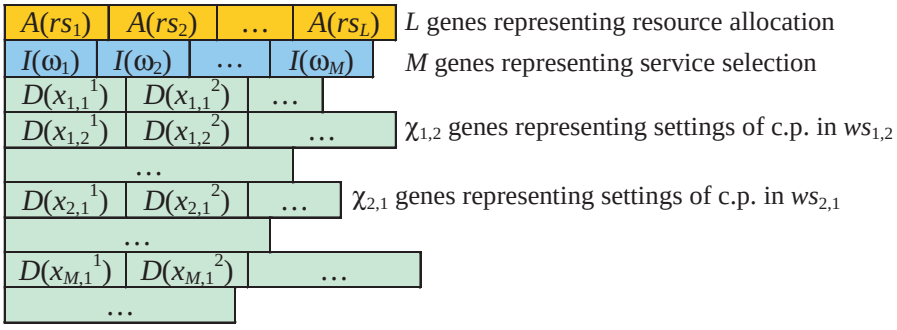
QoS-based SOA composition process supports adaptation in terms of resource allocation, Web service selection, and configurable parameter configuration within services. Thus, each individual includes three types of genes: one represents resource allocation, one represents service selection, and one represents configurable parameter setting. Figure 5 illustrates the different types of genes.

One individual includes L genes which comprise the allocation representations for the L resources in RS. Each gene represents the allocation of a resource rs_i, denoted as $A(rs_i)$. $A(rs_i)$ is an integer value from 1 to M. $A(rs_i) = k$ denotes that rs_i is allocated to the Web service instantiating partner ω_k. It is assumed that resources are independent of the Web services and assumed that these resources can be allocated to any Web services, that is, Web services are not resource dependent. Also, it is assumed that each resource can only be allocated to a single Web service. Hence, the number of resource, L, is greater than or equal to the number of partners, M.

For each partner, a gene is used to represent the selection of the instantiating Web service and there are a total of M such genes. Let $I(\omega_i)$ denote the gene for service selection. Since ω_i can be instantiated by Web services $ws_{i,j}$, $1 \le j \le m_i$, $I(\omega_i)$ is the index of the Web service selected for instantiating ω_i and its value range is from 1 to m_i. In other words, if $I(\omega_i) = j$, then $ws_{i,j}$ is selected to realize ω_i.

For the configurable parameters within individual services, each gene, denoted as $D(x_{i,j}^{k})$, represents a setting of the corresponding configurable parameter $x_{i,j}^{k}$, where $x_{i,j}^{k}$ is a tunable parameter in Web service $ws_{i,j}$. For $ws_{i,j}$, there are $\chi_{i,j}$ configurable parameters. Thus, there are a total of $\sum_{i,j} \chi_{i,j}$ genes representing configurable parameter settings. Each $D(x_{i,j}^{k})$ has its own data type and value range. The data type and value range definitions for $x_{i,j}^{k}$ are defined in the Web service QoS property specification and can be retrieved to define $D(x_{i,j}^{k})$.

Figure 5. Mapping of the system configurations



QoS Analysis Algorithms

The individuals in a population are evaluated using the fitness functions. The fitness functions are defined and realized by QoS analysis algorithms. Essentially, each individual represents a specific configuration of the system (selection and configuration of the Web services and resource allocation for them). Based on the configuration, the overall system properties, such as performance, reliability, and so forth, are determined. In this chapter, we use performance metric as an example to illustrate the QoS Analysis. The performance analysis for the SOA systems is discussed later in the *Performance Analysis* section.

Genetic Algorithm for Composition Analysis

In this section, we discuss the detailed algorithms in the GA paradigm. The overall genetic algorithm is discussed in the *NSGAII-CAA* sub-section. The GA operations, including selection, crossover, and mutation, are discussed in *Selection Operator*, *Mutation Operator*, and *Crossover Operator* sub-sections, respectively.

NSGAII-CAA

The algorithm NSGAII-CAA (non-dominated sorting genetic algorithm II – composition analysis algorithm) is a modified version of NSGA-II. It is specifically adapted for the composition analysis process with some performance improvement features. The algorithm is shown in the following.

Algorithm 1 follows closely the general genetic algorithm flow. In the following sub-sections, the major steps specifically designed for the composition analysis process are discussed.

Algorithm 1.

Algorithm NSGAII-CAA

Input: N_s: population and initial elite set size
D: generation number
Output: Pr: result non-dominated set

1 Initialize population P_0
2 QoS analysis on P_0
3 Create empty elite set P_0'
4 $d := 0$
5 while ($d < D$) do
6 generate a new elite set $P_{d+1}' :=$ NSGAII-crowding-pick (P_d, P_d', N_s)
7 generate the new population $P_{d+1} :=$ Mutation(Recombination(BTS(N_s, P_{d+1}')))
8 QoS analysis on P_{d+1}
9 $d := d + 1$
10 end while
11 $Pr :=$ non-dominated-set (P_d')

Selection Operator

For selecting promising configurations in a population, a comparision standard has to be provided. NSGA-II uses the crowding pick approach (NSGAII-crowding-pick). First, each individual is assigned a priority using non-dominated sort. Then, the individuals are further sorted using the crowding distance sort. Non-dominated sort first identifies the non-dominated solutions and gives them the highest priority and, after priority assignment, moves them out of the set. Then the non-dominated solutions of the remaining individuals are identified and given the second highest priority. This process continues until priorities are assigned to all the individuals. Based on each objective, the crowding distance computation first sorts the individuals with the same priority according to their values. The distances are normalized cross objectives by equalizing the maximum distances (between the highest and lowest values) of the objectives and proportionately adjusting the distances of all pairs of adjacent individuals. For each individual, the distances for all the objectives are summed together. The individual with a larger distance value gets a higher priority.

Based on the priorities, the best individuals can be picked to form the elite set, which stores a fixed number of best individuals ever generated and serves as the mating pool for next generation. Initially, the elite set is empty. For every generation, the population and the elite set from the previous generation are merged into one set. N_s individuals with the highest priorities are picked from the set to form the new elite set, where N_s is the population size. For a J-objective problem with population size N_s, NSGA-II has a storage requirement of $O(N_s^2)$ and time complexity of $O(J N_s^2)$.

The NSGAII-CAA algorithm uses a binary tournament selection (BTS) algorithm to select N_s individuals from the elite set to generate a new set, from which the new generation can be

Algorithm 2.

Algorithm BTS

Input: N_s: population size
P: elite set
Output: P': result new population

```
1    i := 0
2    while (i < N_s) do
3        randomly pick up two configurations s_1, s_2 from P
4        if s_1 has higher priority
5                put s_1 in P'
6        else
7                put s_2 in P'
8         i := i + 1
9    end while
```

produced by recombination and mutation. Each time two individuals are randomly picked from the elite set and one individual with the higher priority is selected. It continues till N_s individuals are selected. The complete BTS algorithm (Algorithm 2) is shown as follows.

Mutation Operator

In a mutation process, a new individual is generated from a randomly picked configuration by slightly modifying it. The probability of mutation of one individual is controlled by a mutation rate, which is normally set as $1/N_s$ where N_s is the population size. In the case study to be discussed later, the mutation rate is set as 0.01. For the individual to be mutated, the mutation point is chosen randomly and the gene of the mutation point is modified. For our specific problem, a random value in a pre-determined range of the chosen gene is used to replace the existing value. Each configurable parameter should have a fixed range so that a random value can be selected for the corresponding gene.

Crossover Operator

While mutation generates a new individual from one parent, crossover process exchanges the genes of one or more parents to reproduce new individuals. We use one-point recombination due to its effectiveness and simplicity. Similar to the mutation process, the crossover point for recombination of two individuals is generated randomly. The one-point recombination process exchanges the genes of two parents on and after the crossover point to reproduce two offsprings.

PERFORMANCE ANALYSIS

We use the Markov chain model to assess the performance for the SOA-based system (Bose, 2001). The SOA-based system consists of multiple Web services that may invoke one another (assume that there is no cyclic invocation). So the system is modeled as a multi-tier client-server system, which include multiple levels of client-server subsystems. In the following sub-sections, we first review the Markov model for performance analysis in the *Markov Model for Performance Analysis* sub-section. Then, in the *Performance Analysis for SOA Systems* sub-section, the method used for performance analysis for SOA system is introduced. Finally, in *A Performance Analysis Example* sub-section, an example is given to illustrate the performance analysis method.

Markov Model for Performance Analysis

Consider a simple client-server system. Let q denote the server queue. Assume that the server has service time t_s and request arrival rate λ. Also, let n denote the average population at the server. From the Markov model, we have (Bose, 2001):

$$\text{Utilization: } \rho = \lambda \cdot t_s,$$
$$\text{Average response time: } t = \frac{t_s}{(1-\rho)},$$
$$\text{Average population: } n = \frac{\rho}{(1-\rho)}. \quad (1)$$

If the load on the server is very high, the server may become a bottleneck. To deal with this problem, a server may be replicated, that is, multiple resources are allocated to that server. In this case, we use the *M*/*M*/*m* queuing model to compute the system performance, which has the following performance characteristics (Bose, 2001).

$$\text{Utilization: } \rho' = \frac{\lambda \cdot t_s}{m},$$
$$\text{Average response time: } t' = t_s + \frac{P_Q}{(m/t_s - \lambda)}, \quad (2)$$

where P_Q denotes the probability that all m servers are busy and $p(0)$ denotes the probability that the queue length is 0. We have

$$P_Q = (m \cdot \rho')^m \frac{p(0)}{m!} \cdot \frac{1}{1-\rho'},$$

$$p(0) = \left[\frac{(m \cdot \rho')^m}{m! \cdot (1-\rho')} + \left(\sum_{i=0}^{m-1} \frac{(m \cdot \rho')^i}{i!} \right) \right]^{-1}$$

Average population: $n' = P_Q \cdot \frac{\rho'}{(1-\rho')}$

Performance Analysis for SOA Systems

In the *Markov Model for Performance Analysis* sub-section, it is assumed that the arrival rates for the input ports of each Web service are known. When one Web service uses services provided by other Web services, it forms the client-server relationship. A Web service $ws_{1,1}$, for some requests, may invoke another Web service $ws_{2,1}$, but for some other requests, may invoke Web service $ws_{3,1}$, and for yet some other requests, may simply serve them locally. Also, a request may be propagated through several layers of service invocation. Thus, each request has a service path. To facilitate performance analysis, it is necessary to compute the effective arrival rate for each Web service in the system, and the computation is based on the invocation paths and the arrival rates from the real clients.

The SOA composition specification specifies the service architecture, including the partners and the links. The QoS specification for the individual services includes several essential parts: (1) for each partner, the arrival rates for its input ports should be specified if it takes input from real clients; (2) for each partner, the distribution of client requests that are forwarded to each of the output ports should also be specified; and (3) for each Web service, the service rate should be specified.

We assume that the client arrival rates follows a Poisson distribution. Also, we assume that the service rate of each Web service has an exponential distribution or is constant. In addition, assume that the arrival rate λ_i for partner ω_i is less than the service rate $1/t_{i,j}$ of the Web service $ws_{i,j}$ which instantiates ω_i. Thus, the output from each partner follows a Poisson distribution with rate λ_i. We also assume that if ω_i, after processing the requests, forwards the requests to other partners, then the probability for the requests going to different partners follows a uniform random distribution. Consider the case that partner ω_i has a request arrival rate λ_i and the probability that ω_i forwards a request to ω_j is $p_{i,j}$. Then, the effective arrival rate for partner ω_j has a Possion distribution with average arrival rate $\lambda_i * p_{i,j}$. From the effective arrival rate, the response time for each Web service can be computed based on the Markov model discussed in the *Markov Model for Performance Analysis* sub-section.

To compute the response time of client requests in terms of the overall system, we need to consider the individual paths of the requests flowing through the service hierarchy. First, according to the effective arrival rate, the response time for each service can be computed. For each path, the response time of the requests can be computed by adding together the response times of the multiple tiers that a request has to go through. The average response time of the overall system is the expected value of the response times of all the paths in the service architecture. The algorithm of computing the average response time is shown in Algorithms 3 and 4.

In the SOA based system, the link delay also affects the performance of the system. A link can be viewed as a network service unit. In the QoS property specifications of the individual services, the link delay can be specified either as a fixed latency or a specific distribution.

Algorithm 3.

Algorithm computeTotalAverageResponseTime

Input: λ : Initial arrival rate of the system
Output: T : TotoalAverageResponseTime

1 Find the starting web service $ws_{i,j}$ of the system
2 recursivePerformanceAnalysis ($ws_{i,j}$, λ)

Algorithm 4.

Algorithm recursivePerformanceAnalysis

$N_{i,j}$: The number of the partners which Web Service $ws_{i,j}$ sends request to
The requests sent by $ws_{i,j}$ are indexed into 0 to $N_{i,j}$–1

Input: $ws_{i,j}$: Web Service to be processed
λ_i : The effective arrival rate of $ws_{i,j}$
Output: t : The response time of $ws_{i,j}$

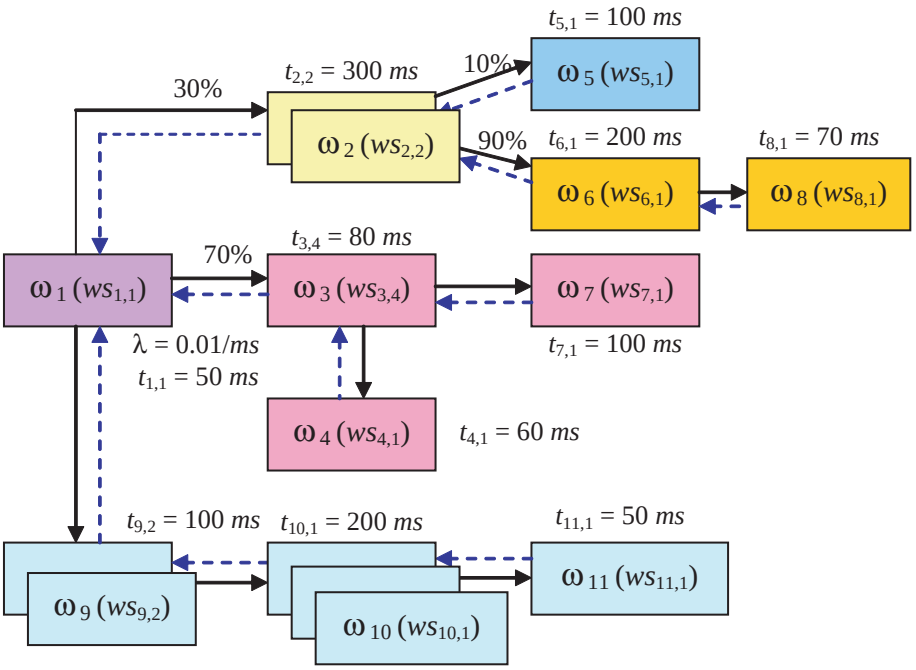
1 if (Web Service $ws_{i,j}$ is empty)
2 return
3 if (Web Service $ws_{i,j}$ isn't replicated)
4 Compute the response time $t_{i,j}$ of $ws_{i,j}$ using $M/M/1$ formula (1)
5 else
6 Compute the response time $t_{i,j}$ of $ws_{i,j}$ using $M/M/m$ formula (2)
7 T += (λ_i /λ) * t
8 j := 0
9 while ($j < N_{i,j}$)
10 Find the Web Service $ws_{j,k}$ which the request j goes
11 Compute the effective arrival rate λ_j of $ws_{j,k}$
12 recursivePerformanceAnalysis($ws_{j,k}$, λ_j)
13 endwhile

A Performance Analysis Example

Here, an example is given to illustrate the performance analysis algorithm.

The example system is shown in Figure 6. The performance analysis is based on a given system configuration, thus, each partner in the system has been instantiated by certain Web service which is specified in Figure 6, such as ω_1 is instantiated by $ws_{1,1}$. The specification for each Web service, partner, and link are also given in Figure 6. For example, the arrival rate λ

Figure 6. An example system for performance analysis



of partner ω_1 is 0.01/*ms* and the service time $t_{1,1}$ for Web service $ws_{1,1}$ (instantiates ω_1) is 50 *ms*. Also, from Figure 6, we can see that the service flow is initiated at ω_1. Requests from partner ω_1 are forwarded to different partners, where 70% are forwarded to partner ω_3 and 30% to ω_2. After receiving the responses from partner ω_2 and ω_3, partner ω_1 sends the requests to partner ω_9 .The possible paths for the requests include ω_1-ω_2-ω_5-ω_9-ω_{10}-ω_{11}, ω_1-ω_2-ω_6-ω_8-ω_9-ω_{10}-ω_{11}, etc. Here, we assume that the transmission time of the requests and responses between any two partners is a constant value, namely, 20 *ms*.

First, we compute the response time for path ω_1-ω_3-ω_7-ω_4. From the properties of partner ω_1 and partner ω_3 in Figure 6, the effective arrival rate of partner ω_3 from partner ω_1 can be computed as 0.7*λ. The requests are processed by Web service $ws_{3,4}$ (instantiates ω_3) with the service time $t_{3,4}$ = 80 *ms*. Then requests from partner ω_3 are forwarded to partner ω_7 and gets processed by Web service $ws_{7,1}$ (instantiates ω_7). Because the requests from partner ω_3 are all forwarded to partner ω_7, the effective arrival rate is the same as partner ω_3, namely, 0.7*λ. After receiving the responses from partner ω_7, partner ω_3 sends the requests to partner ω_4. Similarly, the effective arrival rate of partner ω_4 is 0.7*λ. Based on the effective arrival rate, the response time of Web service $ws_{3,4}$ is computed as 182 *ms*. Similarly, the response times of Web service $ws_{7,1}$ and $ws_{4,1}$ are 333 *ms* and 103 *ms*, respectively.

Then look at path ω_1-ω_2-ω_5 and ω_1-ω_2-ω_6-ω_8. As can be seen, Web service $ws_{2,2}$ (instantiates

ω_2) is replicated. It has an arrival rate 0.3*λ and the service time $t_{3,2}$ for Web service $ws_{2,2}$ is 300 *ms*. Since Web service $ws_{2,2}$ is replicated by 2, we compute the response time of Web service $ws_{2,2}$ using *M*/*M*/*m* (here *m* = 2) queuing model. Then, 10% of the requests from partner ω_2 are forwarded to partner ω_5 and 90% to partner ω_6. So the effective arrival rates of partners ω_5 and ω_6 are 0.1*0.3*λ and 0.9*0.3*λ. The requests forwarded to partner ω_6 are processed further by Web service $ws_{8,1}$ (instantiates ω_8). Thus, the response times of all Web services ($ws_{2,2}$, $ws_{5,1}$, $ws_{6,1}$, $ws_{8,1}$) are computed and they are 423 *ms*, 103 *ms*, 435 *ms*, and 86 *ms* respectively.

After receiving the responses from partner ω_2 or partner ω_3, partner ω_1 sends the requests to partner ω_9. Since partner ω_1 receives all the responses from partner ω_2 and partner ω_3 and all the responses follow Poisson distribution, the effective arrival rate of the responses for partner ω_1 is λ. The requests from partner ω_1 get processed by Web service $ws_{9,2}$ (instantiates ω_9). We compute the response time of Web service $ws_{9,2}$ using *M*/*M*/*m* (here *m* = 2) queuing model due to the replication of Web service $ws_{9,2}$ by 2. Then requests from partner ω_9 are forwarded to partner ω_{10} and gets processed by Web service $ws_{10,1}$ (instantiates ω_{10}). Web service $ws_{10,1}$ is replicated by 3, so the response time of Web service $ws_{10,1}$ is computed using *M*/*M*/*m* (here *m* = 3) queuing model. Finally the requests are forwarded to partner ω_{11} and are processed by Web service $ws_{11,1}$ (instantiates ω_{11}). Thus, the response times of all Web services ($ws_{9,2}$, $ws_{10,1}$, $ws_{11,1}$) are computed and they are 150 *ms*, 333 *ms*, 100 *ms*, respectively.

Note that the possible paths for the requests include ω_1-ω_2-ω_5-ω_9-ω_{10}-ω_{11}, ω_1-ω_2-ω_6-ω_8-ω_9-ω_{10}-ω_{11}, ω_1-ω_3-ω_7-ω_4-ω_9-ω_{10}-ω_{11}. The average response time for path ω_1-ω_2-ω_5-ω_9-ω_{10}-ω_{11} is computed as 100+423+103+150+333+100+10*20 = 1509 *ms*. Similarly, the average response time for path ω_1-ω_2-ω_6-ω_8-ω_9-ω_{10}-ω_{11} is computed as 100+423+435+86+150+333+100+12*20 = 1867*ms* and the average response time for path ω_1-ω_3-ω_7-ω_4-ω_9-ω_{10}-ω_{11} is computed as 100+182+333+103+150+333+100+12*20 = 1541 *ms*. Thus, the average response time of the overall system is 0.3*0.1*1509 + 0.3*0.9*1867 + 0.7*1541 = 1625 *ms*.

CASE STUDY

We use the example system shown in Figure 6 as a case study to illustrate the composition analysis approach. Let *S* denote the example system.

System QoS Specification

In the example system *S*, the QoS attribute vector $Q_S = (q_1, q_2)$, where q_1 represents time and q_2 represents accuracy. The system QoS requirements include the objectives $O = (o_1, o_2)$ and $R = \phi$, where o_1 is "*minimize* (q_1)" and o_2 is "*maximize* (q_2)". Here, q_1 represents response time. For composition specification, we have $\Omega_S = \{\omega_i \mid 1 \le i \le 11\}$. The set of Web services for each partner is shown in Table 1. For example, Web service $ws_{1,1}$ is associated with partner ω_1, Web services $ws_{2,1}$, $ws_{2,2}$ are associated with partner ω_2, Web services $ws_{3,1}$, $ws_{3,2}$, $ws_{3,3}$, $ws_{3,4}$ are associated with partner ω_3, and so forth.

For simplicity, we do not consider the execution environment $E_{i,j}^{ws}$ for Web services $ws_{i,j}$ and the input parameter set $I_{i,j}^{ws}$, for all *i*, *j*, in the example system *S*. The configurable parameter set $X_{i,j}^{ws}$ of each involved Web services $ws_{i,j}$ are specified in Table 2. For example, Web service $ws_{1,1}$ has two configurable parameters $x_{1,1}^{1}$, $x_{1,1}^{2}$, Web service $ws_{2,1}$ has one configurable parameter $x_{2,1}^{1}$, Web service $ws_{3,2}$ doesn't have configurable parameters, and so forth.

The QoS property functions of the Web services are shown in Table 3. Since we have quality attributes q_1 and q_2, correspondingly, we have QoS property functions ${}^1V_{i,j}^{ws}$ and ${}^2V_{i,j}^{ws}$, where ${}^1V_{i,j}^{ws}$ represents the time property function and ${}^2V_{i,j}^{ws}$ represents the accuracy property function. Note that ${}^1V_{i,j}^{ws}$ and ${}^2V_{i,j}^{ws}$ are functions of the

Table 1. Partners and associated Web services

Partner	Web Service	Partner	Web Service	Partner	Web Service
ω_1	$ws_{1,1}$	ω_5	$ws_{5,1}$, $ws_{5,2}$, $ws_{5,3}$	ω_9	$ws_{9,1}$, $ws_{9,2}$
ω_2	$ws_{2,1}$, $ws_{2,2}$	ω_6	$ws_{6,1}$	ω_{10}	$ws_{10,1}$, $ws_{10,2}$
ω_3	$ws_{3,1}$, $ws_{3,2}$, $ws_{3,3}$, $ws_{3,4}$	ω_7	$ws_{7,1}$, $ws_{7,2}$	ω_{11}	$ws_{11,1}$
ω_4	$ws_{4,1}$	ω_8	$ws_{8,1}$		

Table 2. The configurable parameter set of individual Web service specification

Partner	Web Service	Configurable Parameter Set	Partner	Web Service	Configurable Parameter Set
ω_1	$ws_{1,1}$	$X_{1,1}^{ws} = (x_{1,1}^{1}, x_{1,1}^{2})$	ω_6	$ws_{6,1}$	$X_{6,1}^{ws} = \phi$
ω_2	$ws_{2,1}$	$X_{2,1}^{ws} = (x_{2,1}^{1})$	ω_7	$ws_{7,1}$	$X_{7,1}^{ws} = \phi$
	$ws_{2,2}$	$X_{2,2}^{ws} = (x_{2,2}^{1})$		$ws_{7,2}$	$X_{7,2}^{ws} = (x_{7,1}^{1}, x_{7,2}^{2})$
ω_3	$ws_{3,1}$	$X_{3,1}^{ws} = (x_{3,1}^{1})$	ω_8	$ws_{8,1}$	$X_{8,1}^{ws} = \phi$
	$ws_{3,2}$	$X_{3,2}^{ws} = \phi$	ω_9	$ws_{9,1}$	$X_{9,1}^{ws} = \phi$
	$ws_{3,3}$	$X_{3,3}^{ws} = \phi$		$ws_{9,2}$	$X_{9,2}^{ws} = (x_{9,2}^{1})$
	$ws_{3,4}$	$X_{3,4}^{ws} = (x_{3,4}^{1}, x_{3,4}^{2}, x_{3,4}^{3})$	ω_{10}	$ws_{10,1}$	$X_{10,1}^{ws} = (x_{10,1}^{1}, x_{10,1}^{2})$
ω_4	$ws_{4,1}$	$X_{4,1}^{ws} = \phi$		$ws_{10,2}$	$X_{10,2}^{ws} = \phi$
ω_5	$ws_{5,1}$	$X_{5,1}^{ws} = (x_{5,1}^{1})$	ω_{11}	$ws_{11,1}$	$X_{11,1}^{ws} = \phi$
	$ws_{5,2}$	$X_{5,2}^{ws} = \phi$			
	$ws_{5,3}$	$X_{5,3}^{ws} = (x_{5,3}^{1}, x_{5,3}^{2})$			

configurable parameters $X_{i,j}^{ws}$ (it is assumed that the impacts of $I_{i,j}^{ws}$ and $E_{i,j}^{ws}$ are not considered). Since the service time of the Web services follows the exponential distribution, we have

$$ {}^{1}V_{i,j}^{ws}\left(t, X_{i,j}^{ws}\right) = 1 - \frac{1}{\mu_{i,j}\left(X_{i,j}^{ws}\right)} \times e^{-\frac{1}{\mu_{i,j}\left(X_{i,j}^{ws}\right)} \times t} . $$

Note that $\mu_{i,j}(X_{i,j}^{ws})$ denotes the service rate of $ws_{i,j}$,

$$\mu_{i,j}\left(X_{i,j}^{ws}\right)=\frac{1}{t_{i,j}\left(\mu_{i,j}\left(X_{i,j}^{ws}\right)\right)},$$

and here we assume that the configurable parameters in $X_{i,j}^{ws}$ impacts the average service rate. In Table 3, $\mu_{i,j}(X_{i,j}^{ws})$ for each Web service $ws_{i,j}$ is given. Property functions ${}^2V_{i,j}^{ws}$ for all Web services $ws_{i,j}$ are also given in Table 3. It is also a function of the configurable parameters in $X_{i,j}^{ws}$. Here, the value range of the configurable parameters in $X_{i,j}^{ws}$ is 0 to 10.

Consider the Web services $ws_{2,1}$ as an example. The property functions of $ws_{2,1}$ are

$${}^1V_{2,1}^{ws}(t)=1-\left(\frac{1}{\left(2*x_{2,1}^{1}+280\right)}\times e^{-\left(\frac{1}{\left(2*x_{2,1}^{1}+280\right)}\right)\times t}\right)$$

and

$${}^2V_{2,1}^{ws}(t)=0.31*(x_{2,1}^{2})^{0.5}.$$

Table 3. QoS properties of individual Web service specification

Partner	Web Service	$\mu_{i,j}(X_{i,j}^{ws})$	${}^2V_{i,j}^{ws}$
ω_1	$ws_{1,1}$	$0.5*x_{1,1}^{1}+3*x_{1,1}^{2}+10$	$0.63*(x_{1,1}^{2})^{0.2}$
ω_2	$ws_{2,1}$	$2*x_{2,1}^{1}+280$	$0.31*(x_{2,1}^{1})^{0.5}$
	$ws_{2,2}$	$5*x_{2,2}^{1}+250$	$0.63*(x_{2,2}^{2})^{0.2}$
ω_3	$ws_{3,1}$	$3*x_{3,1}^{1}+50$	$0.31*(x_{3,1}^{1})^{0.5}$
	$ws_{3,2}$	130	0.98
	$ws_{3,3}$	200	0.96
	$ws_{3,4}$	$3*x_{3,4}^{1}+x_{3,4}^{2}+2*x_{3,4}^{3}+20$	$0.39*(x_{3,4}^{1})^{0.4}$
ω_4	$ws_{4,1}$	60	0.92
ω_5	$ws_{5,1}$	$3*x_{5,1}^{1}+70$	$0.31*(x_{5,1}^{1})^{0.5}$
	$ws_{5,2}$	90	0.99
	$ws_{5,3}$	$x_{5,3}^{1}+x_{5,3}^{2}+80$	$0.63*(x_{5,3}^{2})^{0.2}$
ω_6	$ws_{6,1}$	200	0.96
ω_7	$ws_{7,1}$	100	0.98
	$ws_{7,2}$	$3*x_{7,2}^{1}+x_{7,2}^{2}+60$	$0.39*(x_{7,2}^{1})^{0.4}$
ω_8	$ws_{8,1}$	70	0.92
ω_9	$ws_{9,1}$	120	0.94
	$ws_{9,2}$	$2*x_{9,2}^{1}+80$	$0.5*(x_{9,2}^{1})^{0.3}$
ω_{10}	$ws_{10,1}$	$2*x_{10,1}^{1}+x_{10,1}^{2}+160$	$0.63*(x_{10,1}^{2})^{0.2}$
	$ws_{10,2}$	170	0.93
ω_{11}	$ws_{11,1}$	50	0.97

The resources in the example system S is $RS = \{rs_1, rs_2, \ldots, rs_{15}\}$, including 15 platforms. We assume that all the platforms have the same computation power.

Configuration and Gene Mapping

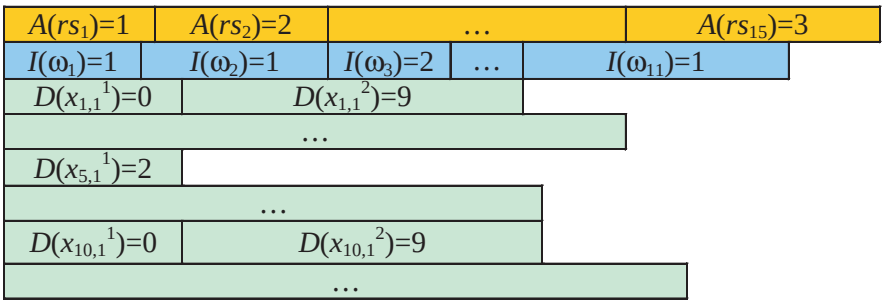
We map the specification for S given in the *System QoS Specification* sub-section to the genetic algorithm paradigm. An example individual for S is given in Figure 7. The genes for selection of Web services are given in the second line. For example, $I(\omega_1) = 1$ represents that $ws_{1,1}$ instantiates ω_1. The genes for resource allocation are given in the first line. We can see that, for example, rs_1 hosts Web service $ws_{1,1}$ since $A(rs_1) = 1$ and $I(\omega_1) = 1$. The remaining genes are for the configurable parameter settings, one line per partner (actually, the Web service that instantiates the partner). For example, $x_{1,1}^{\ 2} = 9$, it means the first configurable parameter $x_{1,1}^{\ 2}$ of the Web service $ws_{1,1}$ is set to 9.

Some other parameters for the genetic algorithm are as follows. The population size N_s and elite set size P are set to 100. The number of generations D is set to 60. These settings are determined based on the convergence speed and output quality we observed.

Performance Analysis

Each system configuration is analyzed by the performance composition algorithm. For the system configuration shown in Figure 7, the service time and accuracy of each Web service, which instantiates the corresponding partner in the example system, are computed according to the configuration mapping and QoS properties of individual Web service specification shown in Table 3. For example, Web service $ws_{1,1}$ is selected to instantiate partner ω_1. The service time and accuracy of $ws_{1,1}$ are 0.5*0 + 3*9 + 10 = 37 *ms* and $0.63*(9)^{0.2} = 0.98$, here $x_{1,1}^{\ 1} = 0$, $x_{1,1}^{\ 2} = 9$. Similarly, Web service $ws_{11,1}$ is selected to instantiate partner ω_{11}. Since Web service $ws_{11,1}$ doesn't have configurable parameters, the service time and accuracy of $ws_{11,1}$ are fixed, namely, 50 *ms* and 0.97, respectively. The average response time of this example system with the given configuration is then computed according to the algorithm "computeTotalAverageResponseTime" presented in the *Performance Analysis for SOA Systems* sub-section and the result is 1412 *ms*. To compute the accuracy of client requests in terms of the overall system, we consider the individual paths of the requests flowing through

Figure 7. Mapping of the system configurations

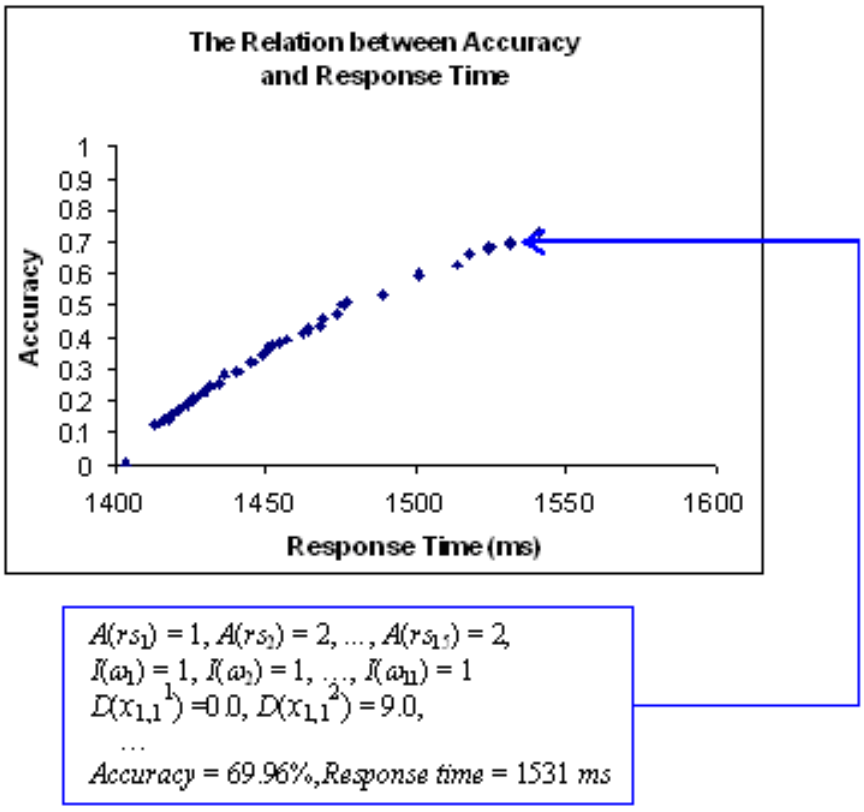


the service hierarchy. For each path, the accuracy of the requests can be computed by multiplying together the accuracy of the multiple tiers that a request has to go through. The accuracy of the overall system is the summation of the accuracy of all the paths in the service architecture. The accuracy of the example system with the given configuration is 0.12.

Analysis Results

For the example system, we consider the tradeoffs of the two QoS objectives—response time and accuracy. Figure 8 shows the Pareto front for the two objectives. There are many possible configurations to meet the functional and QoS requirements of the system *S* and the Pareto front are the configurations that are better than the eliminated solutions in at least one of the objectives. As we can see, in order to obtain higher accuracy, we need longer execution time. Each point in Figure 8 corresponds to a configuration. For example, the configuration corresponding to the point pointed by the arrow in Figure 8 is given in the corresponding box. In this configuration, the configurable parameters for the Web service $ws_{1,1}$ are set to: $x_{1,1}^{1} = 0.0$, $x_{1,2}^{2} = 9.0$, etc. The *average response time* of the system *S* is 1531 millisecond and *accuracy* of the system *S* is 69.96%.

Figure 8. The relationship between accuracy and response time



Based on the analysis results, the system can be adapted statically and/or dynamically according to the modified QoS requirements. When the execution environment or operation condition changes, the system simply searches for the configuration(s) that satisfy the new requirements and constraints. From the case study, we can see that the framework can be used to effectively analyze QoS tradeoffs. The GA algorithm generally converges very fast and, hence, can be used for efficient adaptation decision making.

CONCLUSION

In this chapter, we have presented the QoS analysis process to facilitate dynamic reconfiguration in SOA-based systems. We consider the selections and configurations of Web services to compose a system. We also consider the resource allocation for Web services. The goal is to find the best configurations that satisfy the changing QoS requirements. A case study is used to validate the feasibility of our QoS-based composition analysis techniques and tools.

There are many research directions that can be explored further. The most important direction is to further develop techniques and tools for general QoS behavior analysis. For example, reliability, availability, and so forth, are also important QoS attributes. Currently, we are developing compositional analysis techniques to compute system reliability based on the reliability of individual services. Other QoS attributes will also be considered. Though some quality attributes are very difficult to assess and/or to compose, we plan to investigate these attributes and develop some common mechanisms for thorough analysis.

Dynamic reconfiguration requires the analysis process to be performed in real time so that the adaptation can be completed in a timely way. For the case study discussed in this chapter, the configuration analysis takes close to one second. For large-scale systems, the analysis process may take longer. Thus, it is necessary to investigate mechanisms for achieving real-time performance. For example, special composition analysis service sites can be offered and analysis can be done off-line and results can be provided efficiently to the clients.

We also plan to extend the toolset and develop a complete suite of tools to facilitate QoS analysis of SOA-based systems. We will consider more complicated service architectures and various composition mechanisms. Also, analysis algorithms for various QoS attributes will be incorporated.

REFERENCES

Aksit, M., & Choukair, Z. (2003). Dynamic, adaptive and reconfigurable systems overview and prospective vision. In the *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshop*, Providence, Rhode Island, USA, May 19-22 (pp. 84-89). Los Alamitos, CA: IEEE Computer Society.

BEA Systems, IBM, Microsoft, SAP AG, & Siebel Systems. (2002). *Business process execution language for Web services.* Retrieved May 2005, from http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

Bose, S. K. (2001). *An introduction to queueing systems.* New York: Kluwer/Plenum Publishers.

Corne, D. W., Knowles, J. D., & Oates, M. J. (2000). The pareto envelop-based selection algorithm for multiobjective optimization. In M. Schoenauer et al. (Eds.), *Parallel problem solving from nature – PPSN VI* (pp. 839-848). Berlin: Springer.

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic

algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182-197.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms,* UIUC, July 17-22 (pp 416-423). San Mateo, CA: Morgan Kaufmann.

Hamadi, R., & Benatallah, B. (2003). A petri-net-based model for Web service composition. In K.-D. Schewe, & X. Zhou (Eds.), *Proceedings of the Fourteenth Australasian Database Conference on Database Technologies*, Adelaide, Australia, February 4-7 (pp. 191-200). Australia: Australian Computer Society Inc.

Khare, V., Yao, X., & Deb, K. (2003). Performance scaling of multi-objective evolutionary algorithms. In M. Carlos Fonseca, P. J. Fleming, E. Zitzler, K. Deb, & L. Thiele (Eds.), *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO'03),* Faro, Portugal, April 8-11 (pp. 376-390). Portugal: Springer.

Sirin, E., Parsia, B., & Hendler, J. (2004). Filtering and selecting semantic Web services with interactive composition techniques. *IEEE Intelligent Systems, 19*(4), 42-49.

Srinivas, N., & Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, *2*(3), 221-248.

Tsai, W. T., Song, W., Paul, R., Cao, Z., & Huang, H. (2004). Services-oriented dynamic reconfiguration framework for dependable distributed computing. In the *Proceedings of the 28th Annual International Computer Software and Applications Conference,* Hong Kong, China, September 28-30 (pp. 554-559). Los Alamitos, CA: IEEE Computer Society.

W3C (2002). Web services. Retrieved May 2005, from http://www.w3.org/2002/ws/

W3C (2004a). OWL. Retrieved May 2005, from http://www.w3.org/TR/2004/REC-owl-features-20040210/

W3C (2004b). OWL-S. Retrieved May 2005, from http://www.w3.org/Submission/OWL-S/

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, *8*(2), 173-195.

Zitzler, E., Laumanns, M., & Thiele, L. (2002). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou et al. (Eds.), *Proceedings of the EUROGEN2001 Conference: Evolutionary methods for design, optimization and control with application to industrial problems,* Athens, Greece, September 19-21 (pp. 95-100). Barcelona, Spain: International Center for Numerical Methos in Engineering (CIMNE).

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257-271.

This work was previously published in Advances in Machine Learning Applications in Software Engineering, edited by D. Zhang & J. Tsai, pp. 121-146, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.29
Fuzzy Logic Classifiers and Models in Quantitative Software Engineering

Witold Pedrycz
University of Alberta, Canada

Giancarlo Succi
Free University of Bolzano, Italy

ABSTRACT

The learning abilities and high transparency are the two important and highly desirable features of any model of software quality. The transparency and user-centricity of quantitative models of software engineering are of paramount relevancy as they help us gain a better and more comprehensive insight into the revealed relationships characteristic to software quality and software processes. In this study, we are concerned with logic-driven architectures of logic models based on fuzzy multiplexers (fMUXs). Those constructs exhibit a clear and modular topology whose interpretation gives rise to a collection of straightforward logic expressions. The design of the logic models is based on the genetic optimization and genetic algorithms, in particular. Through the prudent usage of this optimization framework, we address the issues of structural and parametric optimization of the logic models. Experimental studies exploit software data that relates software metrics (measures) to the number of modifications made to software modules.

INTRODUCTION: MODELING SOFTWARE PRODUCTS AND PROCESSES

The important objectives of quantitative software engineering revolve around building models that help express software quality in terms of some software metrics (measures) (Canfora, García, Piattini, Ruiz, & Visaggio, 2005; Cant, Jeffery, & Henderson-Sellers, 1995; Chhabra, Aggarwal, & Singh, 2004; Lanubile & Visaggio, 1997; Lee, 1993; Offutt, Harrold, & Kolte, 1993; Poels &

Dedene, 2000). There have been numerous approaches to such modeling pursuits. In addition to some "standard" linear and non-linear regression models, we can witness other techniques exploiting neural networks, machine learning, neural networks and fuzzy sets (Ebert, 1994, 1996; Mantere & Alander, 2005; Pedrycz, Han, Peters, Ramanna, & Zhai, 2001; Pedrycz & Succi, 2005; Pedrycz, Succi, Musílek, & Bai, 2001; Reformat, Pedrycz, & Pizzi, 2003; Thwin & Quah, 2005).

There are several compelling reasons behind studying different approaches to modeling software processes and software products.

- Typically, the available software data are quite limited yet they may involve a significant number of variables; in this sense, their sparse character requires careful attention.
- As the models produce results to be presented to the user (designers, managers, testers, etc.), it is highly advisable to assure high transparency (user-centricity) of the overall modeling process. In particular, this feature supports an interpretation of results and reveals relationships between software metrics and software quality.
- Software processes are human-oriented and not governed by any laws of physics. This strongly suggests considering modeling practices realized at some abstract levels engaging logic constructs.
- It is highly advisable to develop models in such a way that they could accommodate heterogeneous input information not necessarily being confined to numeric data.
- The models should be endowed with a significant level of flexibility and learning mechanisms to accommodate commonly encountered non-linear relationships and potential variability of the individual projects and software products.

Being aware of these main objectives, we may conclude that ideally the modeling framework should support the development of models in such a way that they combine a high degree of plasticity and learning abilities with an evident transparency and a significant level of interpretability. Interestingly enough, we could state without any exaggeration that the fundamentals of such modeling are inherently rooted in the world of multi-valued or fuzzy logic. The underlying logic nature of the models makes them transparent, and this transparency contributes to a highly interpretative insight into experimental data. The agenda of fuzzy modeling is inherently associated with the transparency of fuzzy models. While this facet of modeling has already started to gain visibility and properly balance the otherwise accuracy-driven fuzzy models, there are still a number of fundamental issues as to the definition of interpretability itself, granularity of models vis-à-vis the characteristics of experimental data, and assessment of the readability of the structure of the model itself (Reformat et al., 2003).

Two-valued logic forms a well-known boundary case of the fuzzy logic. The design of digital systems comes with a diversity of well-established, highly efficient, and scalable architectures and related development algorithms. By acknowledging a point of view that the two-valued logic is just a special case of fuzzy logic, we are then somewhat tempted to generalize or reformulate the already existing architectures and design practices of digital systems and cast them in the framework of fuzzy logic. This point of view is the crux of this approach to the development of fuzzy logic models. We focus on a certain standard technique of implementation of combinational systems by means of multiplexers (the approach which results in an array of multiplexers implementing any Boolean function); see McCluskey (1986) and generalize this concept to the world of fuzzy logic. In essence, we are concerned with the three main phases: (1) building a generic

structure of a fuzzy multiplexers; (2) developing models (networks) exploiting fuzzy multiplexers as their building components (we will be referring to them as networks of fuzzy multiplexers); and (3) designing such networks with the aid of methods of structural and parametric optimization.

As emphasized, the embedding principle (where fuzzy logic subsumes two-values logic and inherits from its fundamental constructs), makes that this starting point of view becomes especially justifiable and appealing considering that multiplexers have been commonly used in the design of digital systems and come with a well-developed design methodology (Ciletti, 1999).

The organization of the material is structured in a way that reflects the research agenda outlined earlier. First, we introduce a basic processing module of a fuzzy multiplexer (fMUX) and discuss its characteristics (the *Fuzzy Multiplexer as a Generic Processing Unit* section). This naturally leads us to the networks formed by fuzzy multiplexers, *A Realization of the Network of fMUXs* section, relates their structure to the expansion theorem by Shannon, and emphasizes the nature of the function decomposition completed in this manner. *The General Development Environment of the Network-Interfaces Issues* section is concerned with the general development of fMUX networks and serves as a pre-requisite to the comprehensive discussion on the design of the networks. *The Development of the fMUX Networks* section concentrates on the genetic optimization of the networks using which we address an issue of their structural optimization (concerned with a selection of an optimal subset of input variables) and parametric learning. The discussion covers architectural considerations of GAs as well as presents the underlying genetic operators pertinent to the optimization realized here. Experimental results dealing with synthetic and software data are included in the *Experimental Studies* section.

The terminology used here adheres to the standards encountered in two-valued logic, digital systems, and fuzzy logic. The logic operators are modeled via t- and t-conorms. If not stated otherwise, we use two standard realizations of t- and t-conorms coming in the form of a product and probabilistic sum. An overbar denotes a complement treated in a usual way encountered in logic (that is $\overline{x} = 1\text{-}x$).

FUZZY MULTIPLEXER AS A GENERIC PROCESSING UNIT

We are concerned with the development of logic-based models of data in the unit hypercube. More precisely, such models realize logic transformations that map the unit hypercubes (say $[0,1]^n$) into [0,1] and come with some well-defined semantics. We require that such mapping is made modular meaning that it is formed on a basis of a collection of some generic processing units (nodes). The basic processing node, referred to as a fuzzy multiplexer (fMUX), realizes a mapping from $[0,1]^2$ into [0,1] that comes in the following form

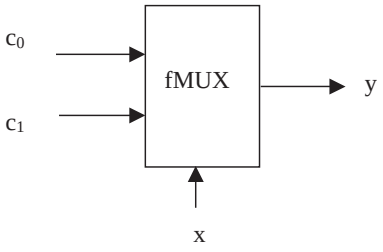
$$y = c_0 * \overline{x} + c_1 * x \qquad (1)$$

where the logic operations (* and +) are implemented using some t- and t-conorms. For the sake of completeness of the presentation, let us recall that by a t-norm we mean a two argument function t: $[0,1]^2$ –>[01,] such that it is monotonic, associative, distributive, and satisfies two boundary conditions of the form 0tx =0 and 1tx =x for all x in [0,1]. For any t-conorm all these conditions hold true while the boundary condition reads as 0sx =x and 1sx =1. Given these t- and t-conorm realizations, (1) is governed by the expression

$$y = (c_0 t\overline{x})\, s\, (c_1 t x) \qquad (2)$$

The structure of (1) can be schematically illustrated as visualized in Figure 1. The variable (x) standing in the earlier-mentioned expression plays a role of a *switching* (selection or select)

Figure 1. A schematic view of the fuzzy multiplexer with one select input (x) and two fixed information inputs (c_0 and c_1)



variable that allows two fixed *information* inputs (c_0 or c_1) to affect the output. The degree to which the produced result depends on these fixed information values is controlled by the select variable. To emphasize the role played by all these signals, we use a concise notation y = fMUX(x, **c**) where **c** (=[c_0 c_1]) denotes a vector of the information inputs. In the two boundary conditions the select variable may assume, we produce a binary switch (as being used in digital systems). It means that if x =1 then y = c_1. Likewise the value of x set up to 0 leads to the output being equal to c_0 meaning that the value of c_0 is transferred to the output of the device.

Figure 2 includes a series of plots of the characteristics of the fuzzy multiplexer being treated as a function of x; noticeable is a fact that different configurations of the values of the information inputs (c_0 and c_1) give rise to different non-linear input-output relationships of the device. By choosing a certain value of the select input, we logically "blend" the two constant logic values present at the information inputs. The detailed non-linear form of the relationship depends on the use of the t-norms and t-conorms. In the specific example illustrated in Figure 2, the t-norm is realized as a product operator that is atb = ab while the t-conorm is treated as the probabilistic sum, asb = a + b – ab, a,b ∈ [0,1].

Using fuzzy multiplexers, we can easily form cascade structures (networks) as commonly

Figure 2. Input-output characteristics of the fuzzy multiplexer for selected values of c_0 and c_1, y = fMUX(x,c_1,c_0):

a - c_0=0.9 c_1=0.8
b- c_0=0.2, c_1=0.8
c – c_0=1.0, c_1=0.0
d- c_0=0.6, c_1=0.3

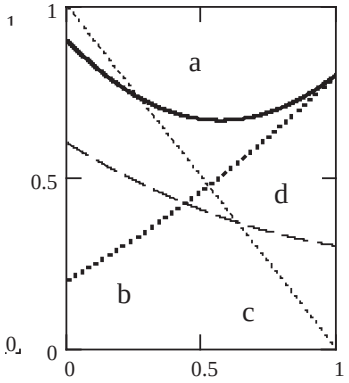
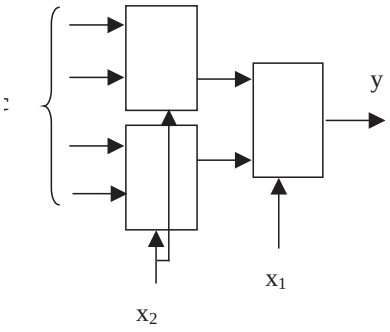
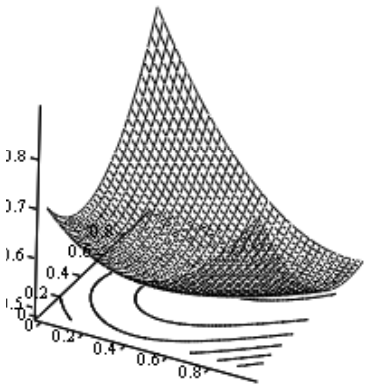


Figure 3. A two-layer network of fuzzy multiplexers

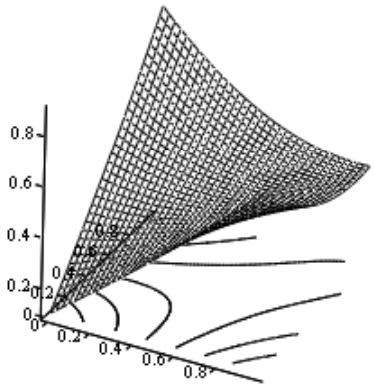


encountered in the two-valued logic constructs (digital systems). An example of a two-layer structure, a network of fMUXs is displayed in Figure 3. The input-output characteristics of the network that are regarded as a function of the select signals x_1 and x_2 are shown in Figure 4. Depending upon the numeric values in the vector of the information inputs (**c**), we encounter various types of non-linearities. Figure 5 includes the corresponding characteristics for the minimum and maximum operations; it becomes apparent that the piece-wise character of the relationships becomes predominant. Regarding the notation,

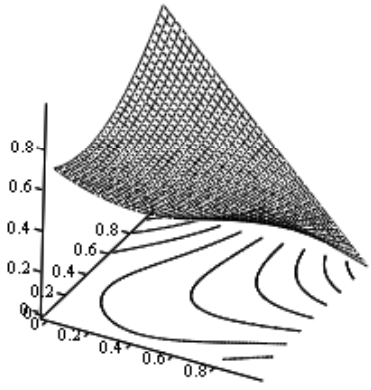
Figure 4. 3D plots of the input-output characteristics of the fMUX for selected combinations of the parameters and t- and t-conorms: ***c*** *= [0.7 0.8 0.9 0.5] (a);* ***c*** *= [0.1 0.8 0.9 0.5] (b);* ***c*** *= [0.7 0.8 1 0] (c);* ***c****=[1 1 1 1] (d)*



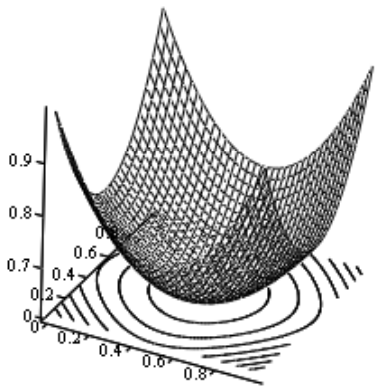
(a)



(b)

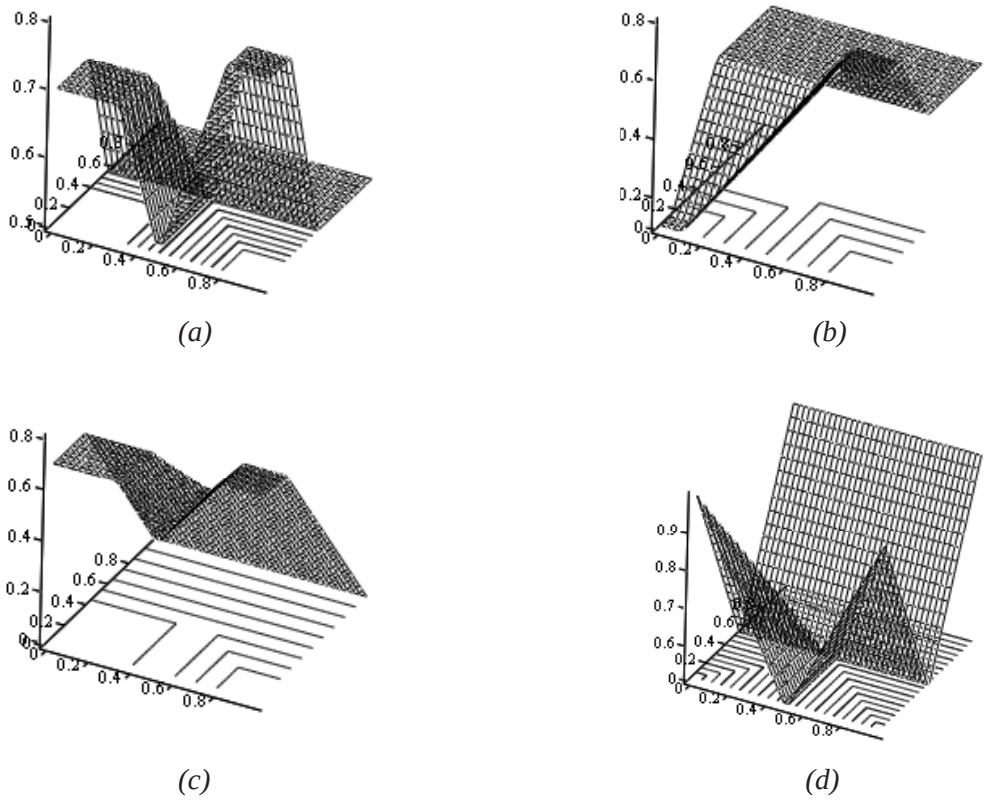


(c)



(d)

Figure 5. 3D plots of the characteristics of the fMUX for selected combinations of the parameters; here t- and t-conorms are realized as a minimum and maximum: $\mathbf{c}$ *= [0.7 0.8 0.9 0.5] (a);* $\mathbf{c}$ *= [0.1 0.8 0.9 0.5] (b);* $\mathbf{c}$ *= [0.7 0.8 1 0] (c);* $\mathbf{c}$*=[1 1 1 1] (d)*



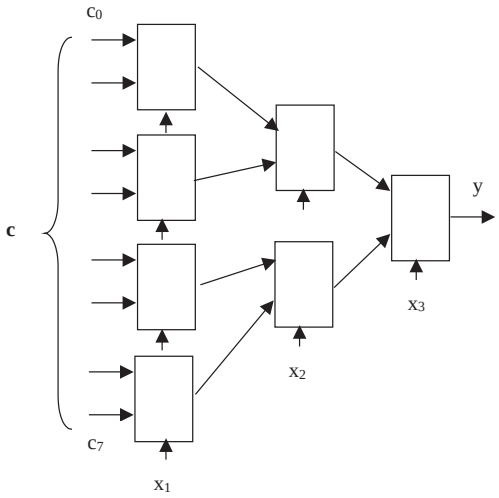
(a) *(b)* *(c)* *(d)*

Figure 3, the switching variables as listed from left to right visualize their location across the network with the last variables being the one realizing the final switching.

A REALIZATION OF THE NETWORK OF FMUXS

The functional module of the fuzzy multiplexer introduced earlier is a generic building block that can be efficiently used to construct larger structures. Its functionality is minimal (in the sense we have here only a single switching variable, that is x). If we are dealing with more variables (x_1, x_2, …,x_n), the resulting structure is formed as a regular multi-level architecture composed of the basic fMUXs as visualized in Figure 6. Noticeably at each level of the network, we assign a single selection variable. Moreover at each level of the network, the number of multiplexers doubles; the output layer comprises one multiplexer, the

Figure 6. Network architecture built with the use of the basic functional modules of fuzzy multiplexers; for illustrative purposes, shown are only three layers of the network



layer next to it two multiplexers, the next one four units, and so forth. With the substantial number of select variables, we can envision some scalability problems (and these have to be addressed at the design phase).

The fMUX network comes with an interesting motivation and exhibits a clear interpretation. As functionality is concerned, it is instructive to go back to the two-valued logic which clearly reveals a rationale behind the use of such networks of multiplexers (Kohavi, 1970). Consider a two-variable Boolean function $f(x_1, x_2)$. According to the classic Shannon expansion theorem, the function can be written down as a sum of products, that is

$$y = f(x_1, x_2) = \overline{x}_2 f(x_1,0) + x_2 f(x_1,1)$$
$$= \overline{x}_2[\overline{x}_1 f(0,0) + x_1 f(1,0)] + x_2[\overline{x}_1 f(0,1) + x_1 f(1,1)] \quad (3)$$

In essence **c** = [f(0,0) f(1,0) f(0,1) f(1,1)] becomes a vector of constant information inputs here; these numeric values uniquely define the given Boolean function.

This successive expansion of the Boolean function maps directly on the two-level structure of the multiplexers; the information inputs to the multiplexer are the functions of one variable, namely $f(x_1,0)$ and $f(x_1,1)$, which are realized by the two multiplexers in the first layer of the network. Here, f(0,0), f(1,0), and so forth, are the information inputs to these multiplexers. The network of the fuzzy multiplexers is just a generalization of this fundamental result to fuzzy functions defined in the unit interval.

When it comes to the interpretation, the network exhibits several interesting properties. We start with the input layer. The outputs of these fMUXs are just logic expressions of a single

input (select) variable (being more specific, the variable and its complement). In the sense of involving only one variable, they are general. There are also a lot of them (especially if we are dealing with the multi-player network). In this sense, what becomes realized at the first layer is just a list of partial realizations of the function, and the outputs there can be treated as generalized variables. In the subsequent layers, these are specialized (by involving another variable), and their number becomes reduced.

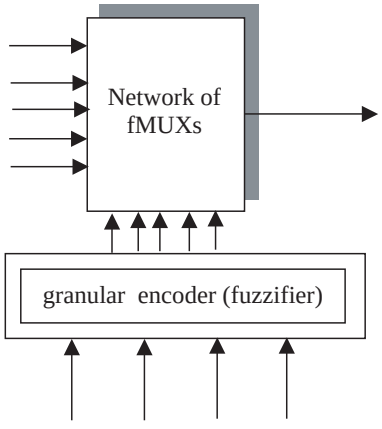
THE GENERAL DEVELOPMENT ENVIRONMENT OF THE NETWORK-INTERFACES ISSUES

The fuzzy multiplexer completes a logic-based processing of input variables and realizes a certain logic-driven mapping between input and output spaces. As they interact with a physical world whose manifestation does not arise at the level of logic (multi-valued) signals, it becomes apparent that there is a need for some interface of the model. Such interfaces are well known in fuzzy modeling. They commonly arise under a name of fuzzifiers (granular coders) and defuzzifiers (granular decoders). The role of the coder is to convert a numeric input coming from the external environment into the internal format of logic membership grades associated with each input variable. This can be shown in the form outlined in Figure 7.

The choice of the fuzzy sets coming as the components of the interfaces is essential to the performance of the fuzzy models. In particular, this concerns: (1) the number of fuzzy sets and (2) their membership functions. The number of fuzzy sets implies a certain level of granularity of the model while the form of the membership function could help in further parametric optimization of the model.

In this study, we consider one of the simplest scenarios. While being quite straightforward, this design alternative exhibits interesting properties, supports computational simplicity and implies a significant level of the interpretability of the model.

Figure 7. A general-layered structure of fuzzy modeling; the granular decoder is used in case of several networks of fuzzy multiplexers



For an input variable that assumes values in the range of [a, b], let us consider two fuzzy sets with triangular membership functions defined as

$$A_1(x) = (x-a)/(b-a) \quad (4)$$

and

$$A_2(x) = 1- (x-a)/(b-a) \quad (5)$$

Evidently, the family $\{A_1, A_2\}$ forms a fuzzy partition of the space as $A_1(x) + A_2(x) = 1$. These two fuzzy sets overlap at the level of 0.5. The linear model of membership is quite common and simple to interpret. It is also highly legitimate in all those cases where we do not have any additional knowledge about the problem at hand. We can look at these membership functions from a different standpoint. The linear normalization of the input variable from the interval of [a, b] to the unit interval [0,1] is just expressed by (4) so in essence A_1 is a result of such normalization. A_2 complements A_1. A_1 is monotonically (linearly) increasing while A_2 is monotonically (linearly) decreasing.

THE DEVELOPMENT OF THE FMUX NETWORKS

In this section, we discuss some general design scenarios and envision their suitability in the development of the fMUX networks.

Selecting Among General Design Scenarios

The development of the fMUX networks entails two fundamental design scenarios:

1. If we consider all input variables (x_1, x_2, …, x_n) to be used in the development of the system then the values of the entries of the vector $\mathbf{c}=[c_0\ c_1\ c_2\ c_k\ldots]$ have to be estimated
2. If the number of the input variables is high (and this implies a high dimensionality of **c** along with all drawbacks of learning we envision under such circumstances), the design of the network has to involve a selection of an optimal subset of the variables and a simultaneous estimation of the pertinent vector of constants (**c**).

In the first scenario, we can use a standard gradient-based learning. It is straightforward; for a given structure (that is the variables being specified in advance along with their arrangement within the network), a detailed form of a performance index to be minimized (Q), specific models of t- and t-conorms, the gradient of Q taken with respect to **c** navigates the optimization process realized throughout the search space,

$$\mathbf{c}(\text{iter}+1) = \mathbf{c}(\text{iter}) - ?\nabla_{\mathbf{c}} Q \quad (6)$$

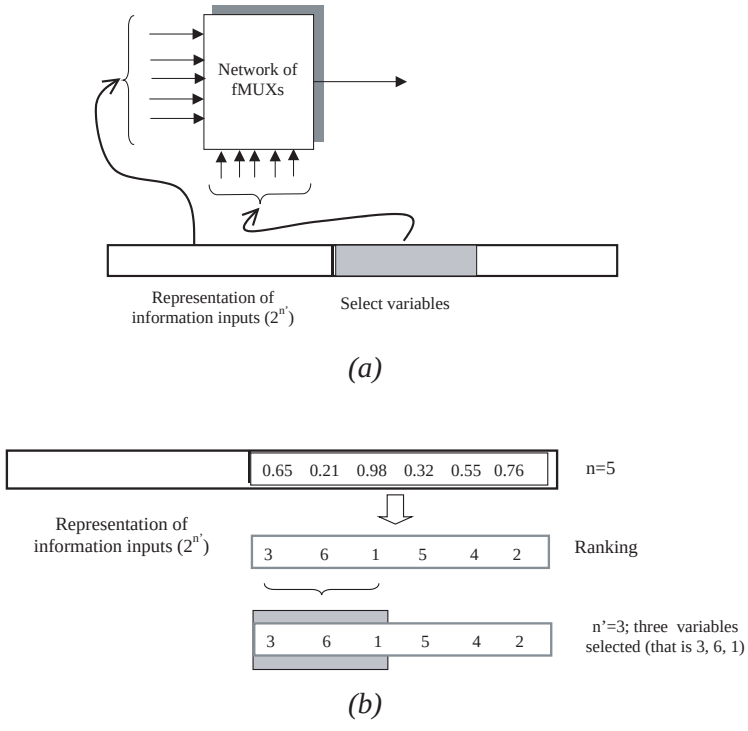
where **c**(iter) denotes the values of the input constants at a certain iteration step (iter); $\beta > 0$ is a learning factor implying an intensity of adjustments of the values of **c**. The gradient of Q can be easily determined. In spite of that, there could be some potential shortcomings of this learning scheme. The most profound one comes with a high dimensionality of the network. If there are a significant number of the variables in the problem, the computed gradient assumes low values. As a result, the learning becomes very inefficient. Note also that the dimensionality of the input vector is equal to 2^n and this expression gives rise to a prohibitively high dimensionality quite quickly even for relatively small values of "n". In light of these, it is very likely that the gradient-based methods will come with a limited applicability and we have to proceed with caution when dealing with the increased problem dimensionality.

The second design scenario involves an optimization of the structure (selection of variables) that helps handle the dimensionality problem in an efficient manner. The parametric optimization concerning the vector of the co-efficients in some reduced format becomes then more efficient. We may also envision frequent situations in which not all variables become essential to the design of the logic mapping (the same holds in pattern recognition where a stage of feature selection becomes a necessity). With the structural and parametric optimization at hand, we have to confine ourselves to some techniques of global and structural optimization. An appealing way to follow is to consider genetic algorithms (GAs).

Genetic Development of the fMUX Networks

Having recognized the primary design objectives, we now concentrate on the details of the underlying genetic optimization. GAs (Goldberg, 1989; Michalewicz, 1996) are well-documented in the literature including a long list of their numerous applications to neuro-fuzzy systems. Bearing this in mind, we elaborate on the fundamental architecture of the GA, its parameters, and discuss some implementation details.

Figure 8. The structure of the fMUX network and its genetic representation (a) and details of the coding of the subset of the input variables (b) through ranking and using the first n' entries of the sub-string



Genotype Representation

The proposed genotype is a direct reflection of the structure of the fMUX network. We consider a floating point coding as this results in compact chromosomes. Let us assume that the number of input variables to be used in the network is given in advance and equal to n' where n' < n. The chromosome consists of two sub-strings of real numbers. The first block (sub-string) contains $2^{n'}$ values of the information inputs, vector. The second block (with n inputs) deals with the subset of the variables to be used in the design. The details are schematically visualized in Figure 8.

As far as the structure of network is concerned, it is instructive to discuss a way in which the select variables are coded in the second block of the chromosome. The second portion of the chromosome corresponds to the subset of the original inputs that are chosen as select variables and requires some processing before being used to identify the structure of the network. As the entries of the chromosome are real-coded, the likelihood of encountering two identical entries is zero (to be on the safe side, we can always break a tie randomly). With the pre-defined number of the inputs (n'), we then use only the first n' entries of the chromosome, and this produces the sequence of the input variables. The entries are ranked (in the increasing order), and the first n' entries of the sub-string are used to choose among all variables. This ordering is directly mapped onto the network where the first variable is the one switching the first layer of the network.

N.B. One could easily optimize the number of the subset of the input variables (n') instead of supplying it externally yet this does not to seem to be very attractive. It is perhaps more justifiable to do a systematic search by sweeping n' from 1 to n. In essence, this systematic search helps us assess approximation and generalization abilities of the networks and get a better sense as to the plausible subset of the variables.

When it comes to the *selection process*, we use an elitist ranking selection (Michalewicz, 1996). This selection mechanism means that individuals to be selected for the next generation are based on their relative rank in the population, as determined by the fitness function. The best individual from each generation is always carried over to the next generation (elitist mechanism) meaning that the solution found so far during the genetic optimization is guaranteed to never disappear.

The *mutation* operator is a standard construct encountered in a number of genetic algorithms, cf. . 0. Given an individual string $\mathbf{a} = [a_1, a_2, \ldots, a_{2n}]$, we generate a new string $\mathbf{a}' = [a_1', a_2', \ldots, a_{2n}']$ where a_i', i=1, 2,…,2n, is a random number confined in the range of [0,1] and subject to the following replacement (mutation) rule: a_i is mutated that is replaced by a_i' with some probability of mutation (p_m) otherwise the entry of the chromosome is left intact, that is $a_i' = a_i$.

The *crossover* operation is realized as the BLX-0.5 crossover operator (Baker, 1985; Eshelman & Schaffer, 1993; Goldberg, 1991; Herrera, Lozano, & Verdegay, 1998) which is carried out as follows. Given are two individuals $\mathbf{a} = [a_1, a_2, \ldots, a_{2n}]$ and $\mathbf{b} = [b_1, b_2, \ldots, b_{2n}]$. The resulting offsprings are formed in the form $\mathbf{a}' = [a_1', a_2', \ldots, a_{2n}']$ and $\mathbf{b}' = [b_1', b_2', \ldots, b_{2n}']$, where a_i', b_i', i=1,2,…,2n, are random numbers located in the range $[\max(0, \min_i - 0.5I), \min(1, \max_i + 0.5I)]$. Here, $\min_i = \min(a_i, b_i)$, $\max_i = \max(a_i, b_i)$, and $I = \max_i - \min_i$. This particular crossover operation provides a good balance between using the information of the parents and avoiding premature convergence. The crossover operator ensures that all values of the generated offspring are confined to the unit interval [0, 1]. The operator is employed with probability of crossover, p_c, otherwise the individuals are left unchanged $\mathbf{a}' = \mathbf{a}$ and $\mathbf{b}' = \mathbf{b}$.

Fitness Function

The fitness function quantifies how the network approximates the data and is taken as

$$1 - \frac{Q}{Q + \varepsilon}$$

with Q being a sum of squared errors between the target values (experimental output data) and the corresponding outputs of the network. The other option is the standard root mean squared error (RMSE). A small positive constant ε standing in the denominator of the earlier expression assures that the fitness function remains meaningful even for Q = 0 (which in practice never occurs).

EXPERIMENTAL STUDIES

In the experimental part of the study, we start with some illustrative material dealing with Boolean expressions and then move on to the software data.

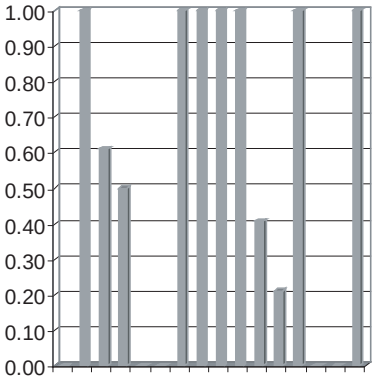
Realization of Boolean Functions

In this experiment, we are concerned with Boolean data; this helps us compare the result produced

Table 1. Structure of the multiplexer network and associated errors; by the classification error, we mean the number of mismatches between the binary (that is subjected to some threshold) output of the network and the data

Number of variables	1	2	3	4	5
Order of variables	x_3	x_5, x_3	x_3, x_5, x_2	x_4, x_3, x_2, x_5	x_4, x_1, x_2,x_5,x_3
MSE	0.478	0.408	0.354	$1.29*10^{-4}$	$2.09*10^{-5}$
Classification error	3	2	4	0	0

Figure 9. Values of the information inputs; observe that their distribution is highly bimodal with the values of information inputs being close to 1 or 0 with a very few exceptions



by the fuzzy multiplexer with the solutions obtained using standard techniques used to design digital systems. The data set comprises of 12 input-output pairs of binary data with 5 inputs and a single output,

(**x**(k)-y(k)): ([1 0 0 0 0] 0), ([1 0 0 0 1] 1), ([1 0 0 1 0] 1), ([1 0 0 1 1] 1), ([1 1 1 0 0] 1),
([1 1 1 0 1] 0), ([1 1 1 1 0] 1), ([1 1 1 1 1] 1), ([1 1 0 0 0] 0), ([1 1 0 0 1] 1),
([1 1 0 1 0] 0), ([1 1 0 1 1] 0)

The development of the network is completed for a varying number of inputs starting from one variable and ending up with all the variables. The results are summarized in Table 1.

From this table (based on the values of the classification error), it becomes obvious that four variables are optimal for the problem. The resulting inputs are shown in Figure 9. With the threshold value of 0.5, we are left with eight significant inputs. Noticeably, all those very close to 1 identify the min terms existing in the data. The one extra here with the value equal to 0.61 (and still deemed relevant) corresponds to the combination of the inputs equal to 0100 (that is $\overline{x}_4 x_3 \overline{x}_2 \overline{x}_5$), and it is subsumed by the remaining sum of the minterms. Alluding to the problem variables being identified by the genetic algorithm, it is interesting to note that x1 has been eliminated. It is not surprising at all noting that it fixed at 1 and thus becomes redundant in the problem. For this optimal number of the inputs, Figure 10 shows how GA performs in terms of the optimization; evidently most learning occurs at the beginning of the process, and this becomes evident for the best individual as well as the average fitness in the population.

The second example is shown here to visualize the effectiveness of the genetic optimization. The binary data describe a three-dimensional XOR problem. With the population of 50 chromosomes, 50 generations, the mutation rate of 0.05, and crossover rate equal to 0.8, the ideal result is obtained after a few initial generations. As expected, the information inputs are either close to 1 or become practically equal to zero (Figure 11).

Experiments with Software Data

In the ensuing comprehensive suite of experiments, we consider a well-documented MIS data

Figure 10. Fitness function (average and best individual) in successive generations

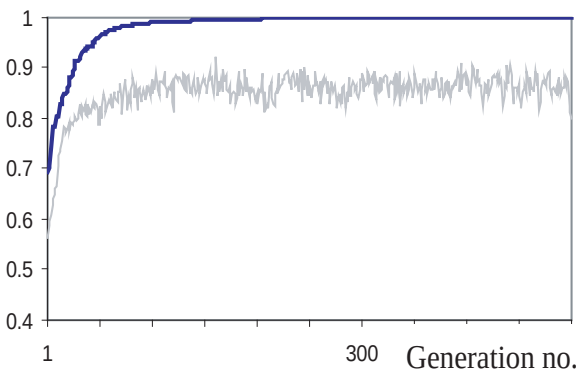
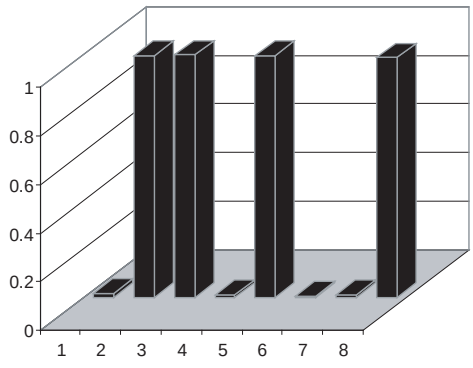


Figure 11. Information inputs of the network of fuzzy multiplexers



set (Munson & Khoshgoftaar, 1996) which has already been used in other experimentation with models of quantitative software engineering. The dataset includes software modules described by a collection of typical software metrics and associated number of changes made to the corresponding module. For the purpose of this experimentation, we focused only on the selected software measures leaving some others out (such as, e.g., number of characters, number of code characters, number of comments); refer to Table 2.

For further experimentation, we eliminated one data point for which the number of changes was overly high (98) in comparison to the rest of the data and which could easily skew up the entire picture. Note that the average number of changes was 7.25. In the learning process, we use 60% of randomly selected data. The rest (40%) is left for testing purposes. The t- and t-conorm are implemented as the product and probabilistic sum.

As discussed in *The General Development Environment of the Network-Interfaces Issues* section, the software measures were linearly normalized to the unit interval. The evolutionary optimization was realized with the following parameters of the GA whose values are quite typical and quite similar to those found in the literature, namely

size of population –300,
number of generations – 500,
mutation rate – 0.05,
crossover rate – 0.8

As a matter of fact, it is worth noting that those parameters could be fine-tuned, however the process might be quite tedious and geared only towards a specific dataset.

In the series of experiments, we modified the number of allowed input variables starting from a single software metrics and moving on to the entire set. The results are reported in terms of the performance index (RMSE) for the training and testing set as well as the corresponding subsets of the variables (see Table 3).

We observe that the performance index on the training set achieves the minimum for two and three variables. With the increase of the dimensionality, there is some increase in the values of the performance index. Given this, we may regard the two software metrics, that is, the number of lines of code, and the estimated program length

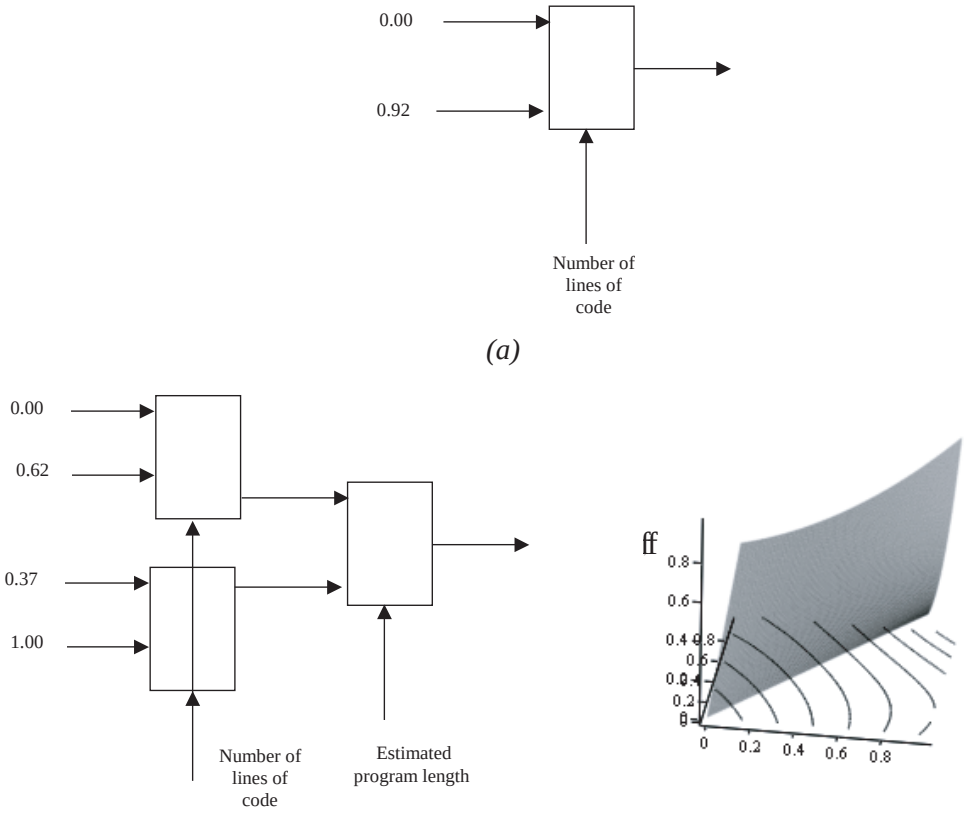
Table 2. A collection of software metrics (measures) used in the experiments

	Software metrics	notes
x_1	Number of lines of code, including comments	
x_2	Program length	N = N1 + N2; N1- the total number of operators, N2-total number of operands
x_3	Estimated program length	$n_1 \log_2 n_1 + n_2 \log_2 n_2$ n_1-the number of unique operators, n_2 – the number of unique operators
x_4	Jensen's estimator of program length	$(\log_2 n_1)! + (\log_2 n_2)!$
x_5	McCabe cyclomatic complexity number	number of decision nodes +1
x_6	Belady's bandwidth metric	average level of nesting

Table 3. Results of genetic optimization of the logic model of the network of fuzzy multiplexers (the switching variables are listed starting from the first layer of the fuzzy multiplexers in the overall architecture)

Number of input variables	1	2	3	4	5	6
Software metrics and their arrangement	x_1	x_1 x_3	x_1 x_5 x_3	x_1 x_4 x_3 x_5	x_3 x_1 x_4 x_6 x_5	x_4 x_1 x_3 x_5 x_6 x_2
Performance index (training data)	5.42	**5.37**	5.37	5.41	5.53	5.76

Figure 12. The detailed architecture of the network with the values of the inputs to the fuzzy multiplexers; shown are also the input-output characteristics of the networks for the two software metrics



to form a suitable subset of software metrics. By adding the third metrics, that is, McCabe cyclomatic complexity, we are not witnessing any improvement (the performance index stays the same). As a matter of act, the performance index on the testing set reaches 8.09 for two metrics and 5.37 for the suite of the three metrics. The topologies of the networks with a single input and two inputs along with their characteristics are visualized in Figure 12.

The findings are intuitively appealing as far as the specific software metrics are concerned and their occurrence in the overall expression. In the case of the single input, the number of lines of code is a fairly compelling indicator of possible changes. The logic expression identifies a direct relationship between the increase in the values of this metrics and the resulting increase in the number of changes made to the software module. This observation is drawn from the logic expression; see Figure 12, $y = 0.92\ t\ x_1 = 0.92\ x_1$ (as the t-norm has been implemented as the product operator). This relationship is linear as illustrated in Figure 12. Given the underlying logic of the fuzzy multiplexer, one could infer that the increase of the number of lines of code leads to the increase of the number of changes in the code. With the two inputs, we observe that the x1 is the first switching variable and the results produced at the first layer are combined by the fuzzy multiplexer guided by x_3 (estimated program length). The logic expression governing the relationships is intuitively appealing: roughly speaking, the increase in the number of changes occurs when both the number of lines and the estimated program length are increasing or at least one of them with the more significant impact caused by the high number of lines. This slightly asymmetric behavior is also reflected in Figure 12.

CONCLUSIONS

We have introduced a logic-driven architecture of fuzzy models based on a concept of fuzzy multiplexers (fMUXs). fMUXs are direct generalizations of fundamental building blocks encountered in two-valued (digital) logic and being used in a design process therein. The design of the fMUX networks has been carried in the framework of genetic optimization. The GA is effectively used at the level of structural and parametric optimization. It is worth stressing that the structural optimization becomes indispensable in case of multi-variable problems. The selected (optimized) subset of input variables leads to an efficient dimensionality reduction of the problem and helps concentrate on the most significant variables. With this regard, the resulting transparency of the model is also worth emphasizing. Given the topology of the network itself and the processing therein, its structure becomes highly interpretable. Any fMUXs construct is translated into a coherent logical description of the experimental data being used in its development. Taking full advantage of this aspect, we were able to reveal and quantify immediate links between some software metrics and the quality of the software product (in our case, it related to the number of changes that were made to the software modules). We emphasize that the network of fMUXs translates into a logic description of experimental software data we used in their development. Given the form of the input interface that linked the data with the logic processing realized through two linear membership functions, we came up with an interesting and highly readable semantics of the relationships conveyed by the logic model. In essence, the logic model endowed with this type of interface expresses "gradual" dependencies between the inputs (software metrics) and some ensuing effects (the number of changes) and quantifies the strength of these relationships

(which is done with the use of the numeric values of the data inputs of the fMUXs positioned at the input layer of the network).

REFERENCES

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms* (pp. 101-111). Mahwah, NJ: L. Erlbaum Associates.

Canfora, G., García, F., Piattini, M., Ruiz, F., & Visaggio, C. A. (2005). A family of experiments to validate metrics for software process models. *Journal of Systems and Software, 77*(2), 113-129.

Cant, S. N., Jeffery, D. R., & Henderson-Sellers, B. (1995). A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology, 37*(7), 351-362.

Chhabra, J. K. , Aggarwal, K. K., & Singh, Y. (2004). Measurement of object-oriented software spatial complexity. *Information and Software Technology, 46*(10), 689-699.

Ciletti, M. D. (1999). *Modeling, synthesis and rapid prototyping with the Verilog HDL*. Upper Saddle River, NJ: Prentice Hall.

Ebert, C. (1994). Rule-based fuzzy classification for software quality control. *Fuzzy Sets and Systems, 63*(3), 349-358.

Ebert, C. (1996). Fuzzy classification for software criticality analysis. *Expert Systems with Applications, 11*(3), 323-342.

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval schemata. In D. Whitley (Ed.), *Foundations of genetic algorithms 2* (pp. 187-202). San Mateo, CA: Morgan Kaufmann Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Boston, MA: Addison-Wesley.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems, 5*, 139-167.

Herrera, F., Lozano, M., & Verdegay J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis. *Artificial Intelligence Review, 12*, 265-319.

Kohavi, Z. (1970). *Switching and finite automata theory.* New York: McGraw-Hill.

Lanubile, F., & Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: Lessons learned. *Journal of Systems and Software, 38*, 225-234.

Lee, H. (1993). A structured methodology for software development effort prediction using the analytic hierarchy process. *Journal of Systems and Software, 21*(2), 179-186.

Mantere, T., & Alander, J. T. (2005). Evolutionary software engineering: A review. *Applied Soft Computing, 5*(3), 315-331.

McCluskey, E. J. (1986). *Logic design principles.* Englewood Cliffs, NJ: Prentice Hall.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs* (3rd ed.). Heidelberg: Springer-Verlag.

Munson, J., & Khoshgoftaar, T. M. (1996). Software metrics for reliability assessment. In M. R. Lyu (Ed.), *Handbook of software reliability engineering* (pp. 493-530). New York: McGraw Hill.

Offutt, A. J., Harrold, M. J., & Kolte, P. (1993). A software metric system for module coupling. *Journal of Systems and Software*, *20*(3), 295-308.

Pedrycz, W., Succi, G., Musílek, P., & Bai, X. (2001). Using self-organizing maps to analyze object-oriented software measures. *Journal of Systems and Software*, *59*(1), 65-82.

Pedrycz, W., Han, L. , Peters, J. F., Ramanna, S., & Zhai, R. (2001). Calibration of software quality: Fuzzy neural and rough neural computing approaches. *Neurocomputing*, *36*(1-4), 149-170.

Pedrycz, W., & Succi, G. (2005). Genetic granular classifiers in modeling software quality. *Journal of Systems and Software*, *76*(3), 277-285.

Poels, G., & Dedene, G. (2000). Distance-based software measurement: Necessary and sufficient properties for software measures. *Information and Software Technology*, *42*(1), 35-46.

Reformat, M., Pedrycz, W., & Pizzi, N. J. (2003). Software quality analysis with the use of computational intelligence. *Information and Software Technology*, *45*(7), 405-417.

Thwin, M. M. T., & Quah, T.-S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software*, *76*(2), 147-156.

This work was previously published in Advances in Machine Learning Applications in Software Engineering, edited by D. Zhang & J. Tsai, pp. 148-167, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 7.30
A Formal Verification and Validation Approach for Real-Time Databases

Pedro Fernandes Ribeiro Neto
Universidade do Estado do Rio Grande do Norte, Brazil

Maria Lígia Barbosa Perkusich
Universidade Católica de Pernambuco, Brazil

Hyggo Oliveira de Almeida
Federal University of Campina Grande, Brazil

Angelo Perkusich
Federal University of Campina Grande, Brazil

ABSTRACT

Real-time database-management systems provide efficient support for applications with data and transactions that have temporal constraints, such as industrial automation, aviation, and sensor networks, among others. Many issues in real-time databases have brought interest to research in this area, such as: concurrence control mechanisms, scheduling policy, and quality of services management. However, considering the complexity of these applications, it is of fundamental importance to conceive formal verification and validation techniques for real-time database systems. This chapter presents a formal verification and validation method for real-time databases. Such a method can be applied to database systems developed for computer integrated manufacturing, stock exchange, network-management, and command-and-control applications and multimedia systems. In this chapter, we describe a case study that considers sensor networks.

INTRODUCTION

Nowadays, the heterogeneity of platforms, distributed execution, real-time constraints, and other features are increasingly making software development a more complex activity. Besides, the amount of data to be managed is increasing as well. Taken together, complexity and data management are causing both risk and cost of software projects to get higher.

Database management systems are used to manage and store large amounts of data efficiently. However, when both data and transactions have timing restrictions, real-time databases (RTDB) are required to deal with real-time constraints (Ribeiro-Neto, Perkusich, & Perkusich, 2004). For an RTDB, the goal is to complete transactions on time, while maintaining logical and temporal consistency of the data. For real-time systems, correct system functionality depends on logical as well as on temporal correctness. Static analysis alone is not sufficient to verify the temporal behavior of real-time systems. To satisfy logical and temporal consistency, concurrency control techniques and time-cognizant transactions processing can be used, respectively. The last occurs by tailoring transaction management techniques to explicitly deal with time.

The real-time ability defines nonfunctional requirements of the system that must be considered during the software development. The quality assurance of real-time systems is necessary to assure that the real-time ability has been correctly specified. Imprecise computation is used as a technique for real-time systems where precise outputs are traded off for timely responses to system events. For that, formal models can be created to verify the requirement specifications, including the real-time specifications (Ribeiro-Neto, Perkusich, & Perkusich, 2003).

Validation as well as verification can be carried out by simulation model. With the simulation model, a random sample will be selected from the input domain of the test object, which is then simulated with these chosen input values. After that, the results obtained by this execution are compared with the expected values. Thus, a simulation model is as a dynamic technique, that is a technique that contains the execution of the test object. One major objective of simulation models is error detection (Herrmann, 2001).

The main motivation for this research is the fact that methods to describe conceptual models of conventional database systems cannot be directly applied to describe models of real-time database systems. It occurs because these models do not provide mechanisms to represent temporal restrictions that are inherent to real-time systems. Also, most of the available models focus on the representation of static properties of the data. On the other hand, complex systems, such as real-time databases, also require the modeling of dynamic properties for data and information. Therefore, the development of methods to design real-time databases with support for both static and dynamic modeling is an important issue.

In the literature, there are few works for real-time database modeling that allow a formal analysis, considering verification and validation characteristics. The existing tools for supporting modeling process especially do not present simulation capacity. The unified modeling language (UML) approach presents a number of favorable characteristics for modeling complex real-time systems, as described in Selic and Rumbaugh (1998) and Douglass (2004). UML also is used for modeling object-oriented database systems. However, the existing tools for UML modeling do not present simulation capacity.

This chapter describes a formal approach to verify and validate real-time database systems. The approach consists of the application of the five steps: (1) building an object model; (2) building a process model; (3) generating an occurrence graph; (4) generating a message-sequence chart; and (5) generating a timing diagram. The two first steps include static and dynamic analysis, respectively. The following steps allow the user

to validate the model. Hierarchical coloured Petri nets (HCPNs) are used as the formal language to describe RTDB models (Jensen, 1998). The proposed approach can be applied to different domains, such as computer-integrated manufacturing, stock exchanges, network management, command-and-control applications, multimedia systems, sensor networks, and navigation systems. In this chapter, we describe a case study considering sensor networks. Sensor networks are used to control and to monitor the physical environment and sensor nodes may have different physical sensors and can be used for different application scenarios.

The remainder of this chapter is presented as follows. First, a background is presented, to ease the comprehension of approach. Concepts about RTDB, quality of services and HCPNs are defined. Second, the formal verification and validation approach for real-time databases is described as well as a sensor network case study. Third, future trends are presented. Finally, conclusions are presented.

BACKGROUND

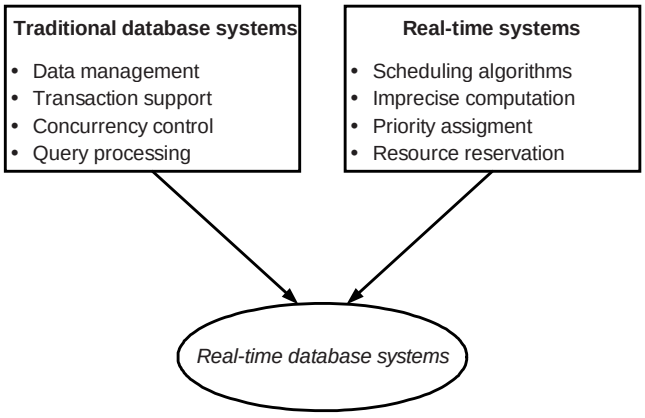
Real-Time Databases (RTDB)

The real-time database-management systems must provide the characteristics of conventional databases besides assuring that the real-time constraints are imposed on both the data and transactions. These constraints arise in applications where the transactions must meet deadlines.

The amount of applications that benefit from the utilization of RTDB is increasing as well. This increase is a consequence of the proliferation of embedded systems that includes both systems that are similar to those present in personal computers and smaller systems with a minimal memory and calculator capacity, such as those present in mobile devices.

An RTDB is required when: The volume of data is large; responses depend on multiple values; responses to aperiodic events are required; and there are constrained timing requirements. The correctness in real-time databases implies: sat-

Figure 1. Real-time database systems



isfying all usual consistency constraints; executing transactions within timing constraints; and satisfying temporal consistency of the data. The real-time data and transactions are also defined. The data items reflect the state of the environment. The transactions are classified with respect to their deadlines, such as hard, soft, or firm; arrival-pattern — periodic, aperiodic, sporadic; and data-access-pattern — read-only, write-only and update. In Figure 1, a schema illustrating the properties of the RTDB is shown.

Data Properties

The data correctness in RTDB is assured by logical and temporal consistency. The real-time data can be classified into static and dynamic. The correctness of static data is guaranteed by the logical consistency, since is has not become outdated. The dynamic data may change continuously to reflect the real-world state, such as object positions, physic measure, stock market, and so on. Each dynamic datum has a timestamp of the latest update and the data can be divided into base data and derived data. A derived datum can be derived from various base data (Kang, 2001).

The external consistency of dynamic data is defined using validity intervals to assure the consistency between the state represented by the database content and the actual state of environment. The validity intervals are of two types as follows (Kang, 2001):

- **Absolute validity interval (*avi*)** is defined between the environment state and the value reflected in the database. The data *x* is considered temporally inconsistent if (*now* - *timestamp*(*x*) > *avi*(*x*)), where *now* is the actual time of system, *timestamp* is the time of the latest update of data.
- **Relative validity interval (*rvi*)** is defined among the data used to derive other data. Consider a data item *y* is derived from a data set $R=\{x_1,x_2,...,x_k\}$. *y* is temporally consistent if the if the data in *R* that the compose are temporally valid and the $|timestamp(x_i \in R) - timestamp(x_j\ 0\ R)| \leq rvi(y)$. This measure arises to produce derived data from data with the approximate time.

The dynamic data are represented by *x*:(*value*,*avi*,*timestamp*) and will be temporally consistent. If both absolute and relative validity interval are satisfied. Consider the example where a data item *t*, with *avi*(*t*)=5, reflect the current temperature and the data item p represent the pressure with *avi*(*p*)=10. The data item y is derived from data set *R*={*t*,*p*} and have relative validity interval *rvi*(*y*)=2. If the actual time is 50, then (a) *t*:(25,5,45) and *p*:(40,10,47) are temporally consistent because as absolute validity interval as relative validity interval is valid. But, (b) *t*:(*25,5,45*) and *p*:(*40,10,42*) are not temporally consistent, because only the absolute validity interval is assured.

Transaction Properties

The real-time transactions are characterized along three dimensions based on the nature of transactions in real-time database systems: the nature of real-time constraints, the arrival pattern, and the data-access type.

- **Real-time constraints:** The real-time constraints of transactions are related to the effect of missing its deadline and can be categorized in hard, firm and soft. Hard deadlines are those that may result in a catastrophe if the deadline is missed. These are typically critical systems, such as a command delayed to stop a train causing a collision.
  To complete a transaction with a soft deadline after its time constraint is undesirable. However, soft deadlines missed can commit

the system performance. The transactions with firm deadline will be aborted if its temporal constraints are lost.

- **Arrival pattern of transactions:** The arrival pattern of transactions refers to time interval of execution. Generally, the transactions are periodically executed in real-time databases, since they are used to record the device reading associated to the environment or to manipulate system events. The arrival pattern can be aperiodic, where there is not a regular time interval between the executions of transactions. The transactions also can execute in random time. However, there is a minimal time interval between the executions of transactions.
- **Data access type:** In relation to data access, the transactions are categorized as: write transactions (or sensors), update transactions, and read transactions. The *write transactions* obtain the state of the environment and write into the database. The *update transactions* derive new data and store them in the database. Finally, the read transactions read data from database and send them.

In the database, it is necessary to guarantee the same views, of the same data item, for different transactions. This property is called *internal consistency* and is assured by the ACID properties. ACID is an acronym for atomicity, consistency, isolation, and durability. These properties are defined for a real-time database as follows:

- **Atomicity:** Is applied for subtransactions, where a subtransaction must be whole executed or neither step must be considered of them.
- **Consistency:** The transaction execution must always change the consistent state of a database in another consistent state. An imprecision limited in the internal consistency can be permitted in order to meet the temporal constraints of transactions.
- **Isolation:** The actions of a transaction can be visible by other transactions before it commits.
- **Durability:** The actions of a transaction need not be persistent, since both data and transactions have temporal validity.

Concurrency Control

The negotiation between logical and temporal consistency, a concurrency-control technique should be capable of using knowledge about the application to determine which transactions can be executed concurrently. Such a technique, named semantic concurrency control, allows increasing the concurrent execution of transactions (method invocation). Based on the knowledge of the application the designer must define which transactions may be concurrently executed and when. Defining compatibilities between the executions of the transactions does this. Therefore, this technique allows relaxing the ACID properties.

Transactions in real-time do not need to be serialized, especially updated transactions that record information from the environment. However, the consequence of relaxing serialization is that some imprecision can be accumulated in the database, and in the vision of the database.

An object-oriented semantic concurrency control technique, described in DiPippo (1995), named *semantic-lock technique*, allows logical and temporal consistency of the data and transactions and allows the negotiation among them. The technique also allows the control of the imprecision resulting from the negotiation. The concurrency control is distributed among the objects, and a compatibility function, says C*F* for short, is defined for each pair of methods for database objects. *CF* is defined as follows:

$$CF(m_{ati}, m_{inv}) = Boolean\ Expression \rightarrow IA$$

where m_{ati} represents the method that is being executed and, m_{inv} represents the method that was invoked. The *Boolean Expression* can be defined based on predicates involving values of the arguments of the methods, the database attributes, and the system in general. *IA* is defined by an expression that evaluates the accumulated imprecision for the attributes of the database object and for the arguments of the methods.

The consequence of using such a concurrency control is that more flexible scheduling for transactions can be determined than those allowed by serialization. Besides, that technique can specify and limit some imprecision that may appear in the system due to relax of the serialization.

Quality of Service (QoS) Management

In a real-time database, the QoS management can help to verify both the correctness and performance of a system, through functions and performance metrics. This is necessary, since the real-time transactions have temporal constraints. Therefore, we consider transactions correct only if they finish within their deadlines using valid data.

The functions defined are the functions of *specification, mapping, negotiation,* and *monitoring.* The function specification defines which QoS parameters are available and determines their syntax and semantics. The mapping function has to be provided to translate the QoS requirements expressed.

The role of a QoS negotiation mechanism is to determine an agreement for the required values of the QoS parameters between the system and the users or applications. A QoS negotiation protocol is executed, every time a new user or application joins an active session, to verify whether the system has enough resources to accept the new user or application request without compromising the current performance. This function usually employs several QoS mechanisms to fulfill its task, such as: *admission control* is used to determine whether a new user can be served, while *resource reservation* has to be called as soon as the user is admitted, in order to guarantee the requested service quality. The negotiation function has the role of the compability function, described above.

We define two performance metrics to guarantee the RTDB performance. These metrics are shown as follows:

1. **Number of transactions that miss the deadline in relation to the amount of transactions that finish with success (*Pt*):** This metric set up the rate of missed deadline of transactions that can be allowed during a time interval. The metric is defined as:

$$Pt = 100 * \left(\frac{MissedDeadline}{FinishTransactions} \right)$$

   where *Pt* is the amount of transactions that miss the deadline (*MissedDeadline*) in relation to the amount of transactions that finish with success (*FinishTransactions*).
2. **Upper imprecision of data (*Impr*):** Is the threshold of imprecision admitted in the data item for it to be considered logically valid. *Impr* is defined as:

$$Impr = CurrentValue * \left(\frac{Imp}{100} \right)$$

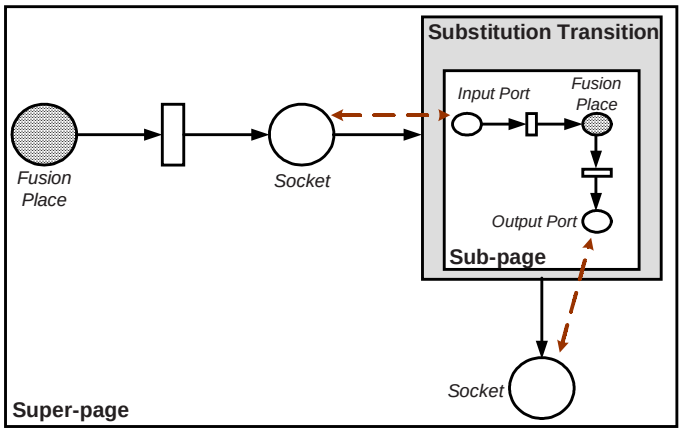
   where *CurrentValue* is the value of data item stored in database and *Imp* is the index of amount of imprecision admitted.

HCPN-Based Modeling

Hierarchical Coloured Petri Nets

Hierarchical coloured Petri nets (HCPNs) are an extension of coloured Petri nets (CPNs) (Jensen, 1998) and are a suitable modeling language for verifying systems, as they can express concurrency, parallelism, nondeterminism, and different levels of abstraction.

Figure 2. Coloured Petri net



In Figure 2, a Petri net is illustrated, where hierarchical levels are allowed. These hierarchical levels are possible due to the inclusion of two mechanisms: substitution transitions and fusion places. A substitution transition is a transition that will be replaced by a CPN page. The page to which the substitution transition belongs is called a superpage and the page represented by the transition is called the subpage. The association between subpages and superpages is performed by means of sockets and ports.

Sockets are all the input and output places of the transition in the superpage. Ports are the places in the subpage associated to the sockets. The ports can be input, output, or input-output. For simulation and state, space-generation sockets and ports are glued together and the resulting model is a flat CPN model. The fusion places are physically different but logically only one forming a fusion set. Therefore, all the places belonging to a fusion set have always the same marking. A marking of a place is the set of tokens in that place in a given moment. The marking of a net is the set of markings of all places in the net at a given moment (Jensen, 1998).

Indeed, these two additional mechanisms, substitution transitions and fusion places, are only graphical, helping in the organization and visualization of a CPN model. They favor the modeling of larger and more complex systems by giving the designer the ability to model by abstraction, specialization, or both.

Design/CPN Tools

Design/CPN (Jensen et al.,1999) is a tool package supporting the use of HCPN. The Design/CPN tool has four integrated parts:

1. The **CPN editor** supports construction, modification, and syntax check of CPN models.
2. The **CPN simulator** supports interactive and automatic simulation of CPN models.
3. The **occurrence graph tool** supports construction and analysis of occurrence graphs

for CPN models (also known as state spaces or reachability graphs/trees).

4. The **perfomance tool** supports simulation=based performance analysis of CPN models.

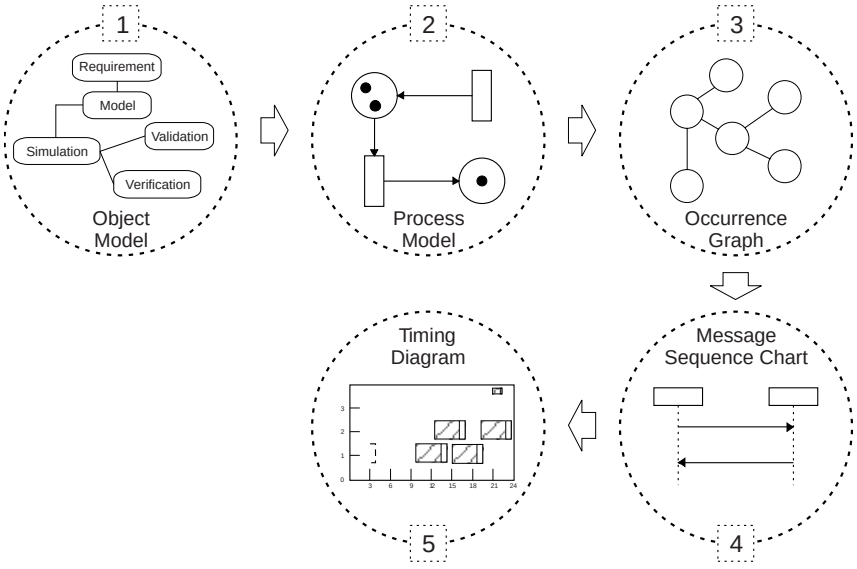
The design/CPN package is one of the most used Petri net tools. Design/CPN supports CPN models with complex data types (colour sets) and complex data manipulations (arc expressions and guards), both specified in the functional programming language Standard ML(Jensen et al., 1999).

REAL-TIME DATABASE VERIFICATION AND VALIDATION METHOD

The formal verification and validation method for real-time database systems consists of the application of the following steps, as illustrated in Figure 3, which are detailed in this section:

1. **Build an object model:** It is used to specify the requirements and identify the main components of the system. It is also used to model static properties of objects, such as attributes, operations, and logical and timing constraints. In any way, the object model defines the discourse universe to the process model.
2. **Build a process model:** It is used to model both functional and dynamic properties of objects. The functional properties define the object operations, while the dynamic property represents the temporal interactions of objects and its answers to the events. The process model is composed of the operations identified in the *object model.*
3. **Generate an occurrence graph:** It is a representation of the state space of the HCPN model.
4. **Generate a message sequence chart:** They are generated for each scenario, considering a possible execution sequence.

Figure 3. Real-time database verification and validation method



5. **Generate a timing diagram:** It is a diagram to show the timing constraints in time sample.

Build an Object Model

In the object model each *object* is a unique entity. Objects with the same data structure (attributes) and behavior (operations), in the context of the particular application environment are grouped into an object *class.* Classes can be grouped in a hierarchical structure. Classes may have attributes; the attributes are structural properties of classes that can have both logical and temporal constraints; the *relationships* are the links between the classes; the *operations* are functions or procedures applicable to the class attributes, and the *method* is the implementation of an *operation* (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991).

The object model consists of a set of: *class diagram*, *object diagram*, and *data dictionary.* The *class diagrams* have shown the general description of the system, while the *object diagrams* shown object instances. The data *dictionary* defines whole entities modeled (class, associations, attributes, operations).

The object model begins with the problem declaration analysis and has the following steps:

1. **Identification of the objects:** The external actors and objects that interact with the system are identified as the problem context. Elements of the object model that emerge from the analysis of the real problem are directly mapped into logical objects. Each instance of an object is assumed to be unique. The objects in an object class have a unique identity that separates and identifies them from all other object instances.
2. **Identification of relationships among objects:** A conceptual relationship among instances of classes. Associations have *cardinality* including one-to-one, one-to-many, and many-to-many. Most object-oriented texts do not address the *nature* of an association (i.e., mandatory or optional), except in the definition of the object behavior.
3. **Addition of attributes to objects:** a data value that can be held by the objects in a class. Attributes may be assigned to different data types (e.g., integer).
4. **Use of generalizations to observe similarities and differences:** the essential characteristics of an object or class, ignoring irrelevant features, providing crisply defined conceptual boundaries. This maintains a focus upon identifying common characteristics among what may initially appear to be different objects. Abstraction enhances reusability and inheritance.
5. **Identification of operations:** the direct manipulation of an object, categorized as: *Constructor:* create an object and/or initialize. *Destructor:* free the state of an object and/or destroy the object. *Modifier:* alter the state of the object. *Selector:* access and read the state of an object. *Iterator:* access all parts of an object in a well-defined order.
6. **Identification of concurrent operations:** In this step, the designer analyzes the system to discover which operations need to be executed concurrently and in that condition this occurs. In follow, it is defined the function that details the situations which the operations can be executed concurrently.
7. **Identification of both logical and temporal constraints:** The designer must declare both logical and temporal constraints to objects. These constraints define the correct states of each object. Thus, the constraints are defined as predicates that include the attributes value, time, and so on. For instance, the absolute validity interval defined to real-time data, in the Background section, expresses a temporal constraint to data objects.

Build a Process Model

The process model captures both functional and dynamic properties of objects. This model is used in the analysis, design, and implementation phases of the software-development life cycle. These phases can be tackled concurrently, using hierarchical coloured Petri nets. HCPNs are used to analyze the system behavior. In this model, the objects are described through HCPN modules (or pages) that are defined from object models. Then, for each object that contains operations identified in the model, a HCPN module is created, where the correspondent operations are modeled. We use the design/CPN tool package (Jensen et al., 1999) for HCPN modeling. For that, the following steps must be performed:

1. **Identification of the objects in HCPN:** In this step, all of the objects in the object model are identified, and for each object identified an HCPN module is constructed.
2. **Identification of functions for each object:** The operations that must be executed by each object are identified. What each object must execute is analyzed without considering its implementation.
3. **Definition of the interface for each object:** The interface of each object is declared, indicating the methods with its respective argument of input and output, the constraints defined to the classes, besides functions that describe the compatibility between methods.
4. **Definition of the internal structure of each object:** The methods detailed in the interface of objects are described, satisfying the requisites identified in the phase of identification of the objects.

Occurrence Graph (OG)

The occurrence graph tool is closely integrated with the design/CPN tool package (Jensen et al., 1999). The basic idea behind occurrence graphs is to make a directed graph with a node for each reachable marking and an arc for each occurring binding element. OGs are directed graphs that have a node for each reachable marking and an arc for each binding element. An arc binding the marking node that the binding element associated occurs at each marking node resultant of occurrence (Jensen, 1998).

The OG has a large number of built-in standard queries, such as *Reachable*, which determines whether there is an occurrence sequence between two specified markings, and *AllReachable*, which determines whether all the reachable markings are reachable from each other. These queries can be used to investigate all the standard properties of a HCPN. In addition to the standard queries, there are a number of powerful search facilities allowing formulating nonstandard queries. The standard queries require no programming at all. The nonstandard queries usually require that 2-5 programming lines of quite straightforward ML code.

Through an occurrence graph, it is possible to verify the properties inherent to the model. The occurrence graph tool allows obtaining reports with general properties about the model. These reports contain information about the graph and metaproperties that are utilities for comprehension of model behavior in HCPN. For instance: *boundness properties*, which supply the upper and lower limit of tokens that each net place can contain, besides marking limits for each place; *liveness properties*, which shown the markings and transitions that are dead (not precede none other marking) and which transitions are live (appear in some occurrence sequence started of the initial marking of the net). Occurrence graphs can be constructed with or without considering time or code segments.

When an occurrence graph has been constructed using the design/CPN it can be analyzed in different ways. The easiest approach is to use the *Save Report* command to generate a standard

report providing information about all standard CPN properties:

- **Statistics:** Size of occurrence graph
- **Boundedness properties:** Integer and multiset bounds for place instances
- **Home properties:** Home markings
- **Liveness properties:** Dead markings, dead/live transition instances
- **Fairness properties:** Impartial/fair/just transition instances

To use the OG tool, the user simply enters the simulator and invokes the Enter Occ Graph command (in the file menu of design/CPN). This has a similar effect as Enter Simulator. It creates the occurrence graph code, that is, the ML code necessary to calculate, analyze, and draw occurrence graphs. Moreover, it creates a new menu, called Occ. This menu contains all the commands which are used to perform the calculation and drawing of occurrence graphs.

Generate a Message Sequence Chart (MSC)

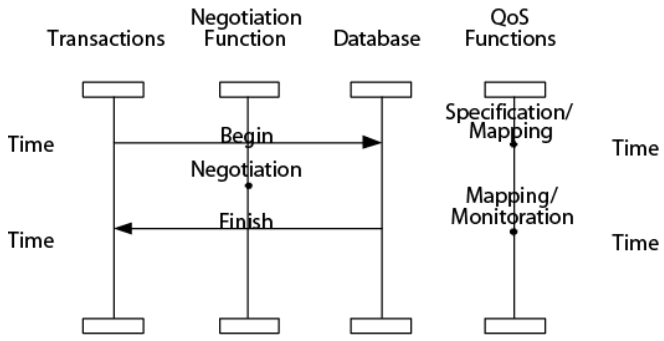
MSC is a graphical and textual language for the description and specification of the interactions between system components. Message sequence charts may be used for requirement specification, simulation and validation, test-case specification and documentation of real-time systems.

As illustrated in Figure 4, the MSC comprises the QoS functions, the transactions with its operations, and the RTDB. In this method, the use of MSC is primordial, since it is possible to verify the properties of real-time database by representing the transactions properties and data properties, both with temporal constraints. Also, it is possible to validate the behavior of objects, its relationships, and the situations where concurrent access to the RTDB occurs through the object operations. To generate the MSC, we use the "*smc.sml*" library of the design/CPN tool package.

Generate a Timing Diagram (TD)

The design/CPN performance tool for facilitating simulation-based performance analysis of HCPN generates the timing diagram. In this context, performance analysis is based on the analysis of data extracted from a HCPN model during simulation. The Performance tool provides random number generators for a variety of probability distributions and high-level support for both data collection and for generating simulation output.

Figure 4. Description of message sequence chart



The random number generators can be used to create more accurate models by modeling certain probability distribution aspects of a system, while the data collection facilities can extract relevant data from a CPN model.

Before data can be collected from a HCPN model, it is necessary to generate the *performance code*, that is, the ML code that is used to extract data from the HCPN model. The design/CPN performance tool can then be used to generate performance reports as a time diagram.

Case Study: Real-Time Database for Sensor Networks

Case Study Overview

A sensor network is considered as application domain to the case study, where the method proposed is applied. For this case study, a scenario where the environment monitored must have a steady temperature is described. The upper and lower bound for temperature is defined. Sensors are placed in the environment with the objective of acquiring and storing the temperature values.

Periodically, data stored in the sensors are sent to a real-time database server, through sensors transactions. The data obtained has temporal validity and the transactions have a deadline. The server is updated in order to allow historical queries. The architecture of the case study is illustrated in Figure 5.

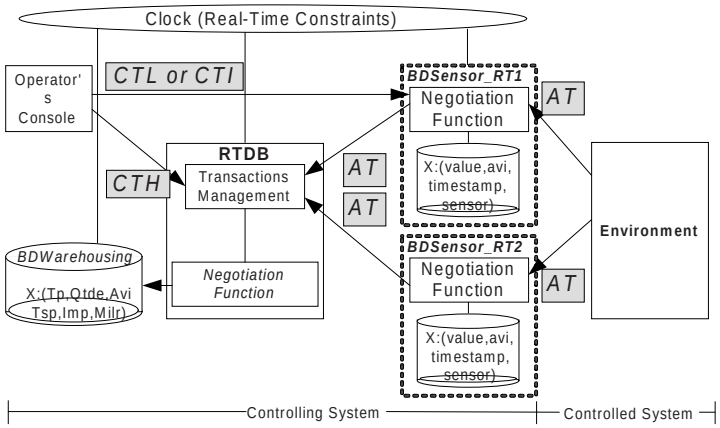
Applying the Proposed Method

Building the Object Model

According to the steps defined to obtain the object model, we have:

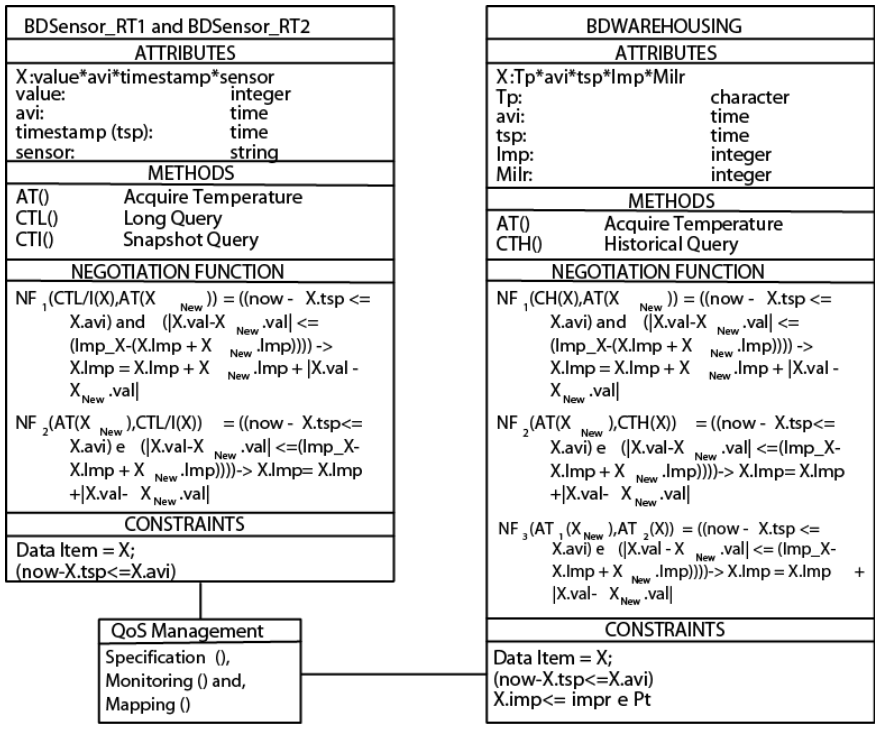
1. **Identification of the objects:** The objects identified in the model are the sensors *BDSensor_RT1 and BDSensor_RT2*, and the real-time database server, called *BDWarehousing.*
2. **Identification of relationships among objects:** The sensors send data to the server through transactions. Each sensor updates the server, while the server is updated by various sensors.

Figure 5. Architecture of the sensor network case study



3. **Addition of attributes to objects:** The data item *X* acquired by the sensor is composed of the following attributes: *Value* is the content of data item; *avi* is the absolute validate interval; *timestamp* is the late update time; and *sensor* identifies which sensor acquired the data. The attributes of the data item stored in the real-time database server has the fields: *Tp*, which is the data item processed; *Qtde*, which is the value that will be updated in the server; *Avi*, which is the absolute validate interval; *Tsp*, which is the late update time; *sensor*, which identifies the sensor that acquired the data; *Imp*, which is the accumulated imprecision; and *Milr*, which is the limit of *Imp*.
4. **Use of generalization to observe similarities and differences:** This step is unnecessary for this model, due to existence of only two objects.
5. **Identification of operations:** The sensors aim at acquiring data of the external environment (method *AT*) and these data can be read by long and snapshot queries (method *CTL* and method *CTI*, respectively). Long queries are performed in a time interval, and snapshot queries are performed in an absolute time. The real-time database server has historical data obtained by sensors (method *AT*), and allows one to query this historical data (method *CTH*).
6. **Identification of concurrent operations:** The *BDSensor_RT1 and BDSensor_RT2* object has two negotiation functions that represent two different types of concurrency. The first situation is observed when

Figure 6. Object model



the data item is being acquired and a query is invocated. The second situation of concurrency is possible when a query is running and an acquisition operation begins. In the *BDWarehousing*, three negotiation functions define the concurrence between the transactions. Besides the situations defined to the sensors, it is possible that two update operations try to access the same data item, where the sensor is updating the item and an applicative program is changing this data.

7. **Identification of both logical and temporal constraints:** In the sensor, the constraints defined to the data are: The type and the absolute validity interval of them. The constraints defined to the server are: the type of data item and the performance metrics *Pt* and *Impr*, described in this chapter.

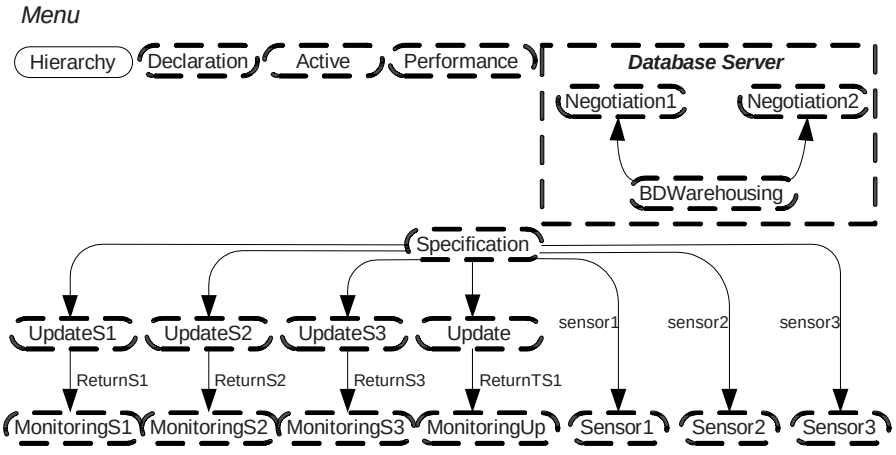
The QoS management is performed by the functions: specification, mapping, and monitoring, in addition to the negotiation function defined for the objects. Figure 6 illustrates the object model for the case study.

Building the Process Model

According to the steps defined to obtain the process model, we have:

1. **Identification of the objects in HCPN:** In this first step, the objects are identified from the object model for the the HCPN modules.
2. **Identification of functions for each object:** The sensor object implements the mechanisms of acquisition and stored data, besides reading the content stored. The real-time database server object implements the update and reading of the database.
3. **Definition of interface for each object:** In the HCPN module of sensor, the interface is defined by the methods: *AT*, *CTL*, and *CTI* and by attribute *X* that represents a record. The interface of the HCPN module to the server object indicates the methods *AT* and *CTH* and the attribute *DB* that represent a record with the fields defined to the data item stored in the server.

Figure 7. Process model



4. **Definition of internal structure for each object:** The internal structure is a hierarchical coloured Petri net to model the methods declared in the interface of the object.

The overview of the process model is illustrated in Figure 7. The HCPN modules are:

- **Declaration:** This represents the declarations, that is, the functions, types, and so on.
- **BDWarehousing:** It is the database server.
- **Negotiation1 and Negotiation:** This represents the negotiation functions.
- **Specification:** It is the module where the temporal parameters are specified.
- **Sensor1, Sensor2, and Sensor3:** This represents the modules for sensors.
- **UpdateS1, UpdateS2, and UpdateS3:** These are the sensors' transactions that update the server.
- **MonitoringS1, MonitoringS2, and MonitoringS3:** They are the monitoring functions related to each sensor transaction.
- **Update and MonitoringUp:** These modules are for the update transaction (only read) and the monitoring function defined for it.
- **Active and Performance:** These are control modules.

Generating the Occurrence Graph

For the real-time database modeled, the full standard report follows. According to the report, we have the full generation in 47 seconds, with 6,713 nodes and 22,867 arcs (see Box 1).

Some places are shown in the *boundedness properties*. The place ObjetoBD'ObjetoBDP represents the repository of data and it has the limit of 1 token. Two different combinations to the token in this place are represented in report (see Box 2).

Box 1.

```
Statistics
------------------------------------------------------------------------
Occurrence Graph
  Nodes:  6,713
  Arcs:   22,867
  Secs:   47
  Status: Full
```

Box 2.

```
Boundedness Properties
------------------------------------------------------------------------
Best Integers                    Bounds  Upper   Lower
ObjetoBD'ObjetoBDP                1       1       1

Best Upper Multi-set Bounds
ObjetoBD'ObjetoBDP               1  1`{nr = t1,vrr = 10,avir = 10,tsr = 33,impr = 1,milr =
8}++ 1`{nr = t1,vrr = 10,avir = 10,tsr = 39,impr = 1,milr = 8}
```

Box 3.

Liveness Properties

Dead Markings: 24 [6713,6712,6711,6710,6703,...]
Dead Transitions Instances: FCObjetoBDLeAt'AvaliaFCLeAt 1

In *liveness properties*, we have 24 dead markings, that is, there are 24 different ways to the net stopping. In relation to dead transition, there is only one FCObjetoBDLeAt'AvaliaFCLeAt. This transition is dead, since neither conflict occurred when a read transaction was executing and a write transaction was invocated (see Box 3).

Generating the Message Sequence Chart

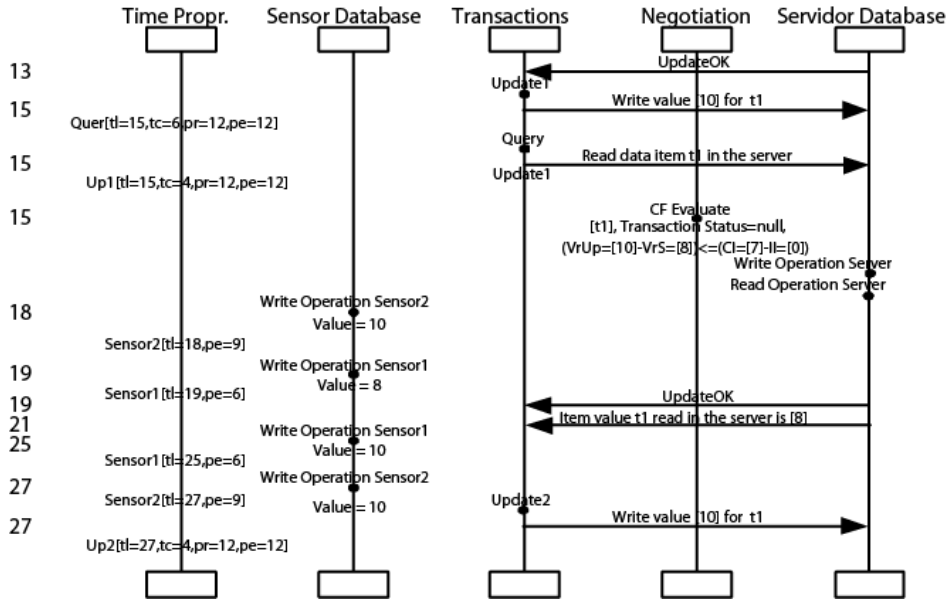
In Figure 8, we have the MSC generated considering the scenario with two sensors acquiring the same data item.

- **Sensor1:** writes periodically in the local database, the release time is 1 time unit (t.u.) and the period is 3 t.u.
- **Sensor2:** writes periodically in the local database, the release time is 9 t.u., and the period is 9 t.u.

Moreover, there are two write transactions and one read transaction.

- **Update 1:** Periodically updates the database server object with respect to the data in *Sensor1.* The release time is 3 t.u., the

Figure 8. Message sequence chart



computational time is 2 t.u., the deadline is 12 t.u., and the period is 12 t.u.

- **Update 2:** Periodically updates the database server object for sensor2. The release time is 9 t.u., the computational time is 4 t.u., the deadline is 18 t.u., and the period is 18 t.u.
- **Query:** Periodically queries the real-time database-server application. The release time is 3 t.u., the computational time is 6 t.u., the deadline is 12 t.u., and the period is 12 t.u.

The release time is the moment when all the necessary resources to the execution of the transaction and sensors are available. Starting from this moment the transaction will be ready to be executed. The computation time is the processing time necessary to execute it. The deadline defines the maximum transaction execution period. Finally, the period defines the periodicity of the transaction and sensor.

In the MSC, it is also possible to verify the QoS functions, where the negotiation between transactions conflicting and the time properties are visible during whole lifetime of a system.

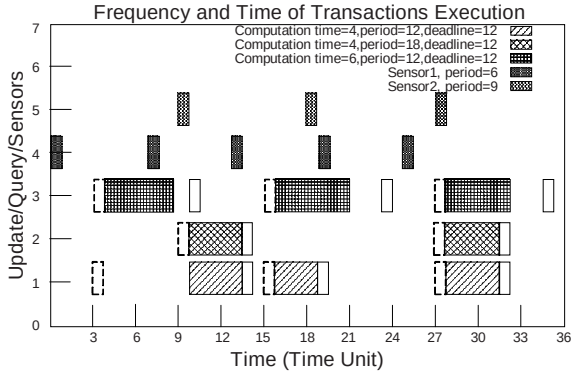
Generating the Timing Diagram

In Figure 9, the timing diagram for the execution of transactions and sensors is presented. In this figure, it is possible to see a representation of the timing model execution for both transactions and sensors, where the execution is represented by rectangles.

In the vertical axis, the transactions and the sensors are represented. In the horizontal axis the release time and the periods for both, transactions and sensors, and computational times and deadline for transactions are represented. For each transaction, we consider three states: start; processing, and committing. The start of the execution of a transaction is illustrated by a dotted line rectangle. The processing is represented by a filled rectangle, and the committing by a continuous line rectangle. For the sensor, we only show the state processing.

In the vertical axis, the *Update 1* transaction is represented in 1. The *Update 2* transaction is represented in 2. The *Query* transaction is represented in 3. The *sensor1* is represented in 4, and the *sensor2* in 5.

Figure 9. Timing diagram



FUTURE TRENDS

As for future trends, we believe that our method should be applied to other application domains. Also, the method can be expanded in order to add more functionality and graphs. Depending on the application domain, the method could be customized and its phases adapted according to the domain features.

Moreover, automatic code generation could be considered. It would be useful to ensure that the programming code for the software is according to the verified and validated model. In this context, we are currently developing an automatic code generator from HCPN model to the Java programming language.

CONCLUSION

In this chapter we presented a method for real-time database verification and validation. The main objective of this method is to make possible the identification of whole components of a system for modeling, analyzing, verifying, and validating them.

The method is based on a model developed using hierarchical coloured Petri nets. The computational tool used to generate the model, verifying the properties and generating graphs for validation by users was the design/CPN tool package.

Using the proposed method, a real system can be studied without the danger, expense, or inconvenience of the manipulation of its elements. It is performed through the analysis of the system's conceptual model and the application of guidelines which drive the developer in the validation and verification activity.

When dealing with complex systems, the process of analysis, verification, and validation needs to be automated. However, the utilization of mathematical models is primordial, which allows automatic verification and validation. It makes it possible to identify various potential deficiencies in the conceptual model, such as contradictions, ambiguity, redundancy, so forth.

REFERENCES

DiPippo, L. C. (1995). *Semantic real-time object-based concurrency control*. PhD thesis, Department of Computer Science and Statistics, University of Island, Kingston, RI.

Douglass, B. P. (2004). *Real time UML: Advances in the UML for real-time systems* (3rd ed.). Boston: Addison-Wesley.

Herrmann, J. (2001). *Guideline for validation & verification real-time embedded software systems*. Software Development Process for Real-Time Embedded Software Systems (DESS), D 1.6.2, V 01.

Jensen, K. (1998). An introduction to the practical use of coloured Petri nets. In W. Reisig & G. Rozenberg (Eds.), *Lectures on Petri nets II: Applications, Lecture Notes in Computer Science* (vol. 1, pp. 237-292). Berlin, Heidelberg, Germany: Springer-Verlag.

Jensen, K. (1999). *Design/CPN 4.0*. Meta Software Corporation and Department of Computer Science, University of Aarhus, Denmark. Retrieved May 19, 2006, from http://www.daimi.aau.dk/designCPN/

Kang, K. D. (2001). *qRTDB: QoS-sensitive real-time database*. PhD thesis, Department of Computer Science, University of Virginia, Charlottesville.

Ribeiro-Neto, P. F., Perkusich, M. L. B., & Perkusich, A. (2003). Real-time database modeling considering quality of service. In *Proceedings of the 5th International Conference on Enterprise Information Systems*, Angers, France (vol. 3, pp. 403-410).

Ribeiro-Neto, P. F., Perkusich, M. L. B., & Perkusich, A. (2004). Scheduling real-time transactions for sensor networks applications. In *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, Gothenburg, Sweden (pp. 181-200). Berlin, Heidelberg, Germany: Springer-Verlag.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design*. Upper Saddle River, NJ: Prentice-Hall.

Selic, B., & Rumbaugh, J. (1998). Using UML for modeling complex real-time systems. In *LCTES '98: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems* (pp. 250-260). Berlin, Heidelberg, Germany: Springer-Verlag.

This work was previously published in Verification, Validation and Testing in Software Engineering, edited by A. Dasso, pp. 111-135, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Section VIII
Emerging Trends

This section highlights research potential within the field of software applications while exploring uncharted areas of study for the advancement of the discipline. Chapters within this section highlight evolutions in social software, state-of-the-art agile software methods, and modeling large software systems. These contributions, which conclude this exhaustive, multi-volume set, provide emerging trends and suggestions for future research within this rapidly expanding discipline.

Chapter 8.1
Social Software Trends in Business: Introduction

Peter Burkhardt
IBM, USA

ABSTRACT

Social networking and Web 2.0 are the hottest words in technology right now; but is there more than just hype? This chapter will define and describe social software and Web 2.0, separate their true concepts from the marketing and buzz, and follow by identifying what might be next for this dynamic technology space. After establishing the social software and Web 2.0 concepts, this chapter identifies the value that they can bring to a company when used in a business context and the shortcomings or pitfalls. This chapter will set the foundation for subsequent discussions of social software and Web 2.0 for specific industry applications.

INTRODUCTION

Social software is the hottest topic in the technology space right now. It is nearly impossible to avoid hearing about Web 2.0 and social networking in mainstream media publications and broadcast news—and it is not just technology magazines or TV shows either. For example, just about every recent issue of *Time* magazine has had an article on Web 2.0, social networking, or something related to those buzzwords.

It is not just the mainstream media that are picking up on this hot topic. The technology itself is being used to tout both its unmatched potential and its less-often-mentioned shortcomings. Web 2.0 is providing the ability for evangelists and skeptics across the globe to weigh in and have equal voice on the very platform they are using to deliver their opinions and thoughts. For those not in the know, this seems like an opportunity that can only be missed at one's own peril.

Every new Web site is clamoring to be known as a Web 2.0 or social networking site and be the next big buzz. Gone are the days when it was good enough to provide a store front on the Internet and generate millions of dollars in sales. These days, you have to have the ability to connect people, provide consumer ratings systems, and allow ev-

erything to be tagged and commented on. Users have become more fickle and will quickly move on to another Web site that enlists more Web 2.0 feel and functionality. Casual Internet users have evolved and do not want to just browse for information; they want to project themselves, get connected to others, and interact or collaborate.

Likewise, traditional corporations are clamoring to put Web 2.0 and social software to work for their own benefit as well. Many managers and executives have said that they need to implement Web 2.0 and social software to have a competitive advantage and not be left behind by the competition. However, many will also admit that they are not quite sure what Web 2.0 really is or what social software really does for their business.

As a result of all this recent press and the amazing amount of hype surrounding Web 2.0 and social software, there are many who believe that these are new concepts (which they use interchangeably), have only recently come to fruition, and have been made available in the last few years. However, like most things, the application of social networking in software is just a new application of a concept that has existed for quite some time. Web 2.0 has been more of a gradual evolution than the instantaneous explosion that the hype would have one believe.

To help one understand where the social software market trend is going, it is extremely important to remove the hype and understand its underlying concepts and history. Only after understanding the foundation concepts of social software can one truly define its value in the marketplace and what social software means to a corporation. The remainder of this chapter will answer the question "What is social software?" and discuss its value to businesses. Subsequent chapters will proceed to examine the relevance of social software and its value as applied in the context of specific business and educational applications.

BACKGROUND

One of the most common issues that leads to confusion in this space and obfuscates the ability to see the value in social software is the interchangeable usage of terms. Just about everyone who talks and writes about this technology space freely and openly substitutes terms such as *Web 2.0*, *social networking*, and *collaboration* as though they all mean the same thing. This only serves to make this market space appear more nebulous than it needs to be and confuses those who are not already knowledgeable on the topic.

Perhaps all of this confusion should be an indicator that there is more marketing hype than substance in the social software trend. However, there are also examples and data to substantiate the validity of the use of social software for the benefit of business. The reality of the situation is that it is quite a bit of both. While there is true business value to be derived from social software, it takes the ability to see through the "buzzword bingo" that regularly occurs.

This is not to say that all terms have been clearly defined to date either. There are many experts in this space who still cannot seem to reach agreement on what certain terms, such as *Web 2.0*, really mean. Therefore, it is important to establish some definitions in order to build our social software foundation of understanding and to see the business benefits of social software.

Social Networking

Contrary to recent technology hype, social networking is a sociology concept that has evolved from sociology studies in the late 1800s and continues to mature even today. While the first studies of social networks may not have been deemed as such, the coining of the term *social network* and the analysis thereof became more mainstream and evolved rapidly within the sociology field in the 1950s to 1970s.

A social network is a grouping of personal relationships that each of us establishes. As opposed to other types of networks, the value of a social network is not in the nodes on the network, which in this case are people, but are in the relationships themselves. Social networking, therefore, is the act of building one's social network. People rely heavily on these relationships to help them with everyday tasks, such as making decisions, forming opinions, and/or finding information:

Our heavy reliance on other people for information and learning is one of the most consistent and robust findings in the social sciences. It also matches our intuition and lived experience in organizations: other people are critical to our ability to find information, learn how to do our work, and develop professionally. (Cross, Abrams, & Parker, 2004, p. 152)

As a result, one of the many uses of a social network is in conjunction with, and as a precursor to, collaboration. Leveraging one's social network vastly improves the ability to collect the right people and information upon which to collaborate toward a common goal. While social networking is a strong precursor for collaboration, the same cannot be said for the relationship when examined in the inverse order.

Social Software

Building upon this definition then, social software is about using technology, and more specifically computer software, to support the process of social networking. It is important to note that the software itself is not performing the activity of social networking. The software is simply being leveraged to support and facilitate the creation and maturation of relationships between individuals, otherwise known as social networking.

The value of social software lies in its ability to do two things. First, it helps to cross traditional barriers that keep individuals from creating relationships. In a corporate environment this could be departmental organization, where people in different departments do not normally talk to each other. The purpose of the software is not to overcome the department structure, but to acknowledge the boundaries and then cross them in order to connect people.

Second, social software assists people to better leverage their existing relationships in order to find knowledge. By exposing contextual relationships between individuals, it can assist in helping to streamline the process of identifying who should be included in a collaborative effort or who knows where to find the knowledge that is being sought. Many people easily confuse this area of value with expertise management. In reality, this may lead to someone eventually locating an expert; however, it is not expertise location in itself because the determination of the expert qualifications is inferred, at best, in social software.

Social Network Analysis

Social network analysis (SNA) is a way of identifying and understanding social linkages and relationships between people. Through the understanding of these relationships, we can then assess information flows and communication breakdowns in a social network. When applied to a business, this information can then be leveraged to identify where talent and expertise could be better applied, how work gets done in a company, and where bottlenecks exist for processes, decision making, and information flow.

SNA starts with gathering relationship information through a series of interviews, questionnaires, and company communication data. After the information has been gathered, it is analyzed to uncover the relationships that exist within an organization. More specifically, the analysis examines the nodes or actors that are in the network, the relationships between the actors, and the attributes that may be affecting those relationships. As a result of the analysis, an organization will

have a better understanding of how things happen and can take actions to increase productivity, efficiency, and/or innovation. Corporate mergers and acquisitions are a prime example of how SNA can benefit a business. After performing an SNA, the two businesses can be organized and melded together in a way that takes advantage of the most efficient processes and people in each business.

Collaboration

Collaboration is defined by Merriam-Webster's Online Dictionary ("Collaboration," n.d.) as "to work jointly with others or together especially in an intellectual endeavor." Collaboration is about more than one person (a team, a group, etc.) working together toward a common goal. There have been many types of software in the past that support collaboration, mainly in the business environment. However, the fault of these applications was that they assumed that the identification of the people being brought together to collaborate had already been conducted. While the tools facilitated the collaboration, they did not ensure that the teams had the right people involved in the collaboration or all of the most appropriate information to collaborate upon. Recent increased focus on social networks has demonstrated their importance as a precursor to collaboration and helped to exponentially increase the value of collaboration.

While collaboration software has been available in the corporate or enterprise space for quite some time (e.g., Lotus Notes, Domino), it has not been prevalent in the public domain and available to end users independent of organizational affiliation. As a result, individuals have had a tough time leveraging the Internet for collaboration in the public domain. That trend is starting to change as new technologies like wikis provide individuals with the ability to organize and collaborate toward a common goal in the public domain. The impact of domains is discussed in greater detail later in this chapter.

Web 2.0

Web 2.0 is the most commonly heard term in the social software trend and the most confusing. Even the experts disagree on the definition. O'Reilly Media is accredited with the creation of the term *Web 2.0* during a conference brainstorming session in 2004 between O'Reilly Media and MediaLive International (O'Reilly, 2005).

In short, Tim O'Reilly (2006, p.1) states, "Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform." It is meant to refer to the second generation of the Internet and is composed of seven core principles.

1. The Web as a platform
2. Harnessing collective intelligence
3. Data as the next Intel Inside
4. End of the software release cycle
5. Lightweight programming models
6. Software above the level of a single device
7. Rich user experiences

Web 2.0 has, without a doubt, become the hottest buzzword today in the technology space and has just about every company and Web site clamoring to claim that they are or use Web 2.0. However, there is also a great deal of misunderstanding, or lack of understanding, as to what Web 2.0 really means when you start to boil down the term.

Critics have stated that Web 2.0 is merely a buzzword, skips over important evolutionary changes to the Web by declaring itself the second generation, does not really have much substance to it, and has yet to produce any meaningful or truly important results. One important critic is Tim Berners-Lee, the inventor of the World Wide Web. In an interview in July 2006, Berners-Lee was asked if it was fair to say that "Web 1.0 was about connecting computers and making information

available; and Web 2 is about connecting people and facilitating new kinds of collaboration." His response was

Totally not. Web 1.0 was all about connecting people. It was an interactive space, and I think Web 2.0 is of course a piece of jargon, nobody even knows what it means. If Web 2.0 for you is blogs and wikis, then that is people to people. But that was what the Web was supposed to be all along. And in fact, you know, this Web 2.0, quote, it means using the standards which have been produced by all these people working on Web 1.0. (Laningham, 2006, p. 1)

In clarifying the definition of Web 2.0, Tim O'Reilly (2006, p. 1) subsequently had this to say: "Ironically, Tim Berners-Lee's original Web 1.0 is one of the most 'Web 2.0' systems out there—it completely harnesses the power of user contribution, collective intelligence, and network effects." So, to summarize the attempts at Web 2.0 definition, O'Reilly coins the concept of Web 2.0 as the next generation of the Internet, gets challenged on that statement, and then clarifies it by saying that the first generation of the Internet is the perfect example of the second generation of the Internet. If that does not confuse people, nothing will.

While there may never be a solid definition for this nebulous concept, for the sake of argument, I will define how Web 2.0 fits into the Internet today in order to discuss it in the context of business. Web 2.0 is a technology concept, unlike social networking (which is a sociology concept). Web 2.0 refers to a style or method for combining existing technologies to empower people. A list of the technologies commonly combined under the Web 2.0 banner is discussed in the next section of this chapter.

The idea is to make the usage of software applications and Web sites faster, easier, and more intuitive for users. This is accomplished through three general thrusts. First, Web 2.0 leverages the power of end users' computers to improve the responsiveness of the user interface (UI) of Web applications. As opposed to previous Web application architecture, where all layers of an application were combined on centralized servers, Web 2.0 moves some of the application logic and management of the UI to the client on the end user's computer. Second, by separating the content from the UI design, users can focus on the publishing and exchange of information without having to know the UI programming required to publish the information. Third, by improving the UI to be more dynamic and responsive, users' attention is drawn to the application and their experience is greatly improved.

These improvements to software applications and Web sites did not happen overnight, though. Many software applications use numbers to indicate versions of the software, with newer versions meant to replace older versions through an installation process. This is where Web 2.0 is very misleading. By using the *2.0* nomenclature associated with software product releases, one might think that Web 2.0 is meant to replace Web 1.0 or is the next version of the Web. This could not be further from the truth. There have been many significant milestones in the evolution of the Web since its inception. The Web continues to change on a regular basis as a result of new ways for combining technologies and concepts, not in a disjointed, iterative manner, but in a smooth evolution.

In reality, Web 2.0 is really nothing new. It is simply a way of recombining and repackaging existing technologies that have been around for quite some time. There are no truly fundamental differences between the technologies used in Web 1.0 and Web 2.0. A brief examination of the most common technologies that are associated with the Web 2.0 banner is included below.

Technologies within Web 2.0

The hypertext markup language (HTML) has been around since the creation of the Internet.

This language is used to create and format the content that is included on Web pages.

The extensible markup language (XML) is a programming language that allows a developer to define content and specific tags that surround the content. Like HTML, the content can be formatted through the use of tags. However, XML also allows the programmer to create his or her own tags and then find and manipulate content through the tags that are associated with the content.

Cascading style sheets (CSS) are used to describe the presentation of a document written in HTML or XML. For each style that is defined in the markup language, CSS tells the client how to display that style. For example, CSS would define that the Header style should be displayed in a bold 12-point font.

JavaScript is a cross-platform scripting language that can be used in a server or client environment to manipulate data and objects. It is most commonly used for client-side Web development to create a more dynamic user experience with a Web page.

Asynchronous JavaScript and XML (AJAX) is a development technique used on Web sites that combines the use of JavaScript and XML to create a more interactive user experience. The concept is to separate the user interface from data in order to be able to exchange data with the server without having to refresh the entire user interface. As a result, the Web page becomes more responsive. AJAX is a key part of Web 2.0 because of its ability to greatly improve the user interface of applications and make them more dynamic.

Really simple syndication (RSS) is a method for publishing information in a structured feed format (usually in XML). This allows clients and other software applications to subscribe to and receive timely information updates. Many people use feed readers to subscribe to a number of Web sites and have the updates aggregated into a single interface. RSS is part of Web 2.0 because of its ability to exchange data in a structured but easily consumable manner between applications.

ATOM is another publishing format that is similar to RSS, but is more robust and flexible. It is based on XML and, while not as widely used as RSS, is gaining market adoption. Like RSS, ATOM is part of Web 2.0 because of its ability to exchange data in a structured but easily consumable manner between applications.

JavaScript object notation (JSON) is a lightweight computer data interchange format. It is a subset of the JavaScript programming language and is commonly used within AJAX as an alternative to using XML.

Tags are keywords that users of a particular application can associate with a piece of content in that application. For example, if a user is posting a blog entry about what he or she ate for breakfast, that entry might be tagged with the keywords *orange juice*, *toast*, *breakfast*, and *eggs* so that other users searching on those keywords will find the blog entry. Users can also tag content created by others.

A folksonomy is a taxonomy where the tags and categorization of data are created and updated in a dynamic way by the consumers of the content. Folksonomies are part of Web 2.0 because they give the responsibility of creating the tags and associating them with the content to the users of the system.

A weblog (blog) is a Web site that is essentially a journal posted on the Web. The most common uses for the journal are either personal publishing, or commentary or news on a particular topic. Blogs are part of Web 2.0 because they enable common users to publish their thoughts and have their voices heard without the need to know how to program Web pages.

Wikis are server software that allows for one or more users to work together to create and edit content. Many wikis serve as a place for multiple people or communities to come together to collaborate on a particular topic. Wikis are part of Web 2.0 because of their ability to facilitate collaboration between users.

A podcast is a radio-style broadcast that is recorded and made available for users to download onto their computers, iPods, or MP3 players. Users generally find and subscribe to podcasts from Web sites as a method to keep up with the information being disseminated on the site. Podcasts are a very valuable tool to both mobile and office workers because they provide the ability to access audio information (e.g., presentations) at the convenience of the listener. Podcasts are part of Web 2.0 because of their ability to be easily created and disseminated by users without costly equipment or extensive experience.

Mashups are Web sites or applications that take multiple data sources and bring them together to provide specialized or situational value. Mashups are part of Web 2.0 because of their ability to let regular users create valuable situational applications without having to perform extensive application development.

Enterprise 2.0

Enterprise 2.0 is a recently identified term that is used to refer to the application of Web 2.0 and social networking concepts in an enterprise business context. First coined by Andrew McAfee (2006b, p. 1), "Enterprise 2.0 is the use of emergent social software platforms within companies, or between companies and their partners or customers."

McAfee (2006a) opines that these new Web 2.0 and social software may replace traditional communications methods in companies because of their ability to make tacit knowledge available to more employees. While it is still too early to tell if this will come to pass, there are some businesses that have taken advantage of them in addition to traditional communications methods.

Anything 2.0

In perpetuating the Web 2.0 moniker, many have started taking the *2.0* label and using it on the end of anything that they wish to express as exciting, new, and/or cutting edge (i.e., Media 2.0, Manufacturing 2.0, etc.). Due to the Web 2.0 buzz, this tactic certainly brings attention where desired. However, like Web 2.0, it does still leave people wondering what the differences from the previous version are and what is so new and great. In a way, this also detracts from the Web 2.0 concept itself because it further reinforces the notion that anything *2.0* is more hype than results.

DO YOU BELIEVE THE HYPE?

With all of the excitement surrounding the Web 2.0 and social software phenomenon, many businesses are in the position of deciding whether they should jump on the Web 2.0 and social software bandwagon or sit this one out. It is not an easy choice when you are a business. The stakes are high and a gamble in the technology space that does not pay off can have quite an effect on the company's bottom line. On the other side of the coin, if the company decides to sit it out and then its competitors can leverage the trend, the company is at a competitive disadvantage in the marketplace and could be equally penalized on the bottom line of the fiscal reports. So, what is a company to do? Should it buy into the hype and take the risk? Is there really something to this social software trend or is this just an example of the emperor's new clothes?

Social software is not a panacea or silver bullet. It will not solve all problems that a business entity has and instantly make things run smoothly while adding millions of dollars to the company's bottom line. It will not instantly resolve tensions in the organization between groups or make people instantly start talking to everyone that they should be talking to. It will not unravel issues with a supply chain or instantly create a distribution network for a product, and it will not result in all of the corporate knowledge being made available online so the company can do more with less people.

Social software can be used to help surface innovative ideas, respond to customers more dynamically, and cross departmental and organizational boundaries where appropriate to get things accomplished. It can assist in rapid growth, the hiring of new talent, and employee retention. It can reduce the loss of tacit corporate knowledge through natural attrition and retirement. It can also improve productivity and allow employees, partners, and customers to do more, faster.

Interestingly enough, there are some teams sometimes found in businesses that may seem like they would be ripe to quickly adopt social software and leverage its capabilities. Contrary to what one might assume, these teams may have the hardest time accepting and adjusting to social software. Two examples of this situation might be with the research and development (R&D) and knowledge management (KM) teams.

Research and development teams have been the staple of innovation in many organizations for the last century. Business entities hire the brightest and most creative minds and give them the funding and freedom to come up with the next set of great innovations that will make the company a success. It is probably not surprising that when the opportunity is created to have others provide innovation and exercise their creativity, R&D may feel a bit like its mission is threatened. However, this is not really the case as experience has shown that R&D is still greatly needed, but its role might slightly change. Instead of being the source of all innovation, it may now participate in the innovation instead of lead, and take on the greater importance by proving the innovation.

Likewise, a knowledge management team may also feel a bit threatened by social software. The reason for this is that the KM team has a wealth of experience in organizing and managing corporate knowledge. Members probably have a rich background in working with knowledge management and collaborative software, and lots of experience in creating information taxonomies. So, the concept of folksonomies may seem to go against everything on which they have worked for the past decade. Again, social software does not make the team obsolete, though; it simply changes the team's role. Social software will allow users to connect and leverage relationships, but it is not a replacement for knowledge management and does not address important aspects of knowledge management, such as the storage and availability of information. Instead, it is a compliment to knowledge management.

Social software is currently in the peak-of-inflated-expectations phase of its hype cycle (*Understanding Hype Cycles*, n.d.). It is not the be-all, end-all in the technology space that some make it out to be. However, it can bring very real results to a business. A business will most likely not use it in all of the ways that it is being used on the social networking Internet sites. If a business takes the time to understand the fundamental concepts behind Web 2.0 and social software, the underlying value propositions associated with each, and select the ones that are appropriate, it can put them to work for the business.

While the benefits are undeniable, many will struggle with measuring those benefits. Most businesses today use objective numbers to measure success. Companies look at things such as revenue, profit, and expenses at the macro level, and sales quotas, wages, product volume delivered, hours worked, and other measurements at the micro level to measure and track progress. In the case of social software, though, the focus is the human element, not finances or output measured.

The problem with measuring the value of social software and social networking is the extreme difficulty of putting an objective measure on a subjective topic. It is just not possible to put a direct measurement on human elements, like relationships or tacit knowledge. For example, a company may deploy a social software environment to help with recruiting new employees. The system will help new employees to grow their network more quickly, which they can, in turn, use to locate knowledge faster. But how

does the company measure that? Any attempt to quantify that value into a dollar amount will need to include a number of assumptions and mostly fictional, best-guess estimates. Inevitably, this is an inaccurate method of measurement, at best, and is usually fraught with skewed data and meaningless results.

Social software may have a direct effect on the company's bottom line, but the direct correlation between social software and items in the general ledger may never be numerically established. Social networking is about personal relationships and personal relationships are about people. In order to make the investment in social software, a company has to value their employees, partners, and customers and be ready to make an investment in people.

FINDING AND APPLYING THE VALUE OF SOCIAL SOFTWARE FOR BUSINESS

It is a good first step to identify that there is value in using social software to support daily interactions and relationships between individuals working at or with a business. However, that is not quite enough to harness that value and put it to work for the business. Not all aspects or implementation methods of social software are appropriate for a business. Likewise, not all existing social networking applications that are available on the Internet are appropriate for use in a business environment. Some sites like LinkedIn (http://www.linkedin.com) offer some value to employees by allowing them to make contacts in other businesses for sales or marketing reasons. Others require more examination and may not necessarily be appropriate.

The very first area of examination must be the concept of, for lack of a better word, what I will call *domains*. In this area of examination, the individual's use of the Internet will be referred to as the public domain; the business IT environment, including use of the Internet to work with customers and partners, will be the enterprise domain. It is extremely important to understand the differences between these domains, the nuances of each domain, and the behavior and motivation of users in each domain. These differences and characteristics will have a profound impact on leveraging social software in a business.

The public domain can largely be characterized as having recreational users with minimal affiliations. Most users will be independent individuals using the Internet for personal use. While many voluntarily identify themselves appropriately, anonymity is entirely possible and is leveraged by some. Their purpose for use will most likely be associated with serving self-interests. This is the domain where many of the most popular social networking sites are currently active. Sites like MySpace (http://www.myspace.com), Facebook (http://www.facebook.com), Flickr (http://www.flickr.com), and others are focused on serving the self-interests of individuals by providing services to connect with each other on an independent, individual level.

The enterprise domain can be characterized as having professional users associated or affiliated with businesses or professional organizations. This association of user with business or professional organization is usually readily indicated in the identity that is used online. The purpose of use will most likely be associated with serving either the associate business or the professional life (e.g., career) of the user and is expected to be used in a professional manner. This is the domain where the traditional business IT communications, financial, sales, marketing, and similar systems are located. Applications such as company e-mail, financial software, collaboration software, portals, supply chain management, and intercompany applications are focused on providing value to the business by facilitating business functions.

The purpose of social software remains the same independent of domain, but the way that social software is valued and, as a result, leveraged

is quite different in each domain. Public-domain users value social software for its ability to facilitate relationships that are self-interest focused. Businesses value social software for its ability to facilitate relationships that benefit the needs of the business. Therefore, businesses do not see a lot of value in having users join MySpace to create a profile and keep up with people's hobbies, likes, and dislikes, but public-domain users do. However, businesses do see a lot of value in using this same concept to provide profiles of their employees that identify their skills, professional affiliations, areas of expertise, and tags for the purpose of locating people with particular skills or knowledge.

To get the most value from social software, businesses should not try to duplicate the most popular Web sites that exist in the public domain and bring them into the enterprise domain. Instead, businesses need to examine the social software and Web sites in the public domain, understand why they are popular and what functions they offer that could be useful to the business, and then implement those functions in areas of their business that will provide the greatest benefit.

For example, podcasting got its start as a way for an audio blog to be taken off line and listened to on personal media devices. This was quickly adopted by the media industry and leveraged to provide audio files such as radio broadcasts and audio journalistic articles for subscribers to listen to on their personal media devices while disconnected. Since then, many businesses have taken a close look at the concept of podcasting as a way to use audio and sometimes video files to deliver information in a quick and asynchronous way to users. As a result, businesses have been able to leverage the concept to accommodate increasingly time-constrained employees, partners, and customers with important presentations, training, recorded meetings, and even sales and marketing materials that are available when they want, where they want. It is even being used to overcome language and cultural barriers by allowing people the ability to pause and listen multiple times to a recording that may be in a second language to them. This greatly increases comprehension and the effectiveness of the information being delivered.

Most importantly, businesses need to understand that all of the individual added value social software brings to the table roll up under one umbrella. The social software trend is about empowering people. Through this empowerment, a business achieves the maximum potential from social software. Freeing people from traditional business control of information and organization and giving them tools that they need to make their voices, thoughts, and ideas heard while transcending boundaries is the value that social software brings to the table.

If a business is satisfied with its current business process and thinks there is no room for improvement, social software is not for it. If it is not interested in improving customer services and does not care about suggestions for cutting operating costs, social software is not for it. If a business is not concerned about employee morale or overcoming communication and cultural barriers, it need not examine social software for its business. If it has all of the innovation that it needs and does not want to find new sources of ideas or better know what its customers need, social software is unnecessary.

However, if a business does care about all of these things, and arguably every business should, then social software brings a great deal of value to the initiatives, not by traditional formal programs or structured initiatives, but through unstructured approaches to tapping into normally unheard ideas. Every company has employees with ideas about how to refine processes that they work with on a daily basis to be more efficient or how to cut wasteful spending. There are plenty of customers and business partners that are willing, and even anxious, to tell a business exactly what they need, how to improve a product, or even become involved with the design and innovation process in

order to contribute their experience and requirements. It is through empowering the people and letting their contributions be heard that the great advances are made.

Fostering innovation in a company is one of the most highly touted uses for social software in business. Empowerment through social software can have a profound effect: "Research and development projects fail more often than they succeed. In fact, out of every 10 R&D projects, five are flops, three are abandoned and only two ultimately become commercial successes" (Rizova, 2006, p. 49). Traditional business approaches to R&D put the source and responsibility for innovation squarely on the shoulders of a single structured department. The brightest minds would be hired to work in this department—people with master's and doctorate degrees and a history of researching, being innovative, and capturing their results in academic papers and prototypes.

Does it not seem that there is something fundamentally wrong with this approach? It is not reasonable to assume that only the people who work in the R&D department can be creative or should be allowed to be creative. Most businesses would agree that there are other people in a company or who work for business partners and customers who are creative. In the overwhelming majority of cases, most businesses do not have the capacity, tools, or methods in place to hear and capture the innovation from those additional sources. That is where social software can support the process. As a rudimentary example, if a business were to implement blogs for their employees and allow them to start expressing their thoughts, frustrations, wishes, ideas, and more, there is plenty of innovation that could be harvested from the blogs and ultimately turned into valuable business assets.

Another major area of benefit under the empowerment umbrella is the ability to provide easier access for users to find, use, and integrate data sources. For example providing RSS or ATOM feeds to key business systems allows a user to aggregate that information to a place that is easy to scan and keep up with (much like scanning the headlines of a newspaper). This makes users more aware and knowledgeable about the business as a whole and makes them more effective in their tasks relating to the business. It also increases their efficiency because what once took an hour of time to seek out multiple data sources and review or interact with them now takes a fraction of the time because they are summarized in a single location. Another example would be to provide users with the ability to create a mashup so that they can generate reports or create situational applications that are context sensitive and specific to their task at hand. By putting this capability in the users' hands, a business removes the shackles of having to request and wait for the IT department to create the needed application or reports.

Additionally, many of the Web 2.0 technology aspects of social software trends provide for a more interactive and responsive user experience in working with data. By improving the user experience of interacting with the data, two things occur. First, users are comfortable using the software and prone to use it more extensively. Second, as power users leverage the software more, additional users will be drawn into using the software through both necessity and word of mouth. This is known as "viral" adoption of the social software system because usage spreads like a virus instead of through traditional IT mandate or formal marketing.

CONCERNS ABOUT SOCIAL SOFTWARE IN THE ENTERPRISE

By removing the traditional business structure and empowering people through such an open solution, most businesses will be legitimately concerned about corporate information security. This is yet another aspect of social software in the enterprise domain that is not an issue or consideration in the public domain. The very thought

of accidentally exposing internal confidential information is something that will immediately send business IT security departments into a full panic. Realistically, there is a balance to be struck between providing openness and ensuring that truly confidential information does not wind up freely available to all.

This balance is struck by understanding which boundaries are going to be crossed by using social software, recognizing the implications of crossing each boundary, and putting the proper guidance and security in place for each. Most businesses already have corporate information guidelines that can be easily extended to cover social software. For example, users should understand that there are different conduct guidelines that apply to blogs only available on a business's internal network vs. blogs available on the company's public Web site. On the internal blog location, the boundaries being crossed are internal (e.g., between departments), with users having the same association (the business) and therefore needing less rigorous guidelines because all users are business employees and working toward the same general business goal. Users should be able to post almost any content with a few exceptions (i.e., no customer names, no personal attacks, etc.). As a result, there may be plenty of blog entries that complain about business processes or strategy, share ideas for new products or solutions, and share personal information.

However, this type of content would not be appropriate to be posted on the business' public Web site. Externally available social software crosses the boundaries between the business and partners, customers, and the general public. As a result, there are additional guidelines that should be applied to address the different associations and affiliations of users of the social software. In this situation, blogs or wikis might be used to publicize new or little-known information about products or solutions and work together with customers and partners on advancing nonconfidential business ideas. Additionally, guidelines will need to be established to deal with information posted by those with negative intentions. These guidelines do not include constructive criticism, but instead are directed toward malicious content and spam.

Most users will intuitively realize what content is appropriate to be posted in social software based on the boundaries that are being crossed. This does not mean that there is no need for a business to deal with exceptions. For example, what happens when someone creates a blog entry that negatively talks about another department in the organization? That is probably not so bad if it is an individual's blog and is expressing the thoughts of the individual. It helps bring an issue to the surface so that it can be addressed and resolved, and everyone can move on. However, what happens if this same entry were to appear in a group or team blog? The author's name posting the entry is explicit, but how does this reflect on the entire team that owns the blog? Members might not all feel this way about the other department, but now these negative thoughts are projected as coming from the entire team. Perhaps in this situation, it is not appropriate and the author should move the entry to his or her personal blog or remove the posting altogether. There is no right answer to these types of scenarios, but some guidelines for working with these situations should be established and publicized to the users of the social software.

Social software also raises privacy concerns in the enterprise domain, but not in the public domain. Most social software solutions work based on a voluntary participation basis, empowering the user to contribute, but not mandating participation in the usage of the software. While this type of opt-in participation does eliminate many of the privacy issues that might be associated, there are some social software (more specifically, social network analysis) tools that analyze corporate communications data to extract social relationship data and make that available. Some would consider this an invasion of privacy

and even have the feeling of the business's "big brother" watching.

However, there is another perspective that data in corporate communications tools belongs to the corporation and not the individual. Additionally, most of these types of social software will analyze only cursory information in the communications data (e.g., e-mail headers, but not the body) and store or transmit only the analysis itself, not the data being analyzed. Businesses have to comply with national employee privacy laws that address the types of employee data that can be made available and used by the business. It is important to always make sure that the business is in full compliance with these laws.

In many businesses, social software is viewed by some as a way for employees to waste time. Initially, some do not see the inherent value in social software, but instead think that it will make the company less productive because users will spend all their time working on things of no value to the business. Those same people are starting to have a realization that they need to use this technology, but are not quite sure just what should be done about it.

This is nothing more than a case of history repeating itself. This same opinion was voiced by many when e-mail first emerged as a new communication tool. It was often viewed by many as a way for people to waste their time sending each other frivolous messages. This same opinion was then applied to the Internet, where users would presumably while away their hours surfing the Internet. Most recently, this same approach has been applied to instant messaging. Many view it as a kid's toy and a way for people to waste time chatting with their friends about nothing in particular. In each of these cases, the naysayers have been proved wrong and each technology has become an integral part of the way business is done on a daily basis. In fact, all of the previous technologies mentioned here have moved into the critical path for businesses and many do not know how they ever operated without them. Social software is destined to follow this same path and become an integral part of doing business.

FUTURE TRENDS

Is there a Web 3.0? Will Web 2.0 one day go the way of Web 1.0? The real question is not whether Web 2.0 will be usurped, as it is only natural that the evolution of the Web continues. The real question is what will be next. At this point, there are two concepts that look to be the most promising and are gaining a large amount of interest. However, they are only conceptual at this point and there has not yet been any real traction in making the concepts into a reality.

The Semantic Web refers to a vision for the Web to become a medium for machines to be able to understand, relate, and compile information without human intervention: "It is envisaged to smoothly interconnect personal information management, enterprise application integration, and the global sharing of commercial, scientific and cultural data" (W3C [World Wide Web Consortium] Advisory Committee, 2007, p. 1). The current configuration of the Web supports the linking and interaction of Web sites, but requires human interaction to interpret the data that is on the Web site. In the Semantic Web, individual pieces of data that are available on the Web are surrounded by descriptive tags that allow computers to understand relationships between pieces of data, as well as the properties of those pieces of data.

By making individual pieces of data readable and understandable to computers, the Semantic Web will truly transition the Web as we know it today from a collection of interacting Web sites to an unbelievable mass of data that is free to be combined and integrated. Allowing machines the ability to combine individual pieces of data will lead to the ability for machines to begin to perform tasks on behalf of humans. For example, a human currently has to submit a query on Google and

then review the results to find the result that best fits the context with which they are interested. In the Semantic Web, a machine would be able to query and review the results for the appropriate context to return to the human.

While the Semantic Web has been discussed as early as 1999, critics have cited that there has not been a great amount of progress made on the Semantic Web coming to fruition over the past 9 years. Also, concerns of privacy and censorship would have been raised because of the ability for machines to understand the data that they are communicating. If this understanding were in place, computers could be made to more rigidly filter or display that data without human approval.

Another direction for the future centers on opening up the individualized social networking Web sites. Many sites now require users to register and then maintain their connections to others within the site. This causes a lot of frustration and extra work when users move to a different social networking site and have to reenter all of their information and connections to friends. Not to mention that their friends might not even be part of the new site; then they have to convince their friends to move over as well. This is a very Web site focused approach to social networking and only a fraction of how true social networking works outside of computers.

To address this issue, there has been much discussion about the Social Graph. The Social Graph refers to a visual mapping of all of our social interactions and connections as human beings. As opposed to having connections to others maintained on a Web-site-by-Web-site basis, it has been proposed that there be one open Social Graph created to manage connections between anyone and everyone. The Social Graph would also take care of maintaining a single identity for a user. The identity and connections could then be mapped to and leveraged by any Web site without having to recreate and store this information in a proprietary way.

This Social Graph further evolves the Internet from a space of interconnected Web sites to a space for interconnected people. After all, users may care about information or knowledge in a contextual and periodic nature (for business use, for personal use, etc.), but care about relationships to others (family, friends, colleagues, etc.) on a constant basis.

In both the Semantic Web and the Social Graph, it is proposed that the Internet move away from a set of connected Web pages and documents and focus connections more on people and data. This shift in direction will prove to be another powerful evolution because it will expose data and relationships at a much more granular level, thereby paving the way for a new level of contextual application. It would not be surprising to see some combination of these two notions in the future.

CONCLUSION

Social software and Web 2.0 are the hottest topics on the Internet. As they continue to draw an incredible amount of attention through the media and online traffic, there is also a great amount of confusion being perpetuated. The intermixing and interchangeable use of terms is, at best, confusing people and leaving others thinking that the social software trend is all smoke and mirrors. While some of the hype surrounding Web 2.0 and social software is just that, there is value in the concepts that underlie the trend.

Just like the value of the social network lies in the relationship between people, instead of the people themselves, the value of defining the terms associated with the social software trend lies in their relationships. Social networking is about the relationships between people. It is about bringing people and knowledge together and better leveraging what you know about who you know. Collaboration is about formalizing the group established through a social network into

a team to work toward a common goal. Web 2.0 is the technology that is being used to empower and facilitate the social networking and, in some cases, the ensuing collaboration.

Many businesses are starting to examine the social software and Web 2.0 trend in order to see how it can add value to their business:

Much like late 1997, when technology specialists were getting asked by senior executives "What is the Internet, exactly, why is it a big deal, and what's our Internet strategy?" The question now is "What's Web 2.0/Enterprise 2.0/social media, exactly, why is it a big deal, and what's our W2.0/E2.0/social media strategy?" (McAfee, 2007, p. 1)

Social software is about relationships and relationships are about people. In order to fully leverage the social software trend, a company needs to value its human resources and be prepared to invest in people. There may never be a direct correlation between the deployment of social software and the bottom line of the ledger sheet (without a few bits of creatively fictional logic), but the company will surely see the value reflected in subjective measures, like customers' satisfaction, increased productivity, enhanced cost cutting, improved employee morale, and more. All of these factors will translate to an improved financial bottom line.

Businesses will need to apply the concepts behind the social software trend to areas of their business where they can be effectively used to unlock and open up the business and provide a voice to users. By empowering users, social software will allow for new sources of innovation and collaboration. According to a 2006 IBM Global CEO study (IBM Global Services, 2006), only 17% of CEOs ranked internal R&D as a source for new ideas. This means that overwhelmingly, businesses will be looking to customers, partners, and other company sources for the ideas and innovation that will power the future of their business.

Unlocking all of this value will not come without some challenges. It is encouraging, however, to know that those challenges can be overcome. Issues of security, privacy, and adoption of social software will emerge within businesses. Nevertheless, these issues have emerged with previous technologies, such as e-mail or the Internet itself, and have been addressed accordingly.

Now that we have had a look at how social software trends can impact businesses, subsequent chapters will examine how elements of social software are being adopted by businesses in different industry segments.

REFERENCES

Collaboration. (n.d.). *Merriam-Webster's Online Dictionary.* Retrieved December 15, 2007, from http://www.m-w.com/dictionary/collaboration

Cross, R., Abrams, L., & Parker. (2004). A relational view of learning: How who you know effects what you know. In M. L. Conner & J. G. Clawson (Eds.), *Creating a learning culture: Strategy, technology, and practice* (pp. 152-168). Cambridge University Press.

IBM Global Services. (2006). *Expanding the innovation horizon: Global CEO study 2006.* Retrieved December 15, 2007, from http://www-935.ibm.com/services/us/gbs/bus/html/bcs_ceostudy2006.html

Laningham, S. (2006). *developerWorks interviews: Tim Berners-Lee*. Retrieved December 15, 2007, from http://www.ibm.com/developerworks/podcast/dwi/cm-int082206.html

McAfee, A. (2006a). Enterprise 2.0: The dawn of emergent collaboration. *MIT Sloan Management Review, 4*(3), 21-28.

McAfee, A. (2006b). *Enterprise 2.0, version 2.0.* Retrieved December 15, 2007, from http://blog.hbs.edu/faculty/amcafee/index.php/faculty_amcafee_v3/enterprise_20_version_20

McAfee, A. (2007). *How to hit the Enterprise 2.0 bullseye*. Retrieved December 15, 2007, from http://blog.hbs.edu/faculty/amcafee/index.php/faculty_amcafee_v3/how_to_hit_the_enterprise_20_bullseye

O'Reilly, T. (2005). *What is Web 2.0?* Retrieved December 15, 2007, from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

O'Reilly, T. (2006). *Web 2.0 compact definition: Trying again*. Retrieved December 15, 2007, from http://radar.oreilly.com/archives/2006/12/web_20_compact.html

Rizova, P. (2006). Are you networked for successful innovation? *MIT Sloan Management Review, 47*(2), 49-55.

Understanding hype cycles. (n.d.). Retrieved December 15, 2007, from http://www.gartner.com/pages/story.php.id.8795.s.8.jsp

W3C (World Wide Web Consortium) Advisory Committee. (2007). *Semantic Web activity statement*. Retrieved December 16, 2007, from http://www.w3.org/2001/sw/Activity.html

This work was previously published in Social Software and Web 2.0 Technology Trends, edited by P. Deans, pp. 1-16, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.2
Social Software and Language Acquisition

Sarah Guth
Università degli studi di Padova, Italy

Corrado Petrucco
Università degli studi di Padova, Italy

ABSTRACT

This chapter describes how the social software tools that characterize Web 2.0, such as wikis and blogs, can be used as a valid substitute for more traditional Learning Management Systems in the context of e-learning and blended learning language courses. First, we will give a brief overview of how the educational arena is changing and the role social software can play in promoting these changes. Then we will describe two experimental courses carried out at the University of Padova using social software. The chapter ends with a discussion of the role of these tools in formal education. The aim of the chapter is to show how these tools allow language educators to take network-based language teaching beyond the limits of planned classroom activities, offering students new opportunities to access and produce real language in real situations.

INTRODUCTION

The evolution of e-learning and computer-assisted language learning (CALL) is inherently connected to the evolution of technology and theories on language learning pedagogy intertwined with ways in which society changes. The advent of the Internet in the 1990s led to a shift toward a focus on authentic communication, task-based learning, and network-based language teaching (NBLT) using sociocognitive approaches. NBLT involves a shift from learners' interaction *with* computers to interaction with other humans *via* the computer (Kern & Warschauer, 2000). Whereas the Internet was initially a place where information and knowledge were delivered by "experts" and information was simply retrieved, toward the end of the 1990s, the Web started to become a place where users, everyday people, *produced* and *shared* content in ever-growing global communities. Contrary to the dominant epistemology of Western culture according to

which knowledge is created and transmitted by experts, the new way of using the Internet was based on the concept of the "wisdom of the crowds" (Surowiecki, 2005). In 2003, Tim O'Reilly and Dale Dougherty dubbed this new revolution "Web 2.0" (O'Reilly, 2005).

Web 2.0 is characterized by what can generically be called "social software": different types of software that enable people to collaborate and create and join online communities. What is particularly interesting is the widespread use these tools have achieved in a very short time, especially among young people, the so-called Digital Natives (Prensky, 2001) or Net Generation (Oblinger & Oblinger, 2005). Whether people are using these tools to express their creativity (YouTube, Flickr), to share their thoughts (MySpace, blogs), or to share their knowledge (Wikipedia), they are creating and participating in online social networks in a way that was not imaginable just a few years ago.

The aim of this chapter is to argue that not only can language educators not ignore this revolution, but they should embrace it. The benefits of using social software for language learning go beyond simply providing new tools students can use to communicate. They promote social networking on a global scale and knowledge sharing and creation beyond the classroom, thereby giving students opportunities to access, use, and produce authentic language in real-world contexts. In this chapter, we will first provide a brief overview of the implications of Web 2.0 and social software for education in a general context. This information served as the foundation for an action research project that was started at the University of Padova in the spring semester, 2006. We will then describe the second stage of the project, which involved the use of wikis, Skype, blogs, and other Web 2.0 tools in an advanced English as a Foreign Language (EFL) course. Though the research is not yet complete, the initial results are positive and confirm results from other studies. Finally, the chapter ends with a discussion of the future of these tools in education.

WEB 2.0 AND SOCIAL SOFTWARE FOR EDUCATIONAL PURPOSES

The educational arena today is finding it necessary to react and adapt to the shift from an industrial to a knowledge-based economy. We are now living in an information society where the way knowledge is created and organized and the very nature of knowledge have changed. The ways knowledge is represented have always been strongly influenced by the tools used to express it. Today it is impossible to think of knowledge without associating it with tools such as search engines, Web sites, repositories of learning objects, and more recently, social software tools such as blogs and wikis. Today's students need to learn how to operate effectively in today's information overload and, at the same time, how to become creators of knowledge. Upon graduation, they will find themselves looking for work in a global knowledge-based, networked economy where they will need to be skilled in collaborative and creative project-based work and critical thinking (Bruns & Humphreys, 2005). At the same time, we must also help students develop "the resources and skills necessary to engage with social and technical change, and to continue learning throughout the rest of their lives" (Owen et al., 2006, p. 3). Language acquisition especially is a lifelong process that cannot end with traditional education, but rather must be cultivated throughout life.

The advent and success of Web 2.0 technologies have led many to speak of e-learning 2.0. Much of the focus during the first decade of e-learning was on the tools, or learning management systems (LMS), through which material could be delivered while less focus was given to the actual pedagogy to be used. We can distinguish between two approaches to information and communications technology (ICT): a technology-centered

approach and a learner-centered approach (Mayer, 2005). The former generally fails to lead to lasting improvements in education. A learner-centered approach, on the other hand, can help students and teachers learn and teach through the aid of technology with a focus on how ICT can be used as an aid to human cognition helping students solve complex tasks and deal with today's information overload. Not only has the quantity and kind of information students access today expanded exponentially in the last few years, but the way they interact with the information and the global communities creating it has changed dramatically as well.

As suggested by Bonaiuti (2007), this change has two specific characteristics: a shift from formal learning to informal learning and a shift from content-based learning to collaborative learning. Pedagogy based on a social-constructivist approach and supported by tools readily available online and part of the everyday lives of students can help create an active learning environment in which students and instructors work together to solve problems contextualized in the real world. What's more, in this context, the collaboration does not stop outside the classroom, but rather flows over into personal lives and develops through the use of social software tools such as blogs, wikis, and forums. To do this, an effort must be made to try to integrate formal and informal learning (Cross, 2006). This is certainly not a new idea (Dewey, 1965), but it is one that is worth experimenting with through the social interactions mediated through the Web, which thus becomes a powerful *zone of proximal development* able to organize knowledge in multiple and flexible contexts (Spiro et al., 1991). It is a model that follows a participative approach to knowledge creation (Sfard, 1998) and is closely tied to social constructivism (Brown, Collins & Duguid, 1989; Jonassen, 2000) by which learning is above all an active process that is socially situated and aimed at solving real problems. Rather than refer to this new trend as e-learning 2.0, Futurelab[1] proposes the term "c-learning," which could be intended as "community learning, communicative learning or collaborative learning, [because] at its heart learning is a social process" (Owen et al., 2006, p. 11).

Social software follows the c-learning trend. The use of social software in education focuses on activities rather than applications (i.e., all those activities that involve sharing knowledge [blogs], collaboratively creating knowledge [wikis], and managing knowledge [social bookmarking]). In education, adopting social software has several advantages with respect to other computer-mediated communication tools.

1. **Enabling communication among many people beyond the classroom:** Unlike many closed LMSs, social software tools give students the opportunity to interact with experts and novices alike on a global scale beyond planned classroom activities.
2. **Providing new ways to share and create content:** These tools provide ways to share and create content in systematic, organized ways (e.g., blogs, photo sharing on Flickr at http://www.flickr.com/, and video sharing on YouTube at http://www.youtube.com/). They also offer students the opportunity to share knowledge beyond the classroom through, for example, public wikis. The process of sharing knowledge often involves the process of transforming tacit knowledge into explicit knowledge, which is where true learning takes place.

   Some social software tools make it possible to collaboratively collect and index information in such a way that not only is it easier for users to access their own resources, but it is also possible to share them with others in the community and exploit other users' online searches. In this way, they function as a filter for the information overload on the Web (e.g., social bookmarking on http://del.icio.us and social annotation on http://www.

diigo.com). This is what Giger (2006) has called "participation literacy" (i.e., it is no longer the individual that filters information, but the collectivity). Another useful tool for filtering information is feed aggregators; syndication makes it possible for information to come to you rather than you going to it. Some tools offer remote hosting (e.g., Bloglines at http://www.bloglines.com/), which means users can access their feeds from any computer, while others are installed locally on a user's computer (e.g., Sage, an extension for Mozilla Firefox at http://sage.mozdev.org/).

4. **Integrating many types of media (audio, video, images):** With the growing number of free tools to create multimedia, the increase in the number of sites dedicated to sharing these resources, and the spread of broadband, students now have the possibility to better express their ideas and creativity by integrating audio, video, and images into the contents they create online.

When several social software tools are used together, students learn how to learn by (1) taking on responsibility for managing the social software tools; (2) working cooperatively and learning from each other; and (3) developing online research skills (Mejias, 2006). All of these activities enhance a student's sense of responsibility and collaboration, both of which can help foster autonomy. If students learn how to exploit the Web for their autonomous language learning in the context of online communities, they will be able to continue the language acquisition process in their informal learning as well. Social software tools take the practice of NBLT beyond the limits of the language learning classroom into the global communities of Web 2.0.

In the next section, we will describe how these tools were aggregated and used in an advanced EFL course at the University of Padova.

BLENDED EFL COURSES USING SOCIAL SOFTWARE

In spring of 2006, an action research project was set up to study the potential of social software to effectively teach blended learning courses for English as a Foreign Language (EFL). The first stage involved an experimental course that was held at the Faculty of Engineering using blogs and a wiki to conduct an upper-intermediate EFL course. Overall, students proved to be highly motivated by using these tools, participated actively, and were satisfied with the course. A careful study of the empirical data corpus led to a redesigning of the course for 28 second-year EFL students in a graduate course in International Communications Studies as the second stage in the project. The year-long course was divided into two semester-long courses: TulanePadovaXchange, a telecollaboration project (Belz, 2005; O'Dowd, 2005) with Italian language students at Tulane University in the United States and EFL students at the University of Padova, and BloggingEnglish, a course aimed at exploring the potential of Web 2.0 tools for autonomous and collaborative language learning in the other semester. In order to work more effectively, students were divided into two groups of 14. One group did the exchange in the first semester, while the other did the blogging course; the groups switched courses in the second semester.

Various data sources were used in the research study, including the following: participant observation (teacher's field notes), posts in students' personal blogs, transcripts of students' online correspondence on the wiki and recordings of their Skype conversations, informal interviews, comparison of students' writing at various points throughout the course, and end-of-course questionnaires. As the courses were blended and students were asked to reflect weekly on their learning process in their personal blogs, it was possible to read students' impressions and then discuss them in class as the courses progressed.

In this way, the instructor was able to check her interpretations of the data with students.

In the following paragraphs, we will describe each course's structure, language learning aims, and preliminary results. This is followed by a brief description of how assessment was carried out.

Padova-Tulane Exchange: Telecollaboration on a Wiki

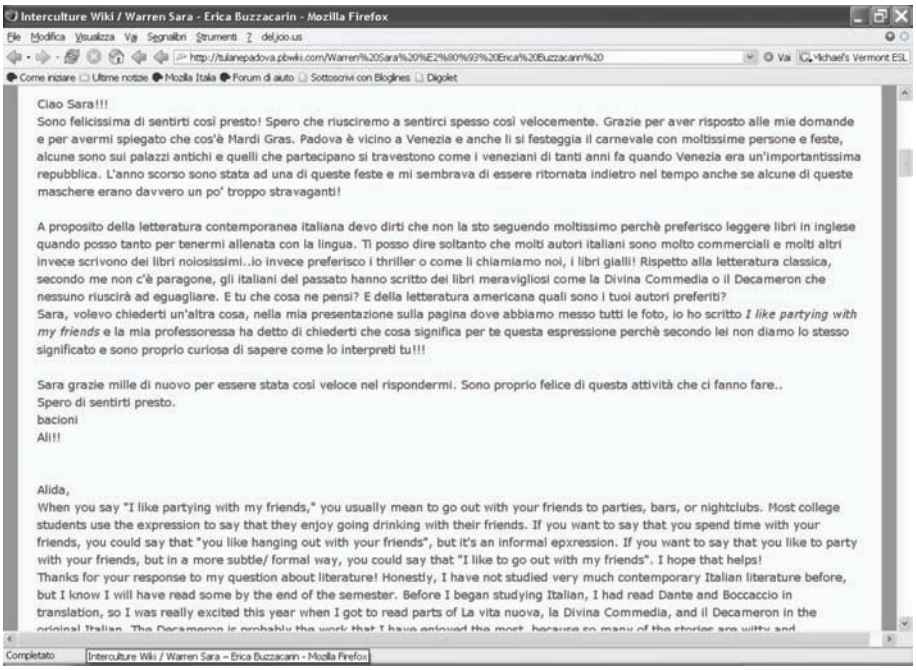
Telecollaboration projects have been carried out in recent years using many tools, from simple e-mail exchanges to specifically designed software (e.g., *Cultura*) (Furstenberg et al., 2001). Regardless of the tool chosen, "the underlying rationale is to provide the members of each parallel class with cost-effective access to and engagement with age peers who are expert speakers of the language under study in an effort to increase intercultural awareness as well as linguistic proficiency, to increase the authentication of foreign language use in the tutored setting, and to broaden the range of discourse options and subject positions available to classroom learners of language" (Belz, 2005, p. 1). Using a wiki as a platform offers advantages that other tools do not. A wiki offers the possibility to create an ever-growing repository of cultural knowledge that can be used and re-used by students in various learning contexts over time (Godwin-Jones, 2004). Furthermore, writing on a wiki involves collective authoring, which by its very nature requires significant amounts of reading and peer correction. Students can also use the revision history function available on most wikis to keep track of changes in a piece of writing over time, promoting close reading, revision, and tracking of drafts in order to focus on writing as a process rather than a product (Lamb, 2004). Therefore, a telecollaboration project called TulanePadovaXchange was set up between the University of Padova and Tulane University using a wiki (http://tulanepadova.pbwiki.com/) and Skype to add an oral component to the course. The course was blended; students had one two-hour face-to-face lesson a week for 10 weeks (one semester) and were expected to do four to six hours online per week.

The Exchange

During the first semester, students spent several weeks exchanging messages on the wiki based on the outcomes of tasks proposed by the *Cultura* model. Each group of two or three peers had a page they used as one might use a discussion thread in a forum (see Figure 1). Two advantages, however, to using the wiki were (1) all of the information exchanged was present on the same page and, therefore, easier to access for reflection and (2) all of the students could see the other groups' exchange pages so they could learn not only from their own peers but from the other exchanges as well. Due to scheduling difficulties, the Skype exchanges took place outside of class time and, at most, students had three 15-minute conversations.

In the second semester, the teachers in the United States and Italy managed to organize one hour a week in the respective language labs when students could meet online so that rather than carrying out the exchanges on partner wiki pages, all exchanges took place orally via Skype interviews. The sixth and final meeting was a videoconference between the two groups carried out using Skype. A significant increase in involvement and motivation took place on both sides of the Atlantic in the second semester. Although students communicated frequently on the wiki in the first semester, the focus of their exchanges seemed to lack purpose leading to a decrease in motivation throughout the course. The exchange was not an integral part of the American students' grade or coursework, but rather an additional activity. During the second semester, the teachers in the United States and Italy chose three topics students were to explore (recycling, alternative energy resources, and water resources) and integrated the exchange into both course syl-

Figure 1. Example of a wiki page for peer exchanges; two Italian students and one American student exchange messages in their native languages



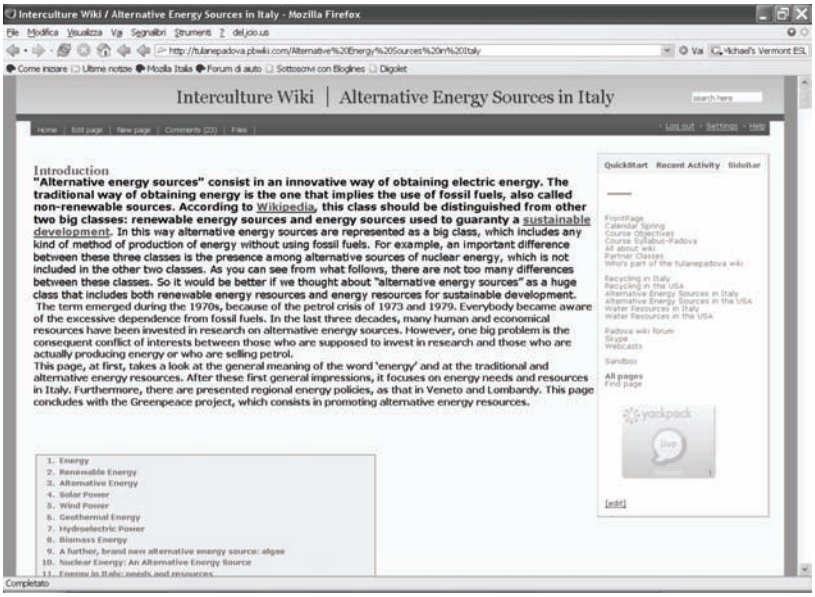
labi. The regular use of Skype in the language lab also increased motivation since students in the two institutions had previously been given very few opportunities to speak in the target language and even fewer to speak with native speakers. All the peer conversations were placed online after each exchange in both streaming format and as downloadable MP3 files offering students on both sides of the Atlantic the opportunity to listen not only to their own conversations but to their peers' as well, as was the case with the wiki pages in the first semester. In her blog, one student noted that "one of the most positive aspects of Skype exchanges is that we can all benefit from every single conversation, that is, not only from the exchange with our American conversation partner, but also from the other exchanges: each of us can learn from the experience made by other students. Therefore… the more we listen and talk to our peers, the more we learn…!" Finally, as O'Dowd (2007) found, the students realized the benefits of telecollaboration with respect to traditional culture learning materials: "[T]he online exchanges provided learners with a different type of knowledge. … As opposed to objective factual information, the accounts which students received from their partners were of a subjective and personalized nature" (p. 146).

The Interculture Wiki Pages

The culture pages in the wiki were developed exclusively by the Italian students in English as the writing component in the course.

During the first semester, the Italian students were given the freedom to create and develop wiki pages on a topic of interest to them based on issues that came up during the asynchronous exchanges with their American peers. Once pages had been created, students were then encouraged, and eventually forced, to read and edit each other's pages. When students create their own pages,

Figure 2. Example of a wiki page before the final edit; each page had the same structure: introduction, table of contents, contents



they maintain a strong sense of ownership. In Lund and Smordal's (2006) wiki experience, the authors state, "[L]earners did not immediately embrace any notion of collective ownership or epistemology but continued a practice where the institutionally cultivated individual ownership persisted" (p. 41). Indeed, writing collectively did not come naturally, and students had to be guided through the process. Collective authoring gives students an opportunity to focus on the writing process (e.g., from collecting information to organizing, rereading, and doing a final edit. They eventually learn that it is actually easier to correct a peer's work and find where meaning is unclear (Berg, 1999) rather than correct their own contributions. The comment function in the wiki allowed students to asynchronously negotiate content and structure.

An attempt was made in the second semester to encourage collective authoring by creating the following six wiki pages for the students to develop: Recycling in Italy/the USA, Alternative Energy Resources in Italy/the USA, and Water Resources in Italy/the USA. The Italian students had to collectively develop the Italian page as preparation for the Skype exchanges (see Figure 2) and then, based on the information exchanged with their American partners, had to develop the American page. An analysis of the six wiki pages shows significant improvements in their ability to collaborate and write collectively with this organization. For example, they learned to propose structure before writing and discuss which part of that structure each individual student was interested in developing. The benefit of working asynchronously is that students must critically read what has been written by others before contributing. As one student commented, "Thanks to this task I was obliged to search some news and first of all … **READ, READ, READ**! My colleagues had already written some generic aspects and so I could begin my discovery from that." In the second semester, there was much more knowledge sharing as students were developing only six pages and developing them together. Nonetheless, they still felt strong ownership for

their personal contributions and had difficulty editing each other's work, as was the case in the first semester. The overall feeling toward the end of the experience was, however, positive. As one student commented, "Developing our six wiki pages and conversing every week with my American peer made me understand the importance of 'learning-by-doing', co-operation and negotiation and, most importantly, made me reflect on the close, triple connection existing between the communicative function of language, the use of technology and the process of creating and sharing knowledge."

Finally, one of the main aims of the project proved successful during the shift from the first to the second semester: creating a repository of shared knowledge. Students during the second semester continued to contribute to the wiki created during the first semester. Rather than being a hindrance to production (i.e., "let's copy what they've done"), reading what other students had written and experienced proved to be motivating for students during the second semester. It is the intention of the authors that the wiki continue to grow with future semesters in such a way that students continue to learn from their peers' experience, avoid some of the pitfalls encountered, and help the repository of intercultural knowledge grow and improve.

Language Learning

The three linguistic aims established at the beginning of the course were:

- To improve students' reading skills and vocabulary in the target language by reading messages written by native speakers of the target language.[2]
- To improve students' written accuracy in the target language via production of contents in a wiki in the target language.
- To develop students' speaking fluency in the target language via interviews in Skype.

During the first semester, students were asked to carry out their asynchronous text exchanges in their native language in order to provide each other with accurate, authentic language and to have the freedom to fully express their ideas regarding their culture. The most commonly cited benefit was the opportunity to learn "real" language, new idioms, slang words, and so forth, which students then reproduced when developing their wiki pages.

The questionnaires and focus groups (self-assessment) and a careful analysis of the corpus of written (wiki) and spoken (Skype recordings) text confirmed that the second two aims were achieved in both semesters. The second aim involves improving reading skills as well, since before you add a contribution to a wiki, you must first carefully read what others have already written. As one student commented, "Collaborative writing is a very stimulating activity, it fosters multi-steps collaboration (reading, writing-integrating) between wiki-contributors." Furthermore, given the nature of the tasks, students had to do a significant amount of searching and reading online to develop the factual contents of their wiki pages. During the second semester, as they developed three pages about Italy, they had to practice their abilities to read in Italian and then produce new knowledge in English. One student commented, "Finding pages in Italian is not only a translation exercise but it involves many other skills, such as reading, summarizing, writing, thinking in a critical way, etc." Finally, accuracy regards not only linguistic accuracy, but also issues such as structure, content, and plagiarism. An analysis of the development of the wiki pages throughout the second semester course shows that with each new page (of the total six), students' abilities to structure the pages, organize the information, and provide references improved. Final feedback on linguistic accuracy was provided by the teacher. In groups, students then had to make a final edit of assigned pages.

There is no doubt that the opportunity to speak on Skype improved students' confidence in speaking in English and ability to express themselves in English. They learned how to deal with uncomfortable pauses, ask for clarification when they did not understand, and, in the process, learn new words and expressions. As one student wrote in her blog, "Skyping pushes me to enrich my vocabulary because, at first, my aim was that of speaking with a native in order to improve my English; now, I want to communicate, convey ideas and for doing so, I must express myself in a more appropriate way." What is more interesting, however, is that this process also helped them learn how to communicate more effectively with non-native speakers of Italian, as students conversed in both languages. Italian students are not accustomed to speaking to non-native speakers of Italian who are trying to communicate in Italian. At first, many found it frustrating, but they became much more aware of the importance of the role of the native speaker when communicating with non-native speakers. For example, they learned to speak more slowly, how to piece together the disjointed sentences their American peers were producing, and understand the American students' pronunciations in Italian. Given the growing number of immigrants in Italy, this can be considered a particularly important social skill.

BloggingEnglish: Reflecting on Social Software and Language Learning

Many blended language courses are carried out using forums or other tools in traditional, closed LMSs to promote communication. However, if current language learning aims to offer students authentic opportunities for accessing and producing language, we asked ourselves if blogs and other Web 2.0 tools were not more suitable. According to Ferdig and Trammell (2004), "[K]nowledge construction is discursive, relational and conversational in nature. Therefore, as students appropriate and transform knowledge, they must have authentic opportunities for publication of knowledge" (p. 1). Blogs do just this if they are public, which means they can be read by anyone on the Web. This gives students a real audience for their writing, beyond the teacher and the classroom, which in turn increases students' sense of ownership and responsibility for what they write (Godwin-Jones, 2004). Furthermore, the journal-like nature of blogs and the fact that posts are archived and can be retrieved promote reflective analysis of their writing (Bryant, 2006; Ferdig & Trammell, 2004). Finally, though at first blogs may appear to be all about writing, they are first and foremost about reading. Therefore, an advanced ESL course, called BloggingEnglish (http://www.bloggingenglish1.blogspot.com/) was designed using a blog and e-tivities (Salmon, 2002) that focus on familiarization with Web 2.0 tools.

The Course

The course was blended, and students had one two-hour face-to-face lesson a week for 10 weeks (one semester) and were expected to do four to six hours online per week. The course blog was used as a substitute for the forums found on traditional LMSs, and students developed their own personal blogs as a place where they could reflect on their learning experience, express their own creativity, and interact with their classmates. This was believed to be important because, as Godwin-Jones (2006) states, "[I]t is possible to create a more student-centred learning environment using blogs, particularly if students create blogs that they control and whose content they own" (p. 13). Throughout the 10-week course, students were given weekly e-tivities to complete, either on the course blog or on their own personal blogs.

The course had three main objectives:

- Students will develop competency in the use of blogs, wikis, social bookmarking, Web

syndication, and other Web 2.0 tools, and explore their potential for language learning.
- Students will improve their written communicative fluency in English and their reflective and critical thinking skills by publishing their thoughts, opinions, and reactions on a course blog and on a personal blog.
- Students will develop their practical research skills using online information networks as they look for, find, and share online resources.

Each week students were faced with a new Web 2.0 tool with the aim of helping them improve their information literacy skills and participation literacy skills (Giger, 2006) (i.e., how to exploit the knowledge of other Web users and share their knowledge in global communities) (see Table 1). They were encouraged to find blogs where the blogger regularly posted useful information and resources with regard to language learning or their future professions (e.g., as translators or tourism professionals) and podcasts containing regularly updated audio files useful for developing listening skills. Students learned how to use a feed aggregator to have updated information from blogs and other Web sites come to them rather than having to regularly check all the sites they had found. They learned how to participate in social bookmarking (i.e., to exploit the Web sites other members of the community had already found and add their own contributions). They discovered how to effectively use sites like YouTube to practice their listening skills in an interesting, personalized way by choosing topics of particular interest to them (see Figure 3 for the e-tivity on YouTube and Figure 4 for a student's post on her personal blog and a peer's comment). Toward the end of the course, two e-tivities were dedicated to contributing to an existing public wiki. These tasks helped students understand not only that they could gain from other people sharing knowledge but that they too had knowledge to share. As one student commented at the end of the tasks, "We are given the great opportunity to 'tell the world' what we've learned so far during the English course." In other words, students learned to create knowledge, share knowledge, and benefit from the sharing of knowledge of other Web 2.0 users.

Table 1. Weekly e-tivities in the second semester

e-tivity 1	Let's get started	To become familiarized with the course blog; introductions
e-tivity 2	Exploring the blogosphere	To learn how to exploit the blogosphere as a source of information
e-tivity 3	Developing your own blog	To develop a space to express your own creativity and opinions
e-tivity 4	Social bookmarking	To learn how to save all the resources you find on the Web and exploit those found by others
e-tivity 5	Feeds and feed aggregators	To learn how to have selected updated information come to you
e-tivity 6	YouTube	To learn how to exploit video online for language learning and discussion
e-tivity 7	Let's wiki on a public wiki	To experiment with writing on a public wiki
e-tivity 8	Cleaning up the wiki	
e-tivity 9	PLEs	To develop a mindmap of your personal learning environment

The final e-tivity aimed to have students stop and reflect on what they had done and learned by individually developing a mindmap (using the open source software FreeMind) of their own personal learning environment (PLE) and publishing it on their personal blogs. A PLE is "a combination of the formal and informal tools and processes we use to gather information, reflect on it, and do something with it, which is essentially what we mean when we talk about learning" (Martin, 2007). This seemed an appropriate way to end a course on social software as these tools are flexible and can be personalized, and as such allow students to develop their own platforms for their informal learning.

Students found that being faced with a new tool each week was challenging. The general reaction was initially frustration and a sense of not being able to accomplish the task. These 24-year-old Italian students are not, in fact, Digital Natives, and there was no overlap with what they did in the classroom and their personal lives, as most had never seen a blog and none had ever used a wiki, social bookmarking, or feed aggregators. This initial frustration, however, inevitably turned into appreciation for what the new tools had to

Figure 3. Example of an e-tivity based on Salmon's model (2002): spark, purpose, task, respond, timeline. The figure shows e-tivity 6 on exploring YouTube. All students completed the optional part of the task (3)

YouTube
This week we're going to explore (no panic!) YouTube. YouTube is the largest online video sharing tool. It is probably the fastest-growing tool on Web 2.0. According to Wikipedia (http://en.wikipedia.org/wiki/Youtube), "the company was named *TIME* magazine's "Invention of the Year" for 2006.[2] In October 2006, Google Inc. announced that it had reached a deal to acquire the company for US$1.65 billion in Google's stock." In other words, it was something special if Google was willing to invest that much money. You might be asking yourselves, as you often have in this course, 'cosa c'entra con l'inglese?' Well, let's see if you can figure out how it may be useful for your language learning, or blogging, or anything else!
Sarah

Purpose: To start reflecting on the potential of YouTube for language learning and Web filtering.

Task:
1 – **Go to the group I have created for us on YouTube. (http://www.youtube.com/group/bloggingenglish)**. There you will find three videos and three discussions. I have invited you to participate, though the invitation should not be necessary. Should you have problems accessing, send a comment to this post. Before clicking on the videos, click on each discussion and read my message first.

2 – **Complete the tasks in the discussions in YouTube.** As indicated in my comments, please complete the tasks on YouTube by adding your own comments.

3 – **Find a video for your blog.** This part of the e–tivity is optional. If you so desire, search for a video on YouTube that says something about you and add it to your blog. How do you do this? When you click on a video, on the right you should see information about the video including 'embed'. Copy the html, go to blogger, click on new post, click on html and paste the html there. Post the message and the video should appear. Again, if you've got questions, feel free to ask via the comment function here! Note: the videos I've put here so far have been from YouTube and I'm no expert so it can't be hard!

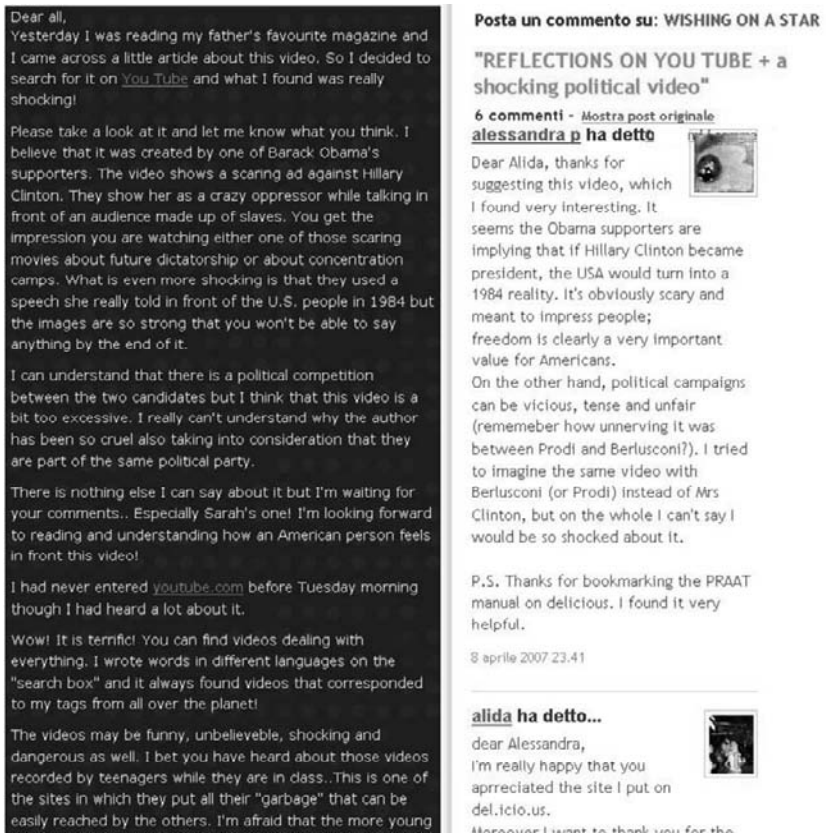
4 – **Reflect**. What do you think are the consequences of this tool? Do you think it has been / can be useful for your English language learning? And for other purposes? Make a blog post reflecting on your YouTube experience.

Respond: Go to each other's blogs (using bloglines!) and read your peers' reflections. Have they said things similar to what you thought? Different? What does this mean? Add comments that help develop the debate on the potential or non-potential of these tools in language learning. Remember, what you contribute may help future students! You've got two weeks to complete this one, so take all the time you need to reflect!

Happy Easter!!!

Timeline: Wednesday, April 11 (task), Tuesday, April 17 (respond).

Figure 4. Example of a student post following an e-tivity and peer comments using the comment function; in this screenshot, the student is responding to e-tivity 6 (see Figure 3); the comment exchange demonstrates how the students are developing a community of learners



offer and an understanding that they were not so difficult to use. A typical entry in students' personal blogs was one student's reaction to learning how to use the feed aggregator: "As has already happened with the previous e-tivities, when we were asked to learn to use new tools, this time I had little problems at first as well. Anyway, after hesitating and being a bit 'angry,' I'm now able to use another tool, and this thanks to my English teacher and e-tivities."

Language Learning

A typical reaction of students at the beginning of the course was "What does this have to do with language learning?" An analysis of the data proves that it can have much to do with language learning, especially reading and writing. As Will Richardson (2004) states in his blog on educational blogging, "Blogging starts with reading. … blogging, at base, is writing down what you think when you read others. … The tool requires writing. The act requires reading. Without reading, you're just writing, not blogging." They developed their reading skills through searching on the Internet to complete the tasks ("I've never read so much!" was one comment) and reading each other's posts before commenting on the course blog or each other's personal blogs. Furthermore, although "writing a weblog appears in the first

instance to be a form of publishing, … as time goes by, blogging resembles more and more a conversation" (Downes, 2004, p. 24). Not only did students send comments to each other's posts in their personal blogs and on the course blogs, engaging them in asynchronous conversation, but comments were also received from strangers on all the personal blogs. Receiving comments from the outside helped students become aware of the fact that they were not only writing for their peers and teacher, but for a global audience as well. Students felt more responsible for what they were writing when they realized anybody could read what they were writing.

Blogging is also, of course, about writing. Students had to write regularly during the course in the form of blog posts and comments. The teacher provided feedback on the language used in the blog posts, and improvements in accuracy took place throughout the course as a result of the feedback. Based on a genre analysis of blogs and online publishing in general, students learned how to be concise and clear and effectively organize their blog posts into small, well-structured readable pieces of information. In their questionnaires, students also mentioned having improved their ability to use different registers (blog, peer feedback, public wiki) and to write more quickly under time pressure. Since blogging involves connecting information to the rest of the Web, students learned to create links to other pages and cite sources of information or images they found on the Web. In this process, they became aware of the importance of attribution and respecting copyright or copyleft, as is the case of Creative Commons, a series of licenses from which an author of any artistic work can choose to allow others to use their work in various ways. Finally, whereas writing on their blogs entailed using relatively informal language, writing on a public wiki where anyone could read or edit their contributions required students to use more formal language and respect sound academic practice. The wiki used, EduTech Wiki (http://edutechwiki.unige.ch/en/Main_Page), is a wiki on educational technologies hosted at the University of Geneva. It is beyond the scope of this chapter to compare the differences in student writing on the TulanePadova wiki and the public EduTech Wiki[3]. However, it is worth pointing out that by being able to compare the two experiences, students came to the overall conclusion that although more challenging, having a real audience makes them take their writing more seriously. As one second-semester student wrote:

[C]ontributing to EduTech Wiki is a good way to make students more responsible and aware of what they're writing. Here anyone can access the pages and contribute to them, while last semester this didn't happen. Writing for a public audience helps us improving our languages, our style and—as in this case—our content (in the sense that maybe we are more careful in looking for sources, links, and information, and in deciding which of them are more reliable than the other ones).

In other words, the change in audience from peers and teacher to any potential Web user influences student writing.

Finally, it is important to point out that four students have continued blogging in English since the end of the course. They have changed the names of their blogs but continue to explore the potential of Web 2.0 and participate in discussions with each other and other bloggers through comments. These students have successfully integrated what they learned in the course into their postgraduation, informal language learning process.

ASSESSMENT

In a context where students are encouraged to work in groups, collaboration must be taken into consideration in assessment. According to Johnson and Johnson (in Swan, Shen & Hiltz, 2006),

"The key to successful cooperative learning is maintaining both individual accountability, in which students are held responsible for their own learning, and positive interdependence, in which students reach their goals if and only if the other students in the learning group also reach theirs" (p. 47). In other words, not only do activities have to be developed in such a way so as to promote group work and responsibility, but also, at least in a formal academic context where students place a lot of importance on grades, collective work must be assessed. Therefore, both courses used continuous assessment and weekly feedback that considered both individual and collective work as well as a final project. Based on the work by Meijas (2006), different percentages were assigned to the various aspects to be assessed. At the start of the course, the grading scheme and rubrics were demonstrated and discussed with the students so they knew what was expected of them.

For the wiki course, the percentages were as follows:

1. *Participation* (individual grade – exchanges with American peers): 25%
2. *Participation* (individual grade – wiki contributions and editing): 25%
3. *Editing the wiki* (collective grade – accuracy): 25%
4. *Final paper/presentation* (individual grade – accuracy): 25%

As can be seen, 75% of the final grade is individual, as this is still considered of primary importance at the university. Nonetheless, the grids used to weekly assess the two participation grades take into consideration collaboration skills (e.g., whether or not the contents of their exchanges and wiki pages responded to what their peers had said or done). Both communicative fluency and linguistic accuracy were taken into consideration. In other words, attempts were made to find a balance between individual and collective grades and between assessment of fluency and accuracy.

For the blogging course, the percentages aimed at encouraging active participation in knowledge creation and sharing.

1. *Distributed research* (links posted to del.icio.us): 10%
2. *Individual analysis (reflection and content–communicative fluency)*: 40%
3. *Final paper (individual grade–accuracy)*: 25%
4. *Comments to classmates' posts (quality of peer assessment)*: 10%
5. *Editing the public wiki (collective grade–accuracy)*: 15%

Here, 20% of the final grade was based on students' abilities to share, collaborate, and do peer review (e.g., distributed research, comments); 40% on their individual abilities to effectively carry out the activities and reflect on their learning experience (e.g., blog posts); and 25% on their individual performances and 15% on group work (e.g., editing the public wiki). Accuracy was considered in only 40% of the grade, while 60% focused on fluency. This choice was based on the fact that throughout their studies, Italian students focus mainly on accuracy and very little on communicative fluency. Many students commented on how much they appreciated the opportunity to be able to just communicate in English. Furthermore, the instructor expected students to emulate the communicative techniques used in the tools they were exploring as part of the course.

In order to keep track of student participation during both courses, grids were used that included task achievement, quality (on a scale of 1 to 10 based on predefined rubrics), and any additional comments. This allowed the instructor to provide weekly qualitative feedback and mid-term feedback as well as to easily calculate the final grade.

DISCUSSION

Numerous studies on network-based language teaching (Kern & Warschauer, 2000) have begun to show the benefits of using networked technologies in the foreign language classroom to connect humans to other humans. As new technologies have developed and Web 2.0 has taken hold, the use of social software in promoting these practices is simply the next step. Never before have language learners had similar opportunities not only to access authentic language but also to participate in the creation of authentic language in real contexts. Given the fact that many of these tools are free, offer remote hosting, and are easy to use, experimenting with them becomes plausible for any language teacher. In the context of the projects discussed in this chapter, for example, the teacher developed both courses without the assistance of a technician or the use of institutional servers. Furthermore, students can develop personal collaborative learning environments (e.g., through blogging and social bookmarking) that they can continue to exploit after courses are finished. In other words, rather than being contained within a closed institutional environment, students can create their own personal learning environments (Attwell, 2007; Wilson, 2006) suited to their own specific language learning styles and needs, and transferable to informal learning.

There are, however, institutional hurdles to overcome. Up until now, many institutions have integrated social software tools into their closed Learning Management Systems (LMS) or installed them on closed university servers. In many ways, this seems to imply simply using social software tools to do what can already be done with other tools in LMSs. An LMS is a centralized environment controlled by an administrator and most often accessible to a limited community of users where the learning process takes place in times and ways that have been established by an instructor. This paradigm tends to promote traditional transmissive pedagogy where the learner has a relatively passive role and cannot take control over his or her own learning. However, today most educators recognize the need for students to become more autonomous and responsible for organizing their own learning. This replication of social software in closed environments works against the very potential of social software, which reaches its full potential when there is a large user base. As has already been stated, combining various tools not only encourages students to take responsibility for their own learning, but also, if these tools are open, to continue using them once their formal education is complete.

Another issue, which seems particularly relevant in Anglo-Saxon contexts where copying and cheating have serious consequences, is destigmatizing collaboration: "[I]t is no longer cheating to find out from or gain the advice of other people or to use information sources not already in your head" (Owen et al., 2006, p. 49). If resource sharing and peer assessment are encouraged and an emphasis is placed on developing skills to learn how to judge resources, paraphrase, and reference electronic sources, then students can gather knowledge from different sources, even their peers, and produce valid academic content. In this way, students learn to learn from one another and the larger Web 2.0 community, thereby developing their participation literacy.

CONCLUSION

This chapter has attempted to offer an introduction to the use of social software in language acquisition. First, we considered the changes that are currently taking place in education toward more collaboration, responsibility, and autonomy for students in both formal and informal learning contexts. We discussed ways in which social software can help meet these new needs. Then we described two blended learning courses carried out at the University of Padova with advanced EFL students, one involving the use of a wiki and Skype in a

telecollaboration project with Tulane University in the United States, and the other involving the use of blogs and other social software tools to help promote autonomous language learning. The initial results, which are quite positive, and a description of how assessment was carried out were presented. Finally, the challenges of using social software in formal education were discussed. In order to achieve the true benefits of using social software in education, institutions and teachers will have to shift from a focus on individual work within the boundaries of a classroom or course to a focus on collective and collaborative work beyond institutional confines.

REFERENCES

Attwell, G. (2007). Personal learning environments—The future of eLearning?. *eLearning Papers*, *2*(1), 1–8. Retrieved April 25, 2007, from http://www.elearningeuropa.info/files/media/media11561.pdf

Belz, J. (2005). Telecollaborative language study: A personal overview of praxis and research. *Proceedings of the 2004 NFLRC Symposium: Distance Education, Distributed Learning, and Language Instruction*. Retrieved October 13, 2006, from http://www.nflrc.hawaii.edu/networks/nw44/belz.htm

Berg, E.C. (1999). The effects of trained peer response on ESL students: Revision types and writing quality. *Journal of Second Language Writing, 8*(3), 215–241.

Bonaiuti, G. (Ed.). (2007). *E-learning 2.0: Il futuro dell'apprendimento in rete, tra formale e informale*. I quaderni di Form@re n. 6. Retrieved May 10, 2007, from http://www.erickson.it/erickson/repository/pdf/PRODUCT_1205_PDF.pdf

Brown, J.S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher, 18*(1), 32–42.

Bruns, A., & Humphreys, S. (2005). Wikis in teaching *and assessment—The M/Cyclopedia project. Proceedings of at the 2005 International Symposium on Wikis*, San Diego, California. Retrieved July 15, 2006, from http://snurb.info/files/Wikis%20in%20Teaching%20and%20Assessment.pdf

Bryant, T. (2006). Social software in academia. *Educause Quarterly, 29*(2), 61–64. Retrieved September 9, 2006, from http://www.educause.edu/apps/eq/eqm06/eqm0627.asp

Cross, J. (2006). *Informal learning: Rediscovering the natural pathways that inspire innovation and performance.* San Francisco, CA: Wiley.

Dewey, J. (1965). *Democrazia ed educazione*. Firenze, Italia: La Nuova Italia.

Downes, S. (2004). Educational blogging. *Educause Review, 39*(5), 14–26. Retrieved August 22, 2006, from http://www.educause.edu/apps/er/erm04/erm0450.asp

Ferdig, R.E., & Trammell, K.D. (2004). Content delivery in the "blogoshpere." *THE Journal*, February 2004. Retrieved September 19, 2006, from http://www.thejournal.com/articles/16626

Furstenberg, G., Levet, S., English, K., & Maillet, K. (2001). Giving a virtual voice to the silent language of culture: The CULTURA project. *Language Learning & Technology, 5*(1), 55–102. Retrieved May 11, 2007, from http://llt.msu.edu/vol5num1/furstenberg/default.html

Giger, P. (2006). *Participation literacy* [licentiate thesis]. Karlskrona, Sweden: Blekinge Institute of Technology. Retrieved from http://www.participationliteracy.com/

Godwin-Jones, R. (2004). Blogs and wikis: Environments for on-line collaboration. *Language Learning & Technology, 7*(2), 12–16. Retrieved September 24, 2006, from https://llt.msu.edu/vol7num2/emerging/

Godwin-Jones, R. (2006). Tag clouds in the blogosphere: Electronic literacy and social networking. *Language Learning & Technology*, *10*(2), 8–15. Retrieved September 24, 2006, from https://llt.msu.edu/vol10num2/emerging/

Jonassen, D. (2000). *Learning as activity.* Proceedings of the Meaning of Learning Project, Learning Development Institute, Presidential Session at AECT, Denver, Colorado.

Kern, R., & Warschauer, M. (2000). *Network-based language teaching: Concepts and practice*. New York: Cambridge University Press.

Lamb, B. (2004). Wide open spaces: Ready or not. *Educause Review*, *39*(5), 36–48. Retrieved September 15, 2006, from http://www.educause.edu/ir/library/pdf/ERM0452.pdf

Lund, A., & Smordal, O. (2006). Is there a space for the teacher in a wiki? *Proceedings of the 2006 International Symposium on Wikis*, Odense, Denmark. Retrieved September 9, 2006, from http://www.wikisym.org/ws2006/proceedings/p37.pdf

Martin, M. (2007). *My personal learning environment.* Blog post from The Bamboo Project: educate, advocate, innovate, collaborate on April 11, 2007. Retrieved April 26, 2007, from http://michelemartin.typepad.com/thebambooproject-blog/2007/04/my_personal_lea.html

Mayer, R. (Ed.). (2005). *The Cambridge handbook of multimedia learning.* New York: Cambridge University Press.

Mejias, U. (2006). Teaching social software with social software. *Innovate*, *2*(5). Retrieved September 19, 2006, from http://www.innovateonline.info/index.php?view=article&id=260

Oblinger, D., & Oblinger, J. (2005). *Educating the Net Generation*. Educause e-book. Retrieved on October, 4, 2007, from http://www.educause.edu/educatingthenetgen

O'Dowd, R. (2005). Negotiating sociocultural and institutional contexts: The case of Spanish-American telecollaboration. *Language and Intercultural Communication*, *5*(1), 40–56.

O'Dowd, R. (2007). Evaluating the outcomes of online intercultural exchange. *ELT Journal*, *61*(2), 144–152.

O'Reilly, T. (2005). What is Web 2.0: *Design patterns and business models for the next generation of software*. O'Reilly.com Web site. Retrieved January 10, 2007, from http://oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1

Owen, M., Grant, L., Sayers, S., & Facer, K. (2006). *Social software and learning.* Futurelab Web site. Retrieved April 25, 2007, from http://www.futurelab.org.uk/research/opening_education/social_software_01.htm

Prensky, M. (2001). Digital natives, digital immigrants. *On the Horizon*, *9*(5), 1–6. Retrieved April 4, 2007, from http://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf

Richardson, W. (2004). Reading and Blogging. Weblogg-ed Web site. Retrieved October 10, 2006, from http://weblogg-ed.com/2004/03/31

Salmon G. (2002). *E-tivities: The key to active online learning.* London and Sterling, VA: Kogan Page Limited.

Sfard, A. (1998). On two metaphors for learning and the dangers of choosing just one. *Educational Researcher*, *27*(2), 4–13.

Spiro, R.J., Feltovich, P.J., Jacobson, M.J., & Coulson, R.L. (1991). Cognitive flexibility, constructivism and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains. *Educational Technology*, *31*(5), 24–33.

Surowiecki, J. (2005). *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies and nations.* Doubleday.

Swan, K., Shen, J., & Hiltz, S.R. (2006). Assessment and collaboration in online learning. *Journal of Asynchronous Learning Networks*, *10*(1), 45–62. Retrieved May 10, 2006, from http://www.sloan-c.org/publications/JALN/v10n1/v10n1_5swan.asp

Wilson, S. (2006). PLE Workshop, Cetis PLE Event in Manchester (UK). Retrieved April 24, 2007, from http://www.cetis.ac.uk/members/scott/blogview?entry=20060609211909

KEY TERMS

Blog: Simply defined, a blog, or weblog is a sort of online journal organized in reverse chronological order, where a person writes about his or her thoughts and interests, including providing links to relevant resources on the Web. Most blogs allow readers to leave comments. There are many types of blogs, from very personal journals to educational blogs. Various types of media, from audio to video to images, can often be integrated into a text blog.

Feed Aggregators: A client software that allows users to receive syndicated Web content from any type of Web site that uses feeds such as newspaper Web sites, blogs, podcasts, and so forth. In other words, rather than having to regularly check Web sites for updated information through the use of feeds (RSS, XML RSD, XML Atom), updated information is sent to the feed aggregator so users have only one place to check for updated content. Users can decide how much of the updated information they would like to receive in the aggregator (e.g., a few lines or the entire text) and whether or not to receive just text or other media as well. Users can also go directly to the Web sites from the aggregator. Feed aggregators provide a useful tool for managing the information overload on the Internet.

Informal Learning: Learning that takes place outside of institutionally defined contexts; for example, learning on the job and in one's personal life. It can be associated with other concepts such as lifelong and continuous learning, both of which are becoming more important in today's information society.

Network-Based Language Teaching (NBLT): NBLT involves teaching languages through the use of computers that are connected to one another in either local or global networks. NBLT allows students to interact with speakers of the target language without having to physically meet with them. In NBLT, the main focus is on authentic communication via the computer and the Internet. Social software offers new opportunities for NBLT.

Participation Literacy: Whereas information literacy refers to a person's ability to effectively manage information, participation refers to the competences required to effectively participate in Web 2.0 environments. In other words, participation literacy involves learning the social skills needed to take part in online communities.

Social Bookmarking: Social bookmarking Web sites allow users to store, classify, share, and search their own Internet bookmarks as well as those of other community members by using tags (folksonomies). Most services offer remote hosting so that users can access their bookmarks from any computer. Social bookmarking can serve as a filter for the information overload on the Internet. When users search on these Web sites, they are not searching the entire Web using an algorithm, as is the case on most search engines, but rather they are viewing Web sites that other users have found to be useful and taken the time to save and describe, and for which they have chosen semantically classified tags.

Social Software: A generic term used to describe various types of software that enable people to collaborate and create, and join online communities. The tools can promote various types of communication: synchronous one-to-one (instant messaging), synchronous one-to-many (Skypecasts), asynchronous one-to-many (blogs), asynchronous many-to-many (wikis), or asynchronous many-to-one (feed aggregators). These tools allow users to share and create content, collaboratively create and edit content and/or manage content.

Web 2.0: A term used to contrast the World Wide Web in the 1990s as a collection of Web sites produced by experts, institutions, and companies (the read-only Web) with the changes that took place starting with the 21st century where Web applications allow end users to create and share content on the Web (the read-write Web).

Wiki: A software application that allows the creation and development of interlinked Web pages; any user can create new pages and edit existing pages. Wikis, therefore, are an effective tool for collaborative authoring, collective learning, and project-based work.

ENDNOTES

1 Futurelab is a not-for-profit organization researching innovation in education, incubating new ideas, communicating thinking and practice, etc. and can be found at http://www.futurelab.org.uk/

2 There was no informal written communication in the second semester so the first aim only applied to the first semester.

3 See Guth, S. (2007). Wikis in Education: Is Public Better? Paper presented at WikiSym'07 October 21–23, 2007, Montréal, Québec, Canada. Retrieved 06 November 2007 from http://ws2007.wikisym.org/space/GuthPaper.

This work was previously published in Handbook of Research on E-Learning Methodologies for Language Acquisition, edited by R. de Cássia Veiga Marriott and P. Lupion Torres, pp. 424-442, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.3
Activity-Oriented Computing

João Pedro Sousa
George Mason University, USA

Bradley Schmerl
Carnegie Mellon University, USA

Peter Steenkiste
Carnegie Mellon University, USA

David Garlan
Carnegie Mellon University, USA

ABSTRACT

This chapter introduces a new way of thinking about software systems for supporting the activities of end-users. In this approach, models of user activities are promoted to first class entities, and software systems are assembled and configured dynamically based on activity models. This constitutes a fundamental change of perspective over traditional applications; activities take the main stage and may be long-lived, whereas the agents that carry them out are plentiful and interchangeable. The core of the chapter describes a closed-loop control design that enables activity-oriented systems to become self-aware and self-configurable, and to adapt to dynamic changes both in the requirements of user activities and in the environment resources. The chapter discusses how that design addresses challenges such as user mobility, resolving conflicts in accessing scarce resources, and robustness in the broad sense of responding adequately to user expectations, even in unpredictable situations, such as random failures, erroneous user input, and continuously changing resources. The chapter further summarizes challenges and ongoing work related to managing activities where humans and automated agents collaborate, human-computer interactions for managing activities, and privacy and security aspects.

INTRODUCTION

Over the past few years, considerable effort has been put into developing networking and middle-

ware infrastructures for ubiquitous computing, as well as in novel human-computer interfaces based on speech, vision, and gesture. These efforts tackle ubiquitous computing from two different perspectives—systems research and HCI research—hoping to converge and result in software that can support a rich variety of successful ubiquitous computing applications. However, although examples of successful applications exist, a good understanding of frameworks for designing ubiquitous computing applications is still largely missing.

A key reason for the lack of a broadly applicable framework is that many research efforts are based on an obsolete application model. This model assumes that ubiquitous computing applications can support user activities by packaging, at design time, a set of related functionalities within a specific domain, such as spatial navigation, finding information on the Web, or online chatting. However, user **activities** may require much diverse functionality, often spanning different domains. Which functionalities are required to support an activity can only be determined at runtime, depending on the user needs, and may need to evolve in response to changes in those needs. Examples of user activities targeted by ubiquitous computing are navigating spaces such as museums, assisting debilitated people in their daily living, activities at the office such as producing reports, as well as activities in the home such as watching a TV show, answering the doorbell, or enhancing house security.

This chapter introduces activity-oriented computing (AoC) as a basis for developing more comprehensive and dynamic applications for ubiquitous computing. Activity-oriented computing brings user activities to the foreground by promoting models of such activities to first class primitives in computing systems.

In the remainder of this chapter, the section on background presents our vision for activity-oriented computing and compares it with related work. Next we discuss the main challenges of this approach to ubiquitous computing. Specifically, we discuss user mobility (as opposed to mobile computing), conflict resolution and robustness, mixed-initiative control, human-computer interaction, and security and privacy.

The main body of the chapter presents our work towards a solution. Specifically, we discuss software architectures for activity-oriented computing and how to address the challenges of mobility and robustness, as well as the options to model user activities. The chapter ends with a discussion of future directions concerning human-computer interactions, and the tradeoff between ubiquity and security and privacy.

BACKGROUND

The vision of AoC is to make the computing environment aware of *user **activities*** so that resources can be autonomously managed to optimally assist the user. Activities are everyday actions that users wish to accomplish and that may be assisted in various ways by computing resources in the environment. Done right, AoC will allow users to focus on pursuing their activities rather than on configuring and managing the computing environment. For example, an AoC system could reduce overhead by automatically customizing the environment each time the user wishes to resume a previously interrupted long-lived activity, such as preparing a monthly report, or organizing a party.

To help make this vision concrete, the following examples illustrate possible applications of AoC:

- **Elderly care:** Rather than relying on hardcoded solutions, AoC enables domain experts such as doctors and nurses to "write prescriptions" for the activities of monitoring the health of the elderly or outpatients. Such descriptions enable smart homes to take charge of those activities, collaborating

with humans as appropriate. For example, the heart rate of an elderly person may be monitored by a smart home, which takes responsibility to alert family members when alarming measurements are detected. Who gets alerted and the media to convey the alert may depend on contextual rules, such as the seriousness of the situation, as prescribed by the doctor; the elder's preferences of who to contact, who is available, who is closer to the elder's home, whether sending an SMS appropriate, and so forth.

- **Entertainment:** While others have explored the vision that music, radio, or television can follow occupants as they move through the house, activity-oriented computing enables a more general approach. Entertainment can be defined as an activity, allowing preferences and constraints to be specified, and underlying **services** to be shared, such as tracking people, identifying and using devices in various rooms. For example, the same location services used for home security activities can be used for entertainment; the television that can be used for entertainment can also be used for displaying images of a visitor at the front door.
- **Home Security:** Many homes have a security system that uses sensors to detect burglary attempts and fires. They are standalone systems with limited capabilities; for example the system is typically either on or off and control is entirely based on a secret code. If the security system were built as an activity service, it could be an open system with richer functionality. For example:
    - A richer set of control options, for example, based on fingerprint readers or voice recognition. These methods may be more appropriate for children or the elderly.
    - More flexibility (e.g., giving neighbors limited access to water the plants when the homeowners are on vacation, the ability to control and interact with the system remotely, or incorporate cameras that ignore dogs).
    - Remote diagnosis, for example, in response to an alarm, police or fire responders may be able to quickly check for false alarms through cameras.
- **Doorbell:** A very simple activity is responding to somebody ringing the doorbell. Today's solution is broadcast; the doorbell is loud enough to alert everybody in the house and then people decide independently or after coordination (through shouting!) how to respond. In activity-oriented computing, a doorbell activity carried out by the hallway selects a person, based on their current location, current activity, and age. If the visitor can be identified, it might be possible to have the person who is being visited respond. Also, the method of alerting the person can be customized, for example, using a (local) sound, displaying a message on the television screen, or flashing the lights. Finally, if nobody is home, the doorbell service can take a voice message or, if needed, establish a voice or video link to a house occupant who might be available in their office or car. Activities such as answering the phone could be handled in a similar way, that is replace the broadcast ringing by a targeted, context-aware alert.

What is Activity-Oriented Computing

Activity-oriented computing adopts a fundamental change of perspective over traditional applications in that **activities** take the main stage and may be long-lived, whereas the agents that carry them out are plentiful and interchangeable; how activities are best supported will evolve over time, depending on the user's needs and context. In AoC, activities are explicitly represented and manipulated by the computing infrastructure.

Broadly speaking, this has two significant advantages.

First, it enables explicit reasoning about user **activities**: which activities a user may want to carry out in a particular context; what functionality (**services**) is required to support an activity; what are the user preferences relative to quality of service for each different activity; which activities conflict; which have specific privacy or security concerns, and so forth.

Second, it enables reasoning about the optimal way of supporting activities, through the dynamic selection of service suppliers (agents) that implement specific functions relevant to the activity. Thanks to the explicit modeling of the requirements of activities and of the capabilities of agents, the optimality of such assignment may be addressed by quantitative frameworks such as **utility** theory. Also, by raising the level of abstraction above particular applications or implementations, activity models make it easier to target a broad range of concrete implementations of similar services in different devices, in contrast to solutions based on mobile code (more in the Challenges section, below).

Related Work

Early work in ubiquitous computing focused on making certain applications ubiquitously available. For that, it explored OS-level support that included location sensing components to automatically transfer user interfaces to the nearest display. Examples of this are the work on teleporting X Windows desktops (Richardson, Bennet, Mapp, & Hopper, 1994) and Microsoft's Easy Living project (Brumitt, Meyers, Krumm, Kern, & Shafer, 2000). This idea was coupled with the idea of desktop management to treat users' tasks as sets of applications independent of a particular device. Examples of systems that exploit this idea are the Kimura project (MacIntyre et al., 2001), which migrates collections of applications across displays within a smart room, and earlier work in aura that targets migration of user tasks across machines at different locations (Wang & Garlan, 2000). Internet Suspend-Resume (ISR) requires minimal changes to the operating system to migrate the entire virtual memory of one machine to another machine (Kozuch & Satyanarayanan, 2002). These approaches focus on making applications available ubiquitously, but do not have a notion of user activity that encompasses user needs and preferences, and therefore do not scale to environments with heterogeneous machines and varying levels of service.

More recent work seeks to support cooperative tasks in office-like domains, for example ICrafter (Ponnekanti, Lee, Fox, & Hanrahan, 2001) and Gaia (Román et al., 2002); as well as domain-specific tasks, such as healthcare (Christensen & Bardram, 2002) and biology experiments, for example, Labscape (Arnstein, Sigurdsson, & Franza, 2001). This research shares with ours the goal of supporting activities for mobile users, where activities may involve several services in the environment, and environments may contain heterogeneous devices. However, much of this work is predicated on rebuilding, or significantly extending, operating systems and applications to work over custom-built infrastructures. In contrast the work described in this chapter supports user activities with a new software layer on top of existing operating systems, and accommodates integration of legacy applications.

Focusing on being able to suspend and resume existing activities in a ubiquitous environment does not go all the way toward the vision of providing ubiquitous assistance for user activities. Such support can be divided into two categories: 1) helping to guide users in conducting tasks; and 2) performing tasks, or parts of tasks, on behalf of users. An early example of the first category is the Adtranz system (Siewiorek, 1998), which guides technical staff through diagnosing problems in a train system. More recent work concentrates on daily life, often for people with special needs, such as the elderly, or those with debilitated health

(Abowd, Bobick, Essa, Mynatt, & Rogers, 2002; Intille, 2002).

Research on automated agents takes assistance one step further by enabling systems to carry out activities on behalf of users. Examples of this are the RETSINA framework (Sycara, Paolucci, Velsen, & Giampapa, 2003), with applications in domains such as financial portfolio management, e-commerce and military logistics; and more recently the RADAR project (Garlan & Schmerl, 2007), which focuses on the office domain, automating such tasks as processing e-mail, scheduling meetings, and updating Web sites.

Consumer solutions for activities in the home are beginning to emerge, mainly from the increasing complexity of configuring home theater equipment. Universal remote controls, such as those provided by Logitech, allow users to define activities such as "Watch DVD," which choose the input source for the television, output of sound through the home theater system, and choosing the configuration of the DVD player (Logitech). However, in these solutions, activities are bound to particular device and device configurations—the activities themselves must be redefined for different equipment, and it is not possible for the activities to move around different rooms in the home, or to allow different levels of service for the same activity.

In this chapter, we discuss the potential and the challenges of having software systems using activity models at runtime. Specifically, we focus on the benefits of using activity models for enabling users to access their activities ubiquitously, and for delegating responsibility for activities to automated agents.

CHALLENGES

Activity-oriented computing raises a number of challenges that must be addressed by any adequate supporting infrastructure and architecture.

- **User mobility:** As users move from one environment to another—for example, between rooms in a house—activities may need to migrate with the users, tracking their location and adapting themselves to the local situation. A key distinction between user mobility in AoC and previous approaches is that no assumptions are made with respect to the users having to carry mobile devices, or to the availability of a particular kind of platform at every location. Since different environments may have very different resources (devices, services, interfaces, etc.) a critical issue is how best to retarget an activity to a new situation. For example, an activity that involves "watching a TV show" can be changed into "listening" when the user walks through a room that only offers audio support. Solving this problem requires the ability to take advantage of context information (location, resource availability, user state, etc.), as well as knowledge of the activity requirements (which services are required, fidelity tradeoffs, etc.) to provide an optimal use of the environment in support of the activity.
- **Conflict resolution:** Complicating the problem of automated configuration and reconfiguration is the need to support multiple activities—both for a single user and between multiple users. If an individual wants to carry out two activities concurrently that may need to use shared resources, how should these activities simultaneously be supported? For example, if the user is engaged in entertainment, should the doorbell activity interrupt that activity? Similar problems exist when two or more people share an environment. For example, if two users enter the living room hoping to be entertained, but having different ideas of what kind of entertainment they want, how can those conflicts be reconciled? Solving this problem requires (a) the ability to detect

when there may be conflicts, and (b) the ability to apply conflict resolution policies, which may itself require user interaction.

- **Mixed-initiative control:** The ability to accomplish certain kinds of activities requires the active participation of users. For example, the door answering activity, which might be associated with a house, requires occupants of the house to respond to requests from the house to greet a visitor. Since humans exhibit considerably more autonomy and unpredictability than computational elements, it is not clear how one should write the activity control policies and mechanisms to allow for this. Standard solutions to human-based activities (such as work-flow management systems) are likely not to be effective, since they assume users will adhere to predetermined plans to a much higher degree than is typically the case in the kinds of domains targeted by ubiquitous computing.
- **Security and privacy:** Some security and privacy issues can be solved through traditional mechanisms for security, but others are complicated by key features of ubiquity; they include rich context information, and user mobility across public or shared spaces such as a car, or an airport lounge. In a multi-user environment with rich sources of context information (such as a person's location), an important issue is how to permit appropriate access to and sharing of that information. Furthermore, which guarantees can be made to a user that wishes to access personal activities in a shared space? What mechanisms can back such guarantees? Are there deeper issues than the exposure of the information that is accessed in a public space? Is it possible that all of a user's information and identity may be compromised as a consequence of a seemingly innocuous access at a public space?
- **Human-computer interaction:** Many of the activities that a ubiquitous computing environment should support will take place outside of standard computing environment (such as a networked office environment). In such environments, one cannot assume that users will have access to standard displays, keyboards, and pointing devices. How then can the system communicate and interact with users effectively? What should be the role of emerging technologies such as augmented reality and natural interaction modalities such as speech, gesture, and ambient properties such as light and smell?

While the challenges above stem from the problem domain, we now turn to the challenges associated with building a solution for AoC.

- **Activity models**. The first challenge is to define what kinds of knowledge should be imparted in systems to make them aware of user activities. Specifically, what should be the contents and form of activity models? What should be the semantic primitives to compose and decompose activities? At what level of sophistication should activity models be captured? Presumably, the more sophisticated the models, the more a system can do to assist users. For example, to help users with repairing an airplane or with planning a conference, a significant amount of domain knowledge needs to be captured. But obviously, capturing such knowledge demands more from users (or domain experts) than capturing simple models of activities. Is there an optimal level of sophistication to capture activity models—a sweet spot that maximizes the ratio between the benefits of imparting knowledge to systems and the costs of eliciting such knowledge from users? Or is it possible to have flexible solutions that allow incremental levels of sophistication for representing each activity, depending

on the expected benefits and on the user's willingness to train the system?

- **System design**. Systems that support AoC should be capable of dynamic reconfiguration in response to changes in the needs of user activities. Ideally, such systems would also be aware of the availability of resources in the environment and respond to changes in those. The questions then become: What is an appropriate architecture to support activity-oriented computing? What responsibilities should be taken by a common infrastructure (middleware) and which should be activity- or service-specific? What are the relevant parameters to guide service discovery (location, quality of service, etc.) and how should discovery be geographically scoped and coordinated? Can activity models be capitalized to handle the heterogeneity of the environment, self-awareness and dynamic adaptation? Furthermore, what operations might be used to manage activities: suspend and resume, delegate, collaborate, others? What should be the operational semantics of each of these operations?
- **Robustness**. In AoC, robustness is taken in the broad sense of responding adequately to user expectations, even in unpredictable situations, such as random failures, erroneous user input, and continuously changing resources. First of all, should adequacy be a Boolean variable—either the system is adequate or it is not—or can it be quantified and measured? Specifically, are there system capabilities and configurations that are more adequate than others to support a user's activity? If so, can measures of adequacy be used to choose among alternatives in rich environments? For example, is the user better served by carrying out a video conference on a PDA over a wireless link, or on the wall display down the hall?

TOWARDS A SOLUTION

To address the challenges identified above, we decided to start with relatively simple models of activities and address concrete problems where the advantages of AoC could be demonstrated. This section summarizes our experience of about six years at Carnegie Mellon University's project **aura**. Initially, this research targeted the smart office domain, and later extended to the smart home domain (more below).

Designing systems for AoC brings up some hard questions. What makes those questions especially challenging, is that to answer them, we need to reexamine a significant number of assumptions that have been made about software for decades. Not surprisingly, our own understanding of how to answer those questions continues to evolve. This section is organized around the set of solution-related challenges identified above; namely, system design, activity models, and robustness.

System Design

The first research problem we focused on, starting around the year 2000, was user mobility in the smart office domain. Here, activities (or tasks) typically involve several applications and information assets. For instance, for preparing a presentation, a user may edit slides, refer to a couple of papers on the topic, check previous related presentations, and browse the web for new developments. An example of user mobility is that the user may start working on the presentation at his or her office, continue at the office of a collaborator, and pick the task up later at home.

The premise adopted for user mobility is that users should not have to carry a machine around, just as people normally don't carry their own chairs everywhere. If they so desire, users should be able to resume their tasks, on demand, with whatever computing systems are available. This premise is neither incompatible with users carrying mobile devices, nor with mobile code. Ideally, the capa-

bilities of any devices or code that travel with the user contribute to the richness of the environment surrounding the user, and therefore contribute to a better user experience. A discussion of why solutions centered on mobile devices, mobile code, or remote computing (such as PC Anywhere) are not entirely satisfactory to address user mobility can be found in (J.P. Sousa, 2005).

Designing a solution to support user mobility is made harder by the heterogeneity of devices where users may want to resume their activities, and by dynamic variations in the resources and devices available to the user. Even in a fairly restricted office domain, it is common to find different operating systems, offering different suites of applications (e.g., Linux vs. PC vs. Mac). In a broader context, users may want to carry over their activities to devices with a wide range of capabilities, from handhelds to smart rooms. In addition to heterogeneity, mobile devices are subject to wide variations of resources, such as battery and bandwidth. Ideally, software would automatically manage alternative computing strategies based on user requirements and on the availability of resources. Moreover, in heavily networked environments, remote servers may constantly change their response times and even availability. Ideally, users should be shielded as much as possible from dealing with such dynamic variations.

Before describing an architecture for supporting user mobility as outlined above, Table 1 clarifies the terminology used throughout this chapter, since although the terms are in common use, their interpretation is far from universal.

Our starting point for supporting user mobility was to design an infrastructure, the **aura** infrastructure, that exploits knowledge about a user's tasks to automatically configure and reconfigure the environment on behalf of the user. Aura is best explained by a layered view of its infrastructure together with an explanation of the roles of each layer with respect to task suspend-resume and dynamic **adaptation**.

First, the infrastructure needs to know *what* to configure for; that is, what a user needs from

Table 1. Terminology

task	An everyday activity such as preparing a presentation or writing a report. Carrying out a task may require obtaining several *services* from an *environment*, as well as accessing several *materials*.
environment	The set of *suppliers*, *materials* and *resources* accessible to a user at a particular location.
service	Either (a) a service type, such as printing, or (b) the occurrence of a service proper, such as printing a given document. For simplicity, we will let these meanings be inferred from context.
supplier	An application or device offering *services* – for example, a printer.
material	An information asset such as a file or data stream.
capabilities	The set of *services* offered by a *supplier*, or by a whole *environment*.
resources	Are consumed by *suppliers* while providing *services*. Examples are: CPU cycles, memory, battery, bandwidth, and so forth.
context	Set of human-perceived attributes such as physical location, physical activity (sitting, walking…), or social activity (alone, giving a talk…).
user-perceived state of a task	User-observable set of properties in the *environment* that characterize the support for the task. Specifically, the set of *services* supporting the task, the user-level settings (preferences, options) associated with each of those services, the *materials* being worked on, user-interaction parameters (window size, cursors…), and the user's preferences with respect to quality of service tradeoffs.

the environment in order to carry out his or her tasks. Second, the infrastructure needs to know *how* to best configure the environment; it needs mechanisms to optimally match the user's needs to the capabilities and resources in the environment.

In our architecture, each of these two sub-problems is addressed by a distinct software layer: (1) the *Task Management* layer determines *what* the user needs from the environment at a specific time and location; and (2) the *Environment Management* layer determines *how* to best configure the environment to support the user's needs.

Table 2 summarizes the roles of the software layers in the infrastructure. The top layer, *task management* (TM), captures knowledge about user needs and preferences for each activity. Such knowledge is used to coordinate the configuration of the environment upon changes in the user's task or context. For instance, when the user attempts to carry out a task in a new environment, TM coordinates access to all the information related to the user's task, and negotiates task support with environment management (EM). Task Management also monitors explicit indications from the user and events in the physical context surrounding the user. Upon getting indication that the user intends to suspend the current task or resume some other task, TM coordinates saving the user-perceived state of the suspended task and recovers the state of the resumed task, as appropriate.

The *EM layer* maintains abstract models of the environment. These models provide a level of indirection between the user's needs, expressed in environment-independent terms, and the concrete capabilities of each environment.

This indirection is used to address both heterogeneity and dynamic change in the environments. With respect to heterogeneity, when the user needs a service, such as speech recognition, EM will find and configure a supplier for that service among those available in the environment. With respect to dynamic change, the existence of explicit models of the capabilities in the environment enables automatic reasoning when those capabilities change dynamically. The Environment Management adjusts such a mapping automatically in response to changes in the user's needs (adaptation initiated by TM), and changes in the environment's capabilities and resources (adaptation initiated by EM). In both cases adaptation is guided by the maximization of a ***utility*** *function* representing the user's preferences (more in the section on Robustness, below).

Table 2. Summary of the software layers in aura

layer	mission	roles
Task Management	***what*** does the user need	monitor tasks, contexts and preferences
		map tasks to needs for services in an environment
		for complex tasks: decomposition, plans, context dependencies
Environment Management	***how*** to best configure the environment	monitor environment capabilities and resources
		map service needs and user-level state of tasks to available suppliers
		ongoing optimization of the utility of the environment relative to user task
Env.	support the user tasks	monitor relevant resources
		fine grain management of QoS/resource tradeoffs

The *environment layer* comprises the applications and devices that can be configured to support a user's task. Configuration issues aside, these suppliers interact with the user exactly as they would without the presence of the infrastructure. The infrastructure steps in only to automatically configure those suppliers on behalf of the user. The specific capabilities of each supplier are manipulated by EM, which acts as a translator for the environment-independent descriptions of user needs issued by TM. Typically, suppliers are developed by wrapping existing applications. Our experience in wrapping over a dozen applications native to Windows and Linux has shown that it is relatively easy to support setting and retrieving the user-perceived state (Balan, Sousa, & Satyanarayanan, 2003; J.P. Sousa, 2005).

This layering offers a clean separation of concerns between what pertains to user activities and what pertains to the environment. The knowledge about user activities is held by the TM and travels with the user to each environment in which he or she wishes to carry out activities. The knowledge about the environment stays with the EM and can be used to address the needs of many users.

A significant distinction of this approach to user mobility is that it does not require code or devices to travel with the user. A generic piece of code, Prism, in the TM layer *becomes* an **aura** for a user by loading models of user activities. Those models are encoded in XML for convenience of mobility across heterogeneous devices (more in the section on Activity Models, below).

Extending to the Home Environment

Although the layered perspective played an important role in clarifying the separation of concerns and protocols of interaction, it captures only the case where users consume services, and software components provide them.

In the smart home domain, software could take responsibility for activities, and users might be asked to contribute services for those activities. For example, a smart home might take charge of the home's security and ask a human to lock the windows when night falls. Other examples of activities include a user watching a TV show, a user checking on a remote family member, the main hallway facilitating answering the door, and the home keeping a comfortable temperature. These examples prompted us to realize that any domain entity could have an aura, and that an aura might find itself on either the supplying or the consuming side, or both. Specifically, **auras** can be associated with:

- People including individual residents, or groups, such as a resident's parents, or the entire family.
- Spaces, such as the main hallway, living room or the entire home. Spaces of interest are not necessarily disjoint.
- Appliances, such as a TV, phone, table or couch. Appliances have a well-defined purpose and may have a range of automation levels, from fairly sophisticated (a smart refrigerator), to not automated at all (an old couch).
- Software applications, such as a media player, a video conferencing app, or a people locator. Applications run on general purpose devices, and which applications are available on one such device define the purpose that the device may serve.

Figure 1 shows the run-time architectural framework for an activity-oriented system. The boxes correspond to types of components, and the lines to possible connectors between instances of those types.[1] Part (a) shows our initial understanding based on the smart office domain, and part (b) shows the more general framework. Contrasting the two, it is now clear that the TM corresponded to an aura (of users) that consumed services but supplied none; and suppliers corresponded to auras of software that supplied services, but consumed none.

Figure 1. Architectural framework

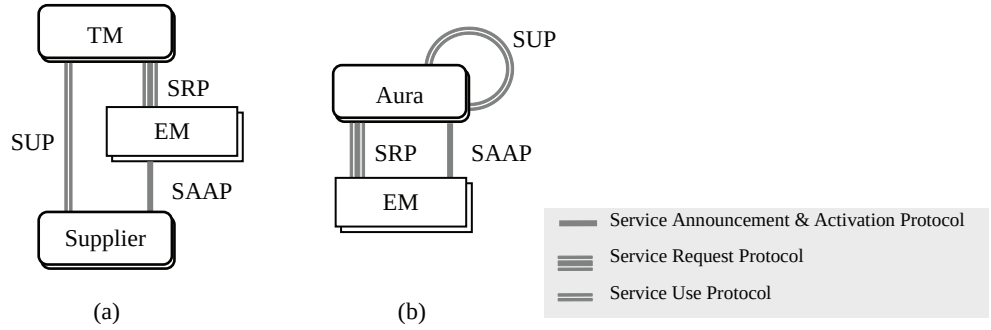
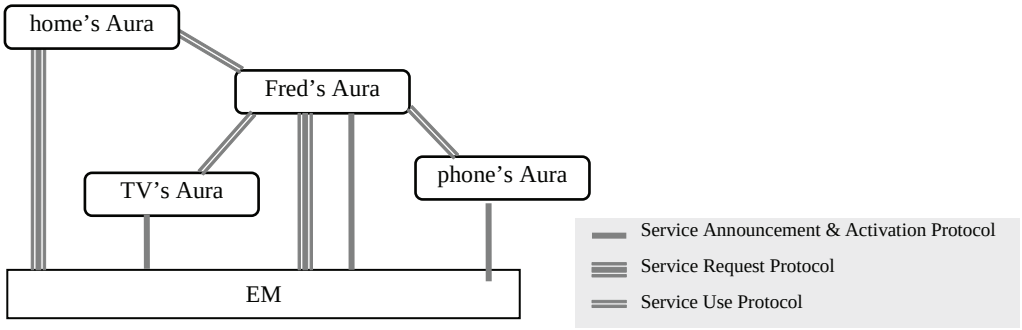


Figure 2. Snapshot of the architecture of one system



In the new architectural framework, when an aura boots up, it first locates the EM (see the section on Service Discovery, below) and then may engage on the service announcement & activation protocol (SAAP) to announce any services that its entity provides, as well as on the service request protocol (SRP) to discover services that are relevant to support the entity's activities. Once the services in other auras are actually recruited by the EM, using the SAAP, the consumer aura and the supplier auras interact via the service use protocol (SUP) to reconstruct the user-perceived state of the activity.

Figure 2 shows an example of an architecture that was dynamically created to respond to the needs of a user, Fred, at a particular place, Fred's home. The boxes correspond to run time components (autonomous processes that may be deployed in different devices) rather than denoting code packaging, and the lines correspond to connectors, that is, actual communication channels that are established dynamically as the components are created. The diagram represents two kinds of components: auras, with rounded corners, and the EM, and it visually identifies the different kinds of interaction protocols between the components (see Figure 1).

The instance of the architecture in the example is the result of Fred's aura interacting with the EM to recruit two suppliers: the TV and the phone's auras, after interpreting Fred's needs for the desired activity. This architecture may evolve to adapt to changes in Fred's needs, and in the figure, Fred's aura is also shown as being recruited by the home's aura to get Fred to open the front door.

Context Awareness

An important decision is how to enable context awareness in activity-oriented computing. Addressing context awareness can be decomposed into three parts—sensing context, distributing contextual information, and reacting to context. We start by discussing the latter.

Potentially, all domain entities, and therefore their auras, might want to react to context. A user's aura may change the requirements on an activity, or change which activities are being carried out depending on context such as user location, or whether the user is sitting alone at the office, driving a car, or having an important conversation with his or her boss. Suppliers contributing services to an activity may want to change modalities of those services based on context. For example, an application that shows confidential information may hide that information automatically if someone else is detected entering the room; or an application that requires text or command input may switch to speech recognition if the user needs to use his or her hands, say, for driving a vehicle. The EM may change the allocation of suppliers for a given activity based on user location. For example, if the user is watching a TV show while moving around the house, different devices may be allocated: the TV in the living room, the computer monitor at the home office, etc. The upshot of this is that contextual information should be accessible to all the boxes in the architecture.

Initially, we thought that a dedicated part in the infrastructure would be in charge of gathering and distributing contextual information; there would be a Context Management component/layer in each environment, just like there is an Environment Management. However, both the contextual information and the policies for distributing such information are associated with each user and not really with the environment where that user happens to be. Therefore, auras are the hub of knowledge about the entities they represent.

Whenever a component wishes to obtain contextual information about entity X, it will ask X's aura. X's aura itself may use a variety of mechanisms to gather information about X. For example, since physical location of a space is normally a non-varying property, the aura for a home can read the home's location from a configuration file. In contrast, the aura for an application running on a cell phone equipped with GPS might obtain the application's location from the device's GPS. The aura for a person P typically obtains P's contextual information via contextual information services (CIS).

Unlike sensors specific to devices or spaces, CIS's are fairly generic. Specifically, devices such as a thermometer attached to a wall, or a window sensor for detecting whether *that* window is open or closed, are accessed only by the auras of the corresponding physical spaces. In contrast, given a training set with a person's face, a generic face recognizer may be able to track that person's location inside an office building by using cameras spread over rooms and halls.

CISs are integrated into the architecture using the same protocols for discovery and activation as other services, which allows for gracefully handling of activity instantiation in both sensor-rich environments, such as a smart building, and in depleted environments, such as a park.

While CIS components release information based on generic premises such as the rule of law (e.g., only an authenticated aura for X or a law enforcement agent with a warrant can obtain information about X), the auras themselves are responsible for knowing and enforcing their entity's privacy policies regulating whom to release which information. As an example of distribution policy, a user may authorize only a restricted group of friends to obtain his or her current location.

Service Discovery

In the initial architectural framework (Figure 1.a), **service** discovery is coordinated by the environ-

ment management layer. When we expanded the focus of our research to the smart home domain, around the year 2004, we revisited the problem of service discovery. Among the questions that prompted this revisiting are: are there real advantages in brokering discovery? What are the relevant parameters to guide service discovery (location, quality of service, etc.)? How can discovery be geographically scoped? Would some measure of physical distance be enough for such scoping? Can discovery be scoped by geographical areas that are meaningful to the user? We elaborate on these below.

This first question is what should be the strategy for discovery. Many activities in the smart home domain involve entities performing services for other entities, and it is up to auras to find and configure the services required by their entities. For example, for watching a TV show, Fred will need some entity to play the video stream for him. Fred's aura takes care of finding and configuring such an entity in Fred's vicinity (for instance the TV in the living room,) and to change the video stream to other convenient entities whenever Fred moves around the home (a TV in the kitchen, a computer in the office, etc.).

One candidate solution would be to have auras broadcast service availability and/or service requests. However, service discovery in ubiquitous computing involves not just matching service names or capabilities but, ideally, it would find *optimal* services in terms of attributes such as desired levels of quality of service, user preferences, proximity, and so forth. Furthermore, scoping the search would be constrained by network administration policies regarding broadcast (see below). Also, it is hard to establish trust based on broadcast mechanisms.

Because finding the optimal entities to perform services is both a non-trivial problem and common across auras, there are clear engineering advantages in factoring the solution to this problem out of individual auras and into a dedicated piece of the infrastructure. Specifically, the benefits of introducing Environment Managers (EMs) as discovery brokers include:

- **Separation of concerns:** It is up to specialized service brokers to know *how* to find the optimal entities to provide services, while each aura retains the responsibility of knowing *what* services are required by their entity's activities at each moment. By providing a separate locus for optimal discovery in EMs, auras can focus on the task-specific knowledge required to interact with other auras, once they are identified.
- **Efficiency:** EMs can cashe service announcements, thereby improving the latency of processing a service request, and reducing the network traffic required to locate a service, whenever one is requested.

auras register the services offered by the entity they represent with an EM. Each service posting includes the type of service and all attributes of the offering entity that characterize the service. For example, Fred's aura announces that Fred is capable of *answering the door* (service type,) along with Fred's contextual attributes pertaining to his current *location* and whether he is *busy*. Although Fred's aura might know about Fred's blood pressure, that wouldn't be directly relevant for his ability to answer the door. As another example, the aura for a printer announces its ability to *print documents*, along with the printer's *location*, *pages per minute* and *queue length* (contextual and quality of service attributes).

auras may request an EM to find services, as needed by the activities of the entities they represent. Service discovery is guided by optimality criteria in the form of **utility** functions over the attributes of the service suppliers and of the requesting entity. Specifically a service request is of the form:

```
find x : service | y • max u(p_x, p_y)
```

This means: find a set of entities $\underline{x}$, each capable of supplying a `service`, given the requestor entity `y`, such that a utility function `u` over the properties of `y` and of the elements of $\underline{x}$ is maximized. The following are examples with simple utility functions:

Track Fred's location. Upon startup, Fred's aura issues:

```
find x1:people-locating | Fred
```

That is, find x_1 capable of providing a people locating service for Fred.

Follow-me video. When Fred wishes to watch a soccer game while moving around the house, his aura issues:

```
find x1:video-playing | Fred · min x1.location – Fred.location
```

That is, find a video player closest to Fred. In this case, maximizing the utility corresponds to minimizing the physical distance between Fred and the video player.

Doorbell. When the doorbell is pressed by someone, the aura for the main hallway issues:

```
find x1:door-answering, x2:notifying | hallway
· x1.busy = no & min (x1.location – hallway.location
+ x1.location – x2.location)
```

That is, find a notifying mechanism and a door answerer that is not busy and both closest to the hallway and to the notifying mechanism.

Utility functions are quantitative representations of usefulness with respect to each property. Formally, selecting a specific value of a property, such as x_1.busy = no, is encoded as a discrete mapping, specifically:

$$u_{x_1 \cdot busy}(yes) \mapsto 0,\ u_{x_1 \cdot busy}(no) \mapsto 1$$

For properties characterized by numeric values, such as the distance to the hallway, we use utility functions that distinguish two intervals: one where the user considers the quantity to be good enough for his activity, the other where the user considers the quantity to be insufficient. *Sigmoid* functions, which look like smooth step functions, characterize such intervals and provide a smooth interpolation between the limits of those intervals. Sigmoids are easily encoded by just two points; the values corresponding to the knees of the curve that define the limits *good* of the good-enough interval, and *bad* of the inadequate interval. The case of "more-is-better" qualities (e.g., *accuracy*) are as easily captured as "less-is-better" qualities (e.g., *latency*) by flipping the order of the *good* and *bad* values (see (Sousa, Poladian, Garlan, Schmerl, & Shaw, 2006) for the formal underpinnings).

In the case studies evaluated so far, we have found this level of expressiveness for utility functions to be sufficient.

Scoping Service Discovery

The second question is how service discovery can be scoped in a way that is meaningful to the user. Specifically, many searches take place in the user's immediate vicinity, such as the user's home.

However, neither physical distance nor network range are good parameters to scope discovery. For example, if the user's activity asks for a device to display a video, the TV set in the apartment next door should probably not be considered, even though it might be just as close as other candidates within the user's apartment, and be within range of the user's wireless network as well. To be clear, once a set of devices is scoped for discovery, physical distance may be factored in as a parameter for *optimality* (see above).

Furthermore, sometimes users may want to scope discovery across areas that are not contigu-

ous. For example, suppose that Fred is at a coffee shop and wants to print an interesting document he just found while browsing the internet. Fred may be willing to have the document printed either at the coffee shop, or at Fred's office, since Fred is heading there shortly. A printer at a store down the street may not be something that Fred would consider, even though it is physically closer than Fred's office.

The question about scoping service discovery can then be refined into (a) if not by distance or network boundaries, how can the range of one environment be defined? and (b) how to coordinate discovery across non-contiguous environments?

When an aura directs a discovery request to an EM, by default, discovery runs across all services registered with that EM. That is, the range of an environment is defined by the services that registered with its EM. The question then becomes, how does an aura know with which EM it should register its services? For example, how would the aura for the TV set in Fred's living room know to register its services with the EM in Fred's apartment, and not with the neighbor's?

Auras resolve their physical location into the network address of the appropriate EM by using the environment manager binding protocol (EMBP).[2] This service plays a similar role to the domain naming service in the internet, which resolves URI/URLs into the network address of the corresponding internet server. Physical locations are encoded as aura location identifiers (ALIs), with structure and intent similar to universal resource identifiers (URIs) in the internet. Like URIs, ALIs are a hierarchical representation meant to be interpreted by humans and resolved automatically. For example, *ali://pittsburgh.pa.us/zip-15000/main-street/1234/apt-6* might correspond to Fred's apartment; and *ali://aura.cmu.edu/wean-hall/floor-8/8100-corridor* to a particular corridor on the 8th floor of Wean Hall at Carnegie Mellon University.

Requests for discovery across remote and/or multiple environments can be directed to the local EM, which then coordinates discovery with other relevant EMs (more below). The following are examples of such requests. When Fred is at home and wishes to print a document at the office, his aura would issue a request like:

```
find x:printing | Fred • u(…)
    @ ali://aura.cmu.edu/wean-hall/floor-8/8100-
corridor
```

Or, if Fred wanted to consider alternatives either at home or at his office:

```
find x:printing | Fred • u(…)
    @ ali://aura.cmu.edu/wean-hall/floor-8/8100-
corridor,
    ali://pittsburgh.pa.us/zip-15000/main-
street/1234/apt-6
```

Or, if Fred wanted to search a number of adjacent environments, such as all the environments in his office building:

```
find x:printing | Fred • u(…)
    @ ali://aura.cmu.edu/wean-hall
```

Any such requests are directed by the requestor aura to the local EM, which then resolves such requests in three steps:

1. Use the EMBP to identify the EMs that cover the desired region.
2. Obtain from such EMs all the service descriptions that match the requested service types.
3. Run the service selection algorithms over the candidate set of services.

Activity Models

What to include in activity models is ultimately determined by the purpose that those models are

meant to serve. In some applications of activity models the goal is to assist users with learning or with performing complex tasks. Examples of these are applications to automated tutoring, expert systems to help engineers repair complex mechanisms, such as trains and airplanes, and automated assistants to help manage complex activities such as organizing a conference (Garlan & Schmerl, 2007; Siewiorek, 1998). For these kinds of applications, models of activities may include a specification of workflow, as a sequence of steps to be performed, and cognitive models of the user.

In the smart office domain, we experimented with enabling users to suspend their ongoing activities and resume them at a later time and/or at another location, possibly using a disjoint set of devices. For that purpose, the models capture user needs and preferences to carry out each activity. Specifically, such models include of a snapshot of the services and materials being used during the activity, as well as utility theory-based models of user preferences (for details on the latter, see (Sousa et al., 2006)).

Figure 3 shows a grammar for modeling **activities**, or tasks, as a set of possibly interconnected services. This grammar follows a variant of the Backus-Naur Form (BNF, see for instance (ISO, 1996)). To simplify reading the specification, we drop the convention of surrounding non-terminal symbols with angle brackets, and since the task models are built on top of XML syntax, we augment the operators of BNF with the following:

```
E ::= t: A; C
```

defines a type E of XML elements with tag t, attributes A, and children C, where t is a terminal symbol, A is an expression containing only terminals (the attribute names), and C is an expression containing only non-terminals (the child XML elements). In this restricted use of BNF, whether a symbol is a terminal or non-terminal is entirely established by context. So, for instance the rule:

Figure 3. Grammar for specifying task models

```
Task  ::= auraTask: id;
      Prefs {ServiceSnapshot | MaterialSnapshot | Config}

ServiceSnapshot ::= service: id type;
      Settings
MaterialSnapshot ::= material: id;
      State

Config ::= configuration: name weight;
      { Service | Connection }

Service ::= service: id;
      {Uses}
Uses  ::= uses: materialId;

Connection ::= connection; id type;
      Attach QoSPrefs
Attach ::= attach: ;
      From To
From  ::= from: serviceId port;
To    ::= to: serviceId port;
```

```
Book = book: year ISBN; Title {Author}
```

allows the following as a valid element:

```
<book year="2004" ISBN="123">
    <title>...</title>
    <author>...</author>
    <author>...</author>
</book>
```

Specifically, in Figure 3, a task (model) is an XML element with tag `auraTask`, with one `id` attribute, and with one `Prefs` child, followed by an arbitrary number of `ServiceSnapshot`, `MaterialSnapshot`, and `Config` children. A task may be carried out using one of several alternative service configurations of services.

Services stand for concepts such as *edit text*, or *browse the web*, and materials are files and data streams manipulated by the services. A service may manipulate zero or many materials; for instance, text editing can be carried out on an arbitrary number of files simultaneously. That relationship is captured by the `Uses` clauses within the `Service` element.

The snapshot of the user-perceived state of the task is captured in the *`Settings`* and *`State`* elements. The *`Settings`* element captures the state that is specific to a service, and shared by all materials manipulated by that service, while the *`State`* element captures the state that is specific to each material. A detailed discussion of this grammar can be found in (Sousa, 2005).

Figure 4 shows one example of a task model for reviewing a video clip, which formally is a sentence allowed, or generated, by the grammar in Figure 3. This example was captured while running the infrastructure described in the section on System Design. The user defined two alternative configurations for this task: one including both playing the video and taking notes, the other, playing the video alone. Both services use a single material: *play video* uses a video file, with material id 11, and *edit text* uses a text file, with material id 21. The user-perceived state of the task is represented as the current service settings, under each service, and the current state of each material. For instance, the state of the video includes the fact that the video is stopped at the beginning (the cursor is set to 0 time elapsed), and it indicates the position and dimensions of the window showing the video.

Extending to the Home Environment

In the smart home domain, in addition to supporting suspend/resume of activities, we wanted to enable users to delegate responsibility for some activities to **auras**. Examples of the latter activities include managing intrusion detection for the home, finding a person to answer the door for a visitor, or assisting with monitoring elder family members.

The research questions then become: is the services and materials view of activities adequate in the smart home domain? For enabling auras to take responsibility for activities, which concepts should activity models capture?

The usefulness of capturing the services needed for an activity seems to carry over well into the smart home domain. For example, in the case of the doorbell scenario, the activity of answering the door requires finding services such as notification can be supplied by devices such as a telephone, a TV, a buzzer, etc., and *door answerer*, which can be supplied by a qualified person (e.g., not a toddler). Selecting the suppliers for such services is guided by the home owner's preferences encoded in the activity model, which may include things such as the door needs to be answered within a certain time, and that the notification service should be in close proximity to the candidate door answerer.

A prototype of this case study has shown that these models can handle sophisticated policies of configuration (e.g., excluding children from answering the door, or specifying criteria for proximity) and that they trivially accommodate the dynamic addition of new notification devices.

Figure 4. Example task model for reviewing a video clip

```
<auraTask id="34">
  <preferences>
    <service template="default" id="1"/>
    <service template="default" id="2"/>
  </preferences>
  <service type="play Video" id="1">
    <settings mute="true"/>
  </service>
  <material id="11">
    <state>
      <video state="stopped" cursor="0"/>
      <position xpos="645" ypos="441"/>
      <dimension height="684" width="838"/>
    </state>
  </material>
  <service type="edit Text" id="2">
    <settings>
      <format overtype="0"/>
      <language checkLanguage="1"/>
    </settings>
  </service>
  <material id="21">
    <state>
      <cursor position="31510"/>
      <scroll horizontal="0" vertical="7"/>
      <zoom value="140"/>
      <spellchecking enabled="1" language="1033"/>
      <window height="500" xpos="20" width="600" mode="min" ypos="100"/>
    </state>
  </material>
  <configuration name="all" weight="1.0">
    <service id="2">
      <uses materialId="21"/>
    </service>
    <service id="1">
      <uses materialId="11"/>
    </service>
  </configuration>
  <configuration name="only video" weight="0.7">
    <service id="1">
      <uses materialId="11"/>
    </service>
  </configuration>
</auraTask>
```

This prototype also highlighted two fundamental differences between the kinds of activities supported in the smart-office domain and the ones we target in the smart-home domain. The first difference is that, while in the office domain services were only provided by automated agents (software), now people may also be asked to provide services. This has implications on how auras control service supply, since people are much more likely than software to do something totally different than what they are being asked. In the example, after being notified to answer the door, a person may get sidetracked and forget about it. It is up to the responsible aura to monitor whether or not the service is being delivered, and react to a "fault" in a similar way as it would in the case of

faulty software: by seeking a replacement (more in the section on Robustness).

The second difference is that, in the smart home domain, auras may take the responsibility for activities; this is related to the question above of which concepts to capture in activity models to enable that to happen. In the smart office domain, when a fault cannot be handled, for example, if a suitable replacement cannot be found for a faulty supplier, the problem is passed up to the entity responsible for the activity, that is, the user. If an aura is to be truly responsible for an activity, it must be take charge of such situations as well.

One way of addressing a hurdle in one activity, is to carry out another activity that circumvents the hurdle. In the example, if the hallway aura cannot find a person to answer the door, it may take a message from the visitor, or initiate a phone call to the person being visited.

A simple enhancement of activity models to allow this is to support the specification of conditions to automatically resume or suspend activities. Such conditions are expressed as Boolean formulas over observation of contextual information. For example, if everyone left the house, resume the intrusion detection activity.

For these models to cover situations as the one where a person could not be found to answer the door, contextual information needs to be rich enough to include semantic observations, such as "the door could not be open for a visitor."

Another scenario where we tested this approach is the elder care scenario. The aura for Susan, Fred's grandmother, runs a perpetual task that recruits a heart monitor service for her. Susan defined under which conditions her aura should trigger the task of alerting the family. When defining such conditions, Susan takes into consideration her physician's recommendations, but also conditions under which she may desire privacy. Fred's aura runs a perpetual task of monitoring contextual postings by Susan's aura. It is up to John to (a) define that posting such a notification should trigger the task of alerting him, and (b) define the means employed by his aura to carry out such a task. For example, if Fred is at the office, his aura sends an instant message to Fred's computer screen; otherwise, it sends a text message to Fred's cell phone.

While these are simple scenarios, they illustrate the ability to chain activities, and to direct the exact behavior of activities, by capturing conditions on contextual information in the models of activities. Such conditions are associated to the operations of either resuming or suspending activities, and can be monitored by auras to automatically initiate the corresponding operation.

Formally, condition-action primitives can be used to express the same space of solutions than other more sophisticated approaches, such as models of activities based on workflow notations, or on hierarchical decomposition of activities. Which approach would be more suitable for end-users to express and understand such models is an open research problem.

ROBUSTNESS

The term *robustness* in activity-oriented computing is interpreted very broadly: is the system's behavior consistent with the users' expectations, even under unanticipated circumstances. In this section, we first use the examples in the Background section to identify key robustness requirements. We then look at the challenges associated with supporting robust operation, distinguishing between general challenges and challenges that are specific to the home environment. Finally, we summarize some results showing how we support robust tasks in an office environment and discuss how these results can be extended to support activities in the home.

Properties

In daily use, the system should correctly identify the users' intent and should support a wide vari-

ety of activities in a way that is consistent with their preferences and policies. If users observe unexpected behavior, the system should be able to explain its behavior. This will increase the users' confidence in the system and will allow the system to improve over time. For example, by adjusting preferences and policies, either manually by the user or automatically by the system (case-base reasoning) the system's future behavior can be made to better match user intent. Similarly, the system should be able to engage users if input is confusing or unexpected. Ideally, the system would be able to recognize undesirable or unsafe actions, for example, a child opening the door for a stranger.

The above properties must also be maintained as the system evolves and under failure conditions. For example, when new services or devices are added (e.g., camera and face recognition software is added to support the doorbell scenario) or become unavailable (e.g., the license for the face recognition software expired), the system should automatically adapt to the available services.

Challenges

When we looked at how to support user activities and tasks in different environments (e.g., work in an office, daily activities in the home, or guiding visitors in a museum), we found that several key challenges are shared across these environments. These generic challenges include capturing and representing user intent, discovering and managing services and devices (suppliers), and optimizing resources allocation to maximize overall system utility. All these functions should be adaptive, that is, automatically adapt to changes in the computational and physical environment and to changes in the goals and preferences of users.

Each environment also adds its own challenges. For example, activities in homes are device-centric (e.g., displays, sound) or include physical actions that involve people (e.g., opening doors). Managing and allocating such "resources" is very different from an office environment, where tasks are computer-centric and are supported by executing applications that use a variable amount of resources (network bandwidth, CPU, battery power). Similarly, the interactions with users are very different in the home (discreet interface for non-experts) and the office (keyboard/mouse/display used by computer knowledgeable users).

Robustness in an Office Environment

In order to achieve robustness in a smart-office environment, we have designed, implemented and evaluated an infrastructure that uses utility theory to dynamically select the best achievable configuration of services, even in the face of failures and coming online of better alternatives (Sousa et al., 2006).

Robustness is achieved through self-**adaptation** in response to events ranging from faults, to positive changes in the environment, to changes in the user's task. Self-adaptation is realized through a closed-loop control system design that senses, actuates, and controls the runtime state of the environment based on input from the user. Each layer reacts to changes in user tasks and in the environment at a different granularity and time-scale. Task Management acts at a human perceived time-scale (minutes), evaluating the adequacy of sets of services to support the user's task. Environment Management acts at a time-scale of seconds, evaluating the adequacy of the mapping between the requested services and specific suppliers. Adaptive applications (fidelity-aware and context-aware) choose appropriate computation tactics at a time-scale of milliseconds.

Let us illustrate the behavior of the system using the following scenario. Fred is engaged in a conversation that requires real-time speech-to-speech translation. For that task, assume the aura infrastructure has assembled three services: speech recognition, language translation, and speech synthesis. Initially both speech recognition

and synthesis are running on Fred's handheld. To save resources on Fred's handheld, and since language translation is computationally intensive, but has very low demand on data-flow (the text representation of each utterance), the translation service is configured to run on a remote server. We now discuss how the system adapts in response to faults, variability in resource and service availability, and changes in the user's task requirements:

- **Fault tolerance**. Suppose now that there is loss of connectivity to the remote server, or equivalently, that there is a software crash that renders it unavailable. Live monitoring at the EM level detects that the supplier for language translation is lost. The EM looks for an alternative supplier for that service, for example, translation software on Fred's handheld, activates it, and automatically reconfigures the service assembly.
- **Resource and fidelity-awareness**. Computational resources in Fred's handheld are allocated by the EM among the services supporting Fred's task. For computing optimal resource allocation, the EM uses each supplier's spec sheet (relating fidelity levels with resource consumption), live monitoring of the available resources, and the user's preferences with respect to fidelity levels. Resource allocation is adjusted over time. For example, suppose that during the social part of the conversation, Fred is fine with a less accurate translation, but response times should be snappy. The speech recognizer, as the main driver of the overall response time, gets proportionally more resources and uses faster, if less accurate, recognition algorithms (Balan et al., 2003).
- **Adaption.** Adaptation is also needed to deal with changes in resource availability. Each supplier issues periodic reports on the Quality of Service (QoS) actually being provided—in this example, response time and estimated accuracy of recognition/translation. Suppose that at some point during the conversation, Fred brings up his calendar to check his availability for a meeting. The suppliers for the speech-to-speech translation task, already stretched for resources, reduce their QoS below what Fred's preferences state as acceptable. The EM detects this "soft fault," and replaces the speech recognizer by a lightweight component, that although unable to provide as high a QoS as the full-fledged version, performs better under sub-optimal resource availability. Alternatively, suppose that at some point, the language translation supplier running on the remote server (which failed earlier) becomes available again. The EM detects the availability of a new candidate to supply a service required by Fred's task, and compares the estimated utility of the candidate solution against the current one. If there is a clear benefit, the EM automatically reconfigures the service assembly. In calculating the benefit, the EM factors in a cost of change. This mechanism introduces hysteresis in the reconfiguration behavior, thus avoiding oscillation between closely competing solutions.
- **Task requirements change**. Suppose that at some point Fred's conversation enters a technical core for which translation accuracy becomes more important than fast response times. The TM provides the mechanisms to allow Fred to quickly indicate his new preferences; for instance, by choosing among a set of preference templates. The new preferences are distributed by the TM to the EM and all the suppliers supporting Fred's task. Given a new set of constraints, the EM evaluates the current solution against other candidates, reconfigures, if necessary, and determines the new optimal resource allocation. The suppliers that remain in the configuration, upon receiving the new preferences, change

their computation strategies dynamically, for example, by changing to algorithms that offer better accuracy at the expense of response time.

Suppose that after the conversation, Fred wants to resume writing one of his research papers. Again, the TM provides the mechanisms to detect, or for Fred to quickly indicate, his change of task. Once the TM is aware that the conversation is over it coordinates with the suppliers for capturing the user-level state of the current task, if any, and with the EM to deactivate (and release the resources for) the current suppliers. The TM then analyses the description it saved the last time Fred worked on writing the paper, recognizes which services Fred was using and requests those from the EM. After the EM identifies the optimal supplier assignment, the TM interacts with those suppliers to automatically recover the user-level state where Fred left off. See Sousa and Garlan (2003) for a formal specification of such interactions.

Extending to the Home Environment

We are currently enhancing this solution to provide robust support for activities in the home. While the key challenges are the same (e.g., optimizing utility, adapting to changes, etc.), extensions are needed in a number of areas.

First, activities in the home are very different from tasks in the office. For example, since some activities in the home involve physical actions, people must be involved (e.g., to open a door), that is, people become suppliers of services. Moreover, some tasks are not associated with individuals, but with the home itself (e.g., responding to the doorbell or a phone call). This change in roles means that it is even more critical to make appropriate allocations since the cost of mistakes is much higher; for example, people will be much less willing to overlook being personally inconvenienced by a wrong decision, than when a suboptimal application is invoked on their computer.

Second, many activities in the home involve the use of devices that are shared by many people, or involve deciding who should perform a certain action. This means that the Task Manager will typically need to balance the preferences and goals of multiple users. An extreme example is conflicts, for example, when multiple users would like to use the same device. In contrast, tasks in the office typically involve only personal resources (e.g., a handheld) or resources with simple sharing rules (e.g., a server).

Third, the methods for interaction with the system will be much different in the home. Even on a handheld, Fred had access to pull down menus and a keyboard to reliably communicate with the system. For the home environment, we are exploring natural modalities of interaction, which are less intrusive, but more ambiguous (more in the section on Future Research).

Finally, uncertainty will play a more significant role in the home, for example, because of unpredictable behavior when people are asked to perform services, or due to ambiguity caused by primitive I/O devices. Work in progress is extending the utility optimization components to explicitly consider uncertainty.

FUTURE RESEARCH

Some of the challenges identified in this chapter are the topic of undergoing and future work, such as research on the kinds of knowledge to capture in activity models so to support mixed-initiative control, including delegation and collaboration among human and automated agents. Below we summarize our current work on human-computer interaction and on security and privacy for AoC systems.

User Interfaces for Managing Activities

Human-computer interaction in the office domain currently uses one de-facto standard modality,

based on keyboards, pointing devices, and windows-based displays. In a more general ubiquitous computing setting, natural modalities such as speech and gesture may be highly desirable, but they also may lead to ambiguity and misunderstanding. For example, if Fred points at a TV where a soccer game is playing and leaves the room, does that mean that Fred wants to keep watching the game while moving around the house, that the TV should pause the game until Fred returns, or that the TV should be turned off?

Rather than trying to pick a privileged modality of interaction, we take the approach that interactions between humans and auras may have many channels that complement and serve as alternatives to each other. For example, users might indicate their intention to suspend an activity verbally, but might sometimes prefer a graphical interface to express a sophisticated set of contextual conditions for when an activity should be automatically resumed. The research questions then become: what are appropriate modalities for each kind of interaction? Is there a role for explicit interactions, as well as for implicit interactions based on sensing and inference? Can different modalities be coordinated, contributing to disambiguate user intentions? What mechanisms can be used to detect and recover from misunderstandings? What are specific technologies that can be harnessed in the home?

To support explicit interactions, we started exploring technologies such as Everywhere Displays and RFID. The Everywhere Displays technology uses a digital camera to track down the location of a user, and then uses a projector to project an image of the interface onto a surface near the user (Kjeldsen, Levas, & Pinhanez, 2004). The feedback loop through the camera allows the image to be adjusted for certain characteristics of the surface, such as color and tilt. The user interacts with this image by performing hand motions over the image, which are then recognized via the camera. This technology supports a metaphor similar to the point-and-click metaphor, although fewer icons seem to be feasible relative to a computer screen, and a rich set of command primitives, such as double clicking or selecting a group of objects, seems harder to achieve.

RFID technology supports a simple form of tangible interfaces (Greenberg & Boyle, 2002). For example, RFID tags can be used to create tangible widgets for activities. In the example where Fred is watching the game on TV, Fred may bind an activity widget with the show playing on the TV by swiping the widget near the TV. That activity may be activated in other rooms by swiping the activity widget by a reader in the room, or deactivated it by swapping the widget again, once activated (see demo video at (Sousa, Poladian, & Schmerl, 2005)).

Tradeoff between Ubiquity and Security

The big question to be answered is: can ubiquity be reconciled with goals of security and privacy? There seems to be a tradeoff between the openness of ubiquitous computing and security assurances. The very meaning of *ubiquity* implies that users should be enabled to use the services offered by devices embedded in many different places. But how confident can users be that those devices, or the environment where they run, will not take advantage of the access to the user's information to initiate malicious actions?

Rather that taking an absolute view of security and privacy, we argue that there are different requirements for different activities. For example, the computing environment at a coffee shop could be deemed unsafe to carry out online financial transactions, but acceptable for sharing online vacation photos with a friend.

In essence, this is a problem of controlling access; ideally, a ubiquitous computing environment would gain access only to the information pertaining to the activities that a user is willing to carry out in that environment, and none other.

Unfortunately, existing solutions for controlling access are not a good fit to this problem because they make a direct association between identity and access. Specifically, once a user authenticates, he gains access to *all* the information and resources he is entitled to, and so does the computing environment where the user authenticated.

A candidate solution would be to associate access control to the cross-product of users and environments; in the example, user Fred at the coffee shop would get access to a limited set of activities, but user Fred at his office would get access to a wide range (possibly all) of Fred's activities. A serious problem with this solution is that it would require the pre-encoding of all the types of environments where the user might want to access his or her activities.

Another candidate solution would be for users to have multiple identities; Fred at the coffee shop would use an identity that has access to the vacation photos, but not to online banking. This solution has two obvious problems; first, separating the activities and associated information for the different identities may not be clear cut, and may quickly become cumbersome for moderately high numbers of activities. Second, if users are given the freedom to define new identities and the corresponding access controls, does that mean that every user should be given security administration privileges?

We are currently investigating an access control model centered on the notion of *persona*. A user is given one identity and may define multiple personae associated with that identity. The user may freely associate activities with personae in a many-to-many fashion, and may also define which credentials are required to activate each persona. This model has a number of benefits, as follows.

First, it allows users to manage which activities are seen by an arbitrary environment (by authenticating specific personae) while drawing a clear boundary on the administrative privileges of each user.

Second, users may draw on rich forms of authentication to make the overhead of authentication proportionate to the security requirements. For example, for activating Fred's financial persona, Fred may require two forms of id to be presented, such as entering a password *and* scanning an id card, while for his social persona, a weak form of authentication, such as face or voice recognition, will suffice.

Third, the model offers users a coherent view of the personal workspace centered on their identity, while enabling users to expand the set of accessible activities at will, by providing the credentials required to activate the desired personae.

CONCLUSION

The key idea of Activity-oriented Computing (AoC) is to capture models of user activities and have systems interpret those models at run time. By becoming aware of what user activities entail, systems can do a better job at supporting those activities, either by facilitating access to the activities while relieving users from overhead such as configuring devices and software, or by taking responsibility for parts or whole activities.

This chapter described the authors' work on building systems to support AoC. It discussed how those systems may address challenges inherent to the problem domain, such as user mobility and conflict resolution, as well as challenges that are entailed by building the systems themselves. Specifically, (a) defining what to capture in activity models (b) designing systems that do a good job at supporting user activities while addressing the challenges in the problem domain, and (c) making those systems robust, self-aware, self-configurable, and self-adaptable. The chapter dissected those challenges, identified specific research questions, and described how the authors

answered these questions for the past six years, as their understanding of the issues improved.

The main contributions of this work are as follows:

- Pragmatic models of user activities that enable mobile users to instantiate activities in different environments, taking advantage of diverse local capabilities without requiring the use of mobile devices, and retaining the ability to reconstitute the user-perceived state of those activities
- Mechanisms that enable scoping service discovery over geographical boundaries that are meaningful to users, and which can be specific to each activity and be freely defined
- A utility-theoretic framework for service discovery that enables optimization of sophisticated, service-specific models of QoS and context properties
- A robustness framework, based on the same utility-theoretic framework, that departs from the traditional binary notion of fault and uniformly treats as an optimization problem faults, "soft faults" (unresponsiveness to QoS requirements,) and conflicts in accessing scarce resources
- Closed-loop control that enables systems to become self-aware and self-configurable, and to adapt to dynamic changes in both user/activity requirements and environment resources
- A software architecture that harmoniously integrates all the features above, additionally (a) integrating context sensing according to the capabilities of each environment, and (b) coordinating adaptation policies with applications that may contain their own fine-grain mechanisms for adaptation to resource variations

This chapter also summarized ongoing and future work towards addressing other challenges, namely, supporting activities where human and automated agents collaborate (mixed-initiative activities), exploring human-computer interaction modalities for AoC in ubiquitous computing environments, and investigating models for security and privacy.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 0615305

REFERENCES

Abowd, G., Bobick, A., Essa, I., Mynatt, E., & Rogers, W. (2002). *The Aware Home: Developing Technologies for Successful Aging.* Paper presented at the AAAI Workshop on Automation as a Care Giver, Alberta, Canada.

Arnstein, L., Sigurdsson, S., & Franza, R. (2001). Ubiquitous Computing in the Biology Laboratory. *Journal of Lab Automation (JALA), 6*(1), 66-70.

Balan, R. K., Sousa, J. P., & Satyanarayanan, M. (2003). *Meeting the Software Engineering Challenges of Adaptive Mobile Applications* (Tech. Rep. No. CMU-CS-03-111). Pittsburgh, PA: Carnegie Mellon University.

Brumitt, B., Meyers, B., Krumm, J., Kern, A., & Shafer, S. (2000). EasyLiving: Technologies for intelligent environments. In Gellersen, Thomas (Eds) *Proceedings of the 2nd Int'l Symposium on Handheld and Ubiquitous Computing* (HUC2000), LNCS 1927, (pp. 12-29), Bristol, UK: Springer-Verlag.

Christensen, H., & Bardram, J. (2002, September). Supporting Human Activities – Exploring Activity-Centered Computing. In Borriello and Holmquist (Eds.) *Proceedings of the 4th International Conference on Ubiquitous Computing*

(UbiComp 2002), LNCS 2498, (pp. 107-116), Göteborg, Sweden: Springer-Verlag.

Garlan, D., & Schmerl, B. (2007). The RADAR Architecture for Personal Cognitive Assistance. *International Journal of Software Engineering and Knowledge Engineering, 17*(2), in press.

Greenberg, S., & Boyle, M. (2002). Customizable physical interfaces for interacting with conventional applications. In Proceedings of the *15th Annual ACM Symposium on User Interface Software and Technology* (UIST 2002) (pp. 31-40). ACM Press.

Intille, S. (2002). Designing a home of the future. *IEEE Pervasive Computing, 1*(2), 76-82.

ISO. (1996). *Extended Backus-Naur Form* (No. ISO/IEC 14977:1996(E)). Retrieved on from http:// www.iso.org: International Standards Organization

Kjeldsen, R., Levas, A., & Pinhanez, C. (2004). Dynamically Reconfigurable Vision-Based User Interfaces. *Journal of Machine Vision and Applications, 16*(1), 6-12.

Kozuch, M., & Satyanarayanan, M. (2002). *Internet Suspend/Resume*. Paper presented at the 4th IEEE Workshop on Mobile Computing Systems and Applications, available as Intel Research Report IRP-TR-02-01.

Logitech, I. Logitech Harmony Remote Controls. Retrieved on from http://www.logitech.com

MacIntyre, B., Mynatt, E., Voida, S., Hansen, K., Tullio, J., & Corso, G. (2001). Support for Multitasking and Background Awareness Using Interactive Peripheral Displays. In *ACM User Interface Software and Technology* (UIST'01), pp. 41-50, Orlando, FL.

Ponnekanti, S., Lee, B., Fox, A., & Hanrahan, P. (2001). ICrafter: A Service Framework for Ubiquitous Computing Environments. In Abowd, Brumitt, Shafer (Eds) *3rd Int'l Conference on Ubiquitous Computing* (UbiComp 2001), LNCS 2201, pp. 56-75. Atlanta, GA: Springer-Verlag.

Richardson, T., Bennet, F., Mapp, G., & Hopper, A. (1994). A ubiquitous, personalized computing environment for all: Teleporting in an X Windows System Environment. *IEEE Personal Communications Magazine, 1*(3), 6-12.

Rochester, U. The Smart Medical Home at the University of Rochester. Retrieved on from http:// www.futurehealth.rochester.edu/smart_home

Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., & Narhstedt, K. (2002). Gaia: A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing, 1*(4), 74-83.

Siewiorek, D. (1998). Adtranz: A Mobile Computing System for Maintenance and Collaboration. In *Proceedings of the 2nd IEEE Int'l Symposium on Wearable Computers* (pp. 25-32). IEEE Computer Society.

Sousa, J. P. (2005). *Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments* (Tech. Rep. No. CMU-CS-05-123). Pittsburgh, PA: Carnegie Mellon University.

Sousa, J. P., & Garlan, D. (2003). *The aura Software Architecture: an Infrastructure for Ubiquitous Computing* (Tech. Rep. No. CMU-CS-03-183). Pittsburgh, PA: Carnegie Mellon University.

Sousa, J. P., Poladian, V., Garlan, D., Schmerl, B., & Shaw, M. (2006). Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Special Issue on Engineering Autonomic Systems, 36*(3), 328-340.

Sousa, J. P., Poladian, V., & Schmerl, B. (2005). Project aura demo video of the follow me scenario. Retrieved from http://www.cs.cmu.edu/~jpsousa/research/aura/followme.wmv

Sycara, K., Paolucci, M., Velsen, M. v., & Giampapa, J. (2003). The RETSINA MAS Infrastructure. *Joint issue of Autonomous Agents and MAS, Springer Netherlands, 7*(1-2), 29-48.

Wang, Z., & Garlan, D. (2000). *Task Driven Computing* (Tech. Rep. No. CMU-CS-00-154). Pittsburgh, PA: Carnegie Mellon University.

ENDNOTES

1 For generality, the protocols of interaction were renamed from previous architecture documentation (e.g. Sousa & Garlan, 2003, Sousa, 2005). For instance, the EM—Supplier protocol is now the Service Announcement and Activation Protocol (SAAP). Since these protocols were already based on peer-to-peer asynchronous communication, no changes were implied by the transition to the new perspective of the architectural framework.

2 As discussed in the subsection on context awareness, there is a variety of mechanisms for auras to obtain their physical location, or more precisely, the location of their corresponding entities.

This work was previously published in Advances in Ubiquitous Computing: Future Paradigms and Directions, edited by S. Mostefaoui, Z. Maamar, and G. Giaglis, pp. 280-315, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 8.4
Integration Concept for Knowledge Processes, Methods, and Software for SMEs

Kerstin Fink
University of Innsbruck, Austria

Christian Ploder
University of Innsbruck, Austria

ABSTRACT

Small and medium-sized enterprises (SMEs) are a vital and growing part of any national economy. Like most large businesses, SMEs have recognized the importance of knowledge management. This Chapter investigates the use of knowledge processes and knowledge methods for SMEs. The learning objectives of this Chapter are to assess the role of knowledge management and knowledge processes in SMEs. Furthermore, the reader should be able to describe major knowledge management programs in SMEs and assess how they provide value for organizations. Empirical studies conducted by the authors show that for SMEs, only four knowledge processes are important: (1) knowledge identification, (2) knowledge acquisition, (3) knowledge distribution and (4) knowledge preservation. Based on the research result of several empirical studies, an integration concept for knowledge processes, knowledge methods, and knowledge software tools for SMEs is introduced and discussed.

INTRODUCTION

The academic literature on knowledge management has become a major research field in different disciplines in the last decade (Davenport & Prusak, 1998). Through knowledge management, organizations are enabled to create, identify and renew the company's knowledge base and to deliver innovative products and services to the customer. Knowledge management is a process of systematically managed and leveraged knowledge in an organization. For Mockler and Dologite (2002, p. 18) knowledge management "refers to the process of identifying and generating, systematically gathering and providing

access to, and putting in use anything and everything which might be useful to know when performing some specified business activity. The knowledge management process is designed to increase profitability and competitive advantage in the marketplace". Before implementing a knowledge integration concept, there must be a common understanding of the term knowledge, its characteristics, and its impact on knowledge management. The multi-faceted nature of the term knowledge is reflected in a variety of definitions (Kakabadse, Kouzmin, & Kakabadse, 2001, p. 138). Davenport and Prusak (1998) use the term *"knowledge in action"* to express the characteristics of the term knowledge management in a way it is valuable for the company and to capture it in words because it resists in the minds of the humans and their action. Davenport and Prusak (1998, pp. 6) identify five key components that describe the term knowledge management:

1. The first component is *experience*. Knowledge develops over time, and it builds on the lifelong learning and training practice of an employee. Experience has a historical perspective, and it is based on the skills the knowledge-worker applies to familiar patterns to make connections between these links.
2. The second component of the term knowledge is *"ground truth"* (Davenport & Prusak, 1998) which is a term used by the U.S. Army's Center for Army Lessons Learned (CALL). CALL used the term "ground truth" to express experiences that come from the ground rather than from theories and generalizations. "Ground truth" refers to the way that the people involved know what works and what does not. CALL experts' take part in real military situations, and they pass their observations to the troops through videotapes or photos. The success of this knowledge management approach lies in "After Action Review" programs which try to cover the gap between what happened during an action and what was supposed to happen. This reflection process helps uncover disparities and differences
3. The third component is *complexity*. The skill to solve complex problems and the ability to know how to deal with uncertainties distinguish an expert from a normal employee.
4. A forth characteristic of knowledge is *judgment*. An expert can judge new situations based on experience gained over time. Furthermore, they have the ability to refine them through reflection. Knowledge, in this sense, is a living system that interacts with the environment.
5. Finally, knowledge is about heuristics and *intuition*. An expert acts based on their intuitive knowledge.

Knowledge is tacit, action-oriented, supported by rules, and it is constantly changing. In a global and interconnected society, it is more difficult for companies to know where the best and most valuable knowledge is, thus it becomes more difficult to know what the knowledge is. A successful implementation of knowledge management only can be achieved in a culture that supports knowledge sharing and transfer. An appropriate organizational culture can empower effective knowledge management. The organizational culture of a company consists of its shared values or norms which are transmitted through common beliefs and feelings, regularities of behavior, and historical processes. Trompenaars and Hampden-Turner (2006, p. 6) define culture as a group of people concerned with problem solving processes and reconciliation dilemmas. Culture itself has three different levels. The first, and highest level, is *national culture* or regional society; the second level describes *organizational culture*, and, finally, *professional culture* focuses on the knowledge of specific groups. A knowledge culture is the most important value for the implementation of knowledge management,

because organizational knowledge resides in the culture, structure and individuals who make up the organization.

Besides culture and networking, the objective for knowledge management technology is the creation of a connected environment for the exchange of knowledge (Mentzas, Apostolou, Young, & Abecker, 2001). These new software products facilitate communication and interaction among people as well as among people and systems. Mentzas et al. (2001, p. 95) discuss two key components that are required to support the sharing of information and knowledge:

- *Collaboration facilities* for knowledge workers are mainly the domain of groupware products. Other technology examples in this group are email systems, workflow automation, discussion groups, document management, shared databases, scheduling and calendar functions.
- *Discovery facilities* are required for searching and retrieval purposes. Knowledge workers are in constant need of finding and accessing information and knowledge from other experts. A wide variety of information sources support the finding of expertise, and they include the Internet, corporate Intranets, legacy systems and corporate local area networks (LANs).

Knowledge management is more than the technological solutions provided to give people access to better and more relevant information (Wang & Plaskoff, 2002, p. 113). It is important that the design of the knowledge management systems reflects the mindset of the knowledge workers and their way of offering highly qualitative knowledge solutions with quick solution processes. An effective knowledge management system must integrate people, processes, technology and the organizational structure.

Historically, knowledge management focused on the domain of larger organizations. Consequently issues of culture, networking, organizational structure and technological infrastructure have been examined upon the implementation of knowledge management initiatives in large multi-national organizations and seem to give little relevance (Delahaye, 2003) to small and medium enterprises (SMEs). However, the success and growth of SMEs depends on how well they manage the knowledge of their knowledge workers. Managers in SMEs have to recognize that the uniqueness and creativity of each knowledge worker will lead to customer satisfaction and the success of the SMEs. Dezouza & Awazu (2006) point out that SMEs have to compete with know-how in order to gain competitive advantages. As SMEs do not have much money to spend on knowledge management initiatives, knowledge must be leveraged so that goals can be achieved in an effective and efficient manner. There are several research articles dealing with knowledge management in SMEs (2001), but only a few empirical studies have been conducted to see the impact of knowledge processes in them. McAdam & Reid (2005) concluded that the time is right for knowledge management within the SME-sector. The results of their comparative study of large organizations and SMEs showed that both have much to gain from the development of knowledge management systems. Salojärvi, Furu & Sveiby (Dunkelberg & Wade, 2007) concluded that SMEs should be able to enhance their performance and competitive advantages by a more conscious and systematic approach to knowledge management.

There are several quantitative (2007, p. 240) and qualitative definitions of the term SME depending on regional and national differences. Street & Cameron (Fink & Ploder, 2007a, 2007b) conducted a literature review from 1990 until 2002 to analyze the current status of SMEs and found a variety of definitions of SMEs with the following clustering: individual characteristics of the entrepreneur, organizational characteristics of the SME, relationship characteristics,

performance characteristics, strategic planning characteristics or relationship characteristics. In 2000, the European Council set the clear strategic goal for the European Union (EU) of becoming "the most competitive and dynamic economy in the world, capable of sustaining economic growth with more and better jobs and greater social cohesion" by the year 2010 (ec.europa.eu/growthandjobs). SMEs are playing a key role in European economic performance because they account for a high proportion of the Gross Domestic Product (GDP) and employ some two thirds of the European workforce. According to the OECD Small and Medium Enterprise Outlook 2005 (www.oecd.org) SMEs are very important for strengthening economic performances. They represent over 95 percent of enterprises in most OECD countries, and generate over half of private sector development. Looking at the European countries of Austria and Switzerland including Liechtenstein a similar SME landscape can be found. According to the Austrian Statistical Year Book (www.statistik.at) and the Austrian Institute for SMEs Research (www.kmuforschung.ac.at) in the year 2006, 99.7 percent of companies in Austria, or 297,800, were SMEs. According to the data from CHSME (www.kmu.admin.ch), 99.7 percent of the companies in Switzerland are SMEs.

In the United States (US), the definition of small business is set by a government department called the Small Business Administration (SBA) Size Standards Office. The SBA uses the term "size standards" to indicate the largest a concern can be in order to still be considered a small business. It must also be independently owned and operated. Unlike the European Union, which has simple definitions applied to all industries, the United States has chosen to set size standards for each individual industry. This distinction is intended to better reflect industry differences. SMEs are also of high importance for in the US Economy. Similar to Europe, more than 97 percent of the firms in the US can be defined as SME.

A comparable influence of SMEs on economic value can be found in the report of the Asia-Pacific Economic Cooperation (www.apec.com), where about 90 percent of enterprises are SMEs. During their 2006 meeting in Beijing the members agreed to strengthen SME´s competitiveness for trade and investment. For example, SMEs account for more than 95 percent of companies in Australia. Of the 624,010 SMEs in Australia, more than two thirds employ between one and four people. A further 180,880 SMEs employ between five and 19 people meaning that 93.5 percent of people employed by SMEs in Australia are employed by what can be described as 'micro-SMEs', namely companies with fewer than 20 employees.

In section two the authors introduce an integration concept for the implementation of knowledge management systems in SMEs by taking the knowledge processes, knowledge methods and supporting knowledge software tools into consideration. Section three discusses future research and describes objectives.

INTEGRATION CONCEPT FOR KNOWLEDGE MANAGEMENT IN SMES

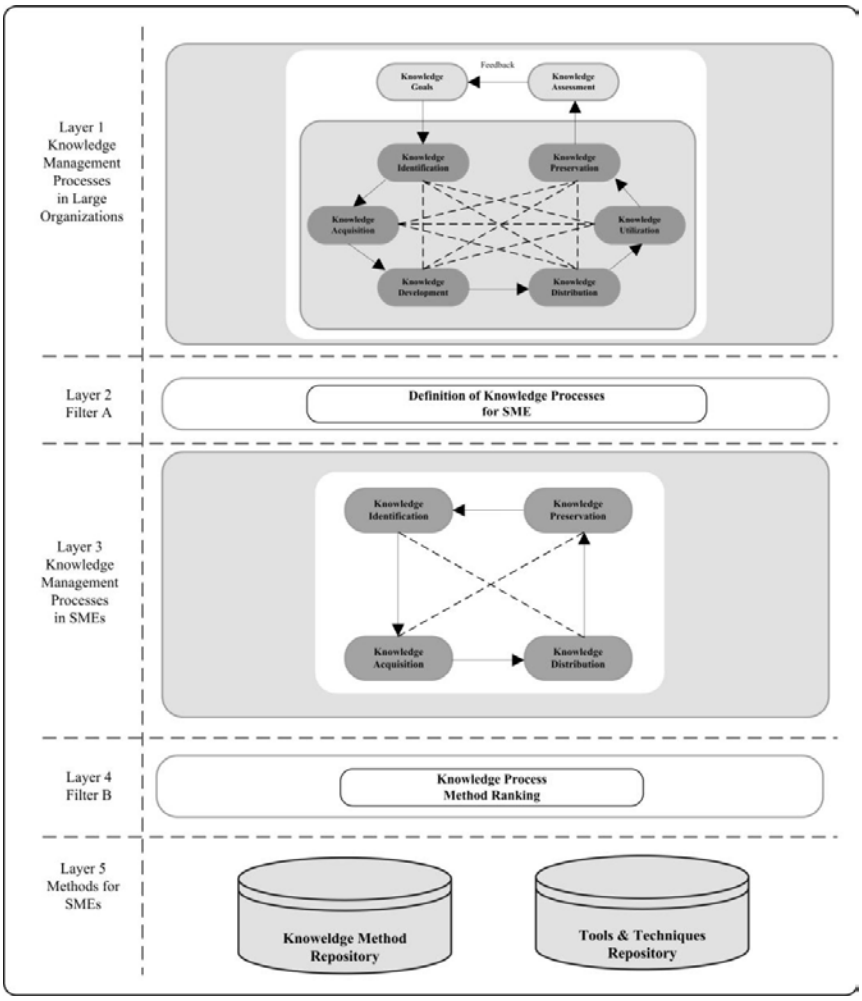
Research Framework and SME Definition

This section focuses on discussing the integration concept for SMEs. Our research findings (2006) indicate that SMEs need only four key knowledge processes (1) Knowledge Identification, (2) Knowledge Acquisition, (3) Knowledge Distribution and (4) Knowledge Preservation and therefore the authors propose a knowledge layer concept designed specifically for implementing knowledge management in SMEs. The empirical studies conducted by the authors combine the concepts of knowledge processes as well as knowledge methods for SMEs in a single study. The key objective of this section is the matching

of knowledge methods to knowledge processes in these companies. Figure 1 illustrates the research process for modeling knowledge processes in SMEs and assigning knowledge methods to each of the four key processes with supporting software tools. The basic research model is the "building block" approach by Probst, Raub & Romhardt (Laudon & Laudon, 2006) with their description of the knowledge processes (Figure 1, layer 1). Involved are eight components that form two cycles, one inner cycle and one outer cycle. Among other knowledge process models (2006), the building block approach of Probst, Raub & Romhardt (Edwards & Kidd, 2003) has the advantage that it is well known in European companies, including SMEs, and furthermore it has a unique and complete design.

The authors use the definition of SMEs of the European Commission 2006 for their research design. The European Commission analyzes SMEs by using the following three characteristics: (1) number of employees, (2) annual turnover and (3) total assets. Characterized through these three factors, the European Commission differentiates between (1) middle enterprises [fewer than 250 employees and less than EURO 50 million

Figure 1. Knowledge integration layer concept for SMEs



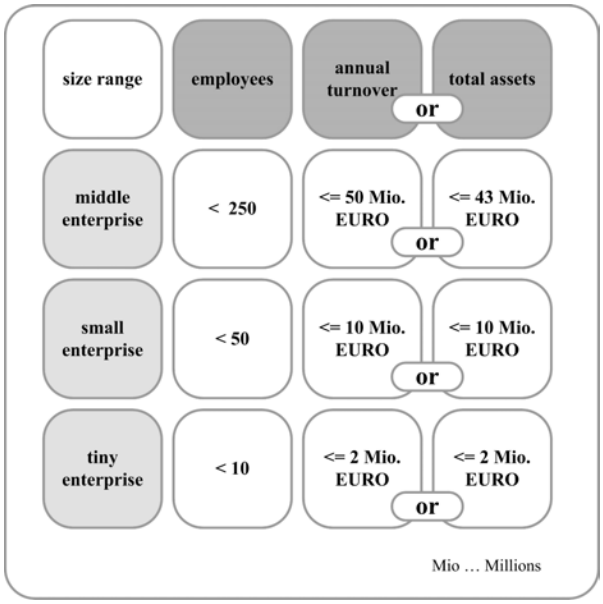
annual turnover or less than EURO 43 million total assets], (2) small enterprises [fewer than 50 employees and less than EURO 10 million annual turnover or less than EURO 10 million total assets] and (3) micro enterprises [fewer than 10 employees and less than EURO 2 million annual turnover or less than EURO 2 million total assets]. Figure 2 illustrates the European Commission's definition of SMEs. Focusing on this definition, the authors follow the research view of a quantitative perspective of SMEs. This means, that all enterprises with fewer than 250 employees and less than EURO 50 million annual turnover or less than EURO 43 million total assets in Austria and Switzerland including Liechtenstein are the target population. In figure 1 layer 2 symbolizes the quantitative view of the SME definition.

Knowledge Processes Approaches (Layer 1)

Business process modeling (Davenport & Prusak, 1998) has become a major research field in the information systems discipline in the last ten years. Davenport sees the term business process as "a structured, measured set of activities designed to produce a specified output from a particular customer or market" (Probst et al., 2006; Rao, 2004). However, in recent years, not only business process management, but also knowledge management has been developing into a new research field (2003, p. 124). The term knowledge process modeling comes from the linking of these two research fields. Key features of knowledge intensive processes can be described as follows: diversity of sources and media, variance and dynamic development of the process organization, plentiful process participants with different expertise, use of creativity, high level of innovation and influence on the area of the decision. Edwards and Kidd (2000) used the following five characteristics to emphasize that knowledge management and business process management should be integrated:

- Knowledge management is important for business if the initiative implies an advantage

Figure 2. SME definition according to the European Commission definition



for the customers. The idea of implementing the customer's requests – may be internal or external – is the basis for including the customer.
- Knowledge doesn't follow business borders. Business processes also model activities carried out by global trading companies and lay the foundations for modeling knowledge intensive processes.
- Knowledge management can only be efficient if it follows a structured model. Business processes are modeled by structured actions and they are necessary to deduce knowledge intensive processes.
- The success of knowledge management depends on the measurement of knowledge. There is a similarity to the measurement of business processes. The measurement of the knowledge potential provides a central position and biases the success.
- Knowledge management is affected by a holistic approach. Every part of the business process modeling is important for success but every aspect should be considered.

In addition, knowledge management and business process modeling initially focused on large companies. The knowledge economy has to shift from focusing on large companies to small and medium-sized enterprises (SME) because of their importance for industrial economies. The main reason behind today's change of focus is that all businesses depend on methods and tools of knowledge management in order to gain competitive advantages and to deal with the knowledge potential of their employees.

The basic research model is the "building block" approach by Probst, Raub & Romhardt (2006) with their description of the knowledge processes (Figure 1, layer 1). Involved are eight components that form two cycles, one inner cycle and the other outer cycle. The inner cycle is composed of six key knowledge processes:

- *Knowledge Identification* is the process of identifying external knowledge for analyzing and describing the company's knowledge environment.
- *Knowledge Acquisition* refers to what forms of expertise the company should acquire from outside through relationships with customers, suppliers, competitors and partners in co-operative ventures.
- *Knowledge Development* is a building block which complements Knowledge Acquisition. It focuses on generating new skills, new products, better ideas and more efficient processes. Knowledge Development includes all management actions consciously aimed at producing capabilities.
- *Knowledge Distribution* is the process of sharing and spreading knowledge which is already present within the organization.
- *Knowledge Utilization* consists of carrying out activities to ensure that the knowledge present in the organization is applied productively for its benefit.
- *Knowledge Preservation* is the process where the selective retention of information, documents and experience required by management takes place.

In addition, there are two other processes in the outer cycle, Knowledge Assessment and Knowledge Goals, which provide the direction to the whole knowledge management cycle:

- *Knowledge Assessment* completes the cycle, providing the essential data for strategic control of knowledge management.
- *Knowledge Goals* determine which capabilities should be built on which level.

Knowledge Processes in SMEs (Layer 2 and 3)

The research method for the identification of knowledge processes in SMEs were expert

interviews or what Gillham (Gillham, 2000, p. 64) referred to as "elite interviewing". This kind of interviewing is chosen to address someone in a special position or an expert. Gillham lists several characteristics of open-ended interviews (Davenport & Prusak, 1998; Ruggels, 1997):

- The respondents will know more about the topic and the setting than the interviewer. Sometimes they can even tell the interviewer what questions to ask.
- By virtue of their authority and experience, they will have their own structuring of their knowledge. They will not allow an interview for which they have to answer a series of questions addressed at them.
- The best thing the interviewer can hope for is a response to a topic raised.
- The experts can be particularly informative about the location of documents, records, or other experts.
- The experts will expect some control over the interviewer, and they also will demand a level of accountability and feedback.

These five characteristics of elite interviews also apply to the interview situation for the knowledge processes in SMEs. The managers or company owners were highly motivated to articulate their view of knowledge processes and wanted to share their personal position about the key knowledge processes. The first interview session was conducted in 2004 and was limited to Austrian SME managers. This study was the proving ground for the future procedure of the empirical studies in 2005/2006/2007. The research hypothesis was:

Hypothesis: SMEs need a simple knowledge process model in order to implement knowledge management successfully.

The second interview session was conducted from December 2005 to February 2006 (Figure 1, layer 2). The research method was the elite interview. The data sample ranged from all industry sectors in which SMEs could be found in Austria at this time with a special focus on enterprises in the of consulting and information technology sector. The survey subjects were CIOs (Chief information officer) and CEOs (Chief executive officer) in Austrian and Swiss SMEs as these are recognized as proficient in answering questions concerning knowledge management (1980, p. 21). The data sample of 36 interviewees was the proving ground for asking open ended questions. The data was analyzed by content analysis defined by Krippendorf (Davenport & Prusak, 1998) as "a research technique for making replicable and valid inferences to the content". The interview sessions lasted approximately one hour and the authors were the interviewers. The result of the Austrian and Swiss research showed clearly that in both countries only four knowledge processes identified in the Probst, Raub & Romhardt (2006) model are ranked as important for the implementation of knowledge management in SMEs (see Figure 1, layer 3):

- **Knowledge identification:** In SMEs it is highly important to identify the key sources of knowledge, experiences and know-how in order to stay competitive on the market.
- **Knowledge Acquisition:** The know-how of SMEs resides in many cases in the head of the experts or knowledge worker.
- **Knowledge distribution:** This process focuses on the sharing of explicit and implicit knowledge between knowledge workers in SMEs. As SMEs are characterized by smaller groups, a knowledge sharing culture which facilitates the exchange of knowledge with other groups and utilizes knowledge tools and mechanisms is especially important.
- **Knowledge preservation:** It is well recognized that the most critical asset of any company are the sum of its collective knowl-

edge and intellectual property (Laudon & Laudon, 2006; Schwartz, 2006). Knowledge preservation and growth of this asset requires effective knowledge management throughout SMEs, so as to make sure that the right information is available to the right people when they need it. In addition, the managers of the SMEs in our study pointed out that the process of *knowledge disposal* is also relevant for SMEs with the objective of not overloading the information flow between the individuals. From the content analysis of the expert interviews with Austrian and Swiss managers, knowledge disposal can be identified as an integrated part of knowledge preservation.

There were no significant differences in the answers given by the managers of SMEs in Austria and Switzerland. In general, it can be stated that SMEs are satisfied with only four knowledge processes instead of the original framework with eight building blocks. This implies that hypothesis 1 is verified. These four key knowledge processes and the basic framework they provide for assigning knowledge methods in SMEs will be part of our future research.

Knowledge Methods (Layer 4)

Based on a literature review (Fink & Ploder, 2007b) a list of existing knowledge methods which support one of the four key knowledge processes was developed (Figure 1, layer 4). The objective of this empirical study was to find out which of the methods are most relevant for SMEs. The data sample of 587 enterprises was stochastically appointed from the target population. It was average allocated across the federal states of Austria, Switzerland to get a representative result. In Austria there are 535,031 SMEs and in Switzerland/Liechtenstein there are 308,819. The online questionnaire was carried out in summer 2006 after a pre-test with 30 respondents. The respondents were divided into seven industry sectors. 60 percent were from the three key industries: industry, information & consulting and trade & handcraft, with the remaining 40 percent dispersed over various other industries. Figure 3 lists the key methods for the four knowledge management

Figure 3. List of knowledge methods

processes, which are stored in the knowledge method repository (Figure 1, layer 5) and can be extended to new knowledge methods.

Knowledge Software-Support (Layer 5)

In a next step, the objective was to match a cost-efficient software product to each knowledge method which is usable in practice. In the research design the focus was on Freeware and Shareware software tools in order to fulfill the presetting of cost-efficient software support. An online research method was used which resulted in a list of evaluated cost-efficient software products. The evaluation of each software product was conducted by applying the ISO/IEC 9126 norm. The Quality Model of the norm is divided into two parts which are important for the evaluation of the software products to support knowledge methods:

- the internal and the external quality of the software as well as
- the quality for use.

The ISO norm (see Figure 4) lists five characteristics to evaluate software products: (1) functionality, (2) reliability, (3) usability, (4) efficiency and (5) assignability. For each characteristic a different number of items were assessed by a likert scale from -2 up to +2. The process used by the authors to make the assessment is shown in the appendix. The data sample of the Quality Model included more than 200 different software products. A key research finding was that some of the software products cannot be used in practice because their quality is inadequate. Finally there were 45 software products which are suitable for use in SMEs.

Table 1 gives an overview of all methods supporting the four knowledge processes for SMEs and the corresponding cost-efficient software tools. Table 1 also lists the absolute number of

Figure 4. Example of ISO 9126-1

Functionality	ISO Ranking	Reliability	ISO Ranking	Usability	ISO Ranking	Efficiency	ISO Ranking
Accuracy	2	Maturity	2	Comprehensibility	1	Time Responsibility	1
Adequacy	2	Fault Tolerance	1	Learnability and Usability	2	Resource Responsibility	1
Interoperability	1						
Subtotal	5		3		3		2

Assignability	ISO Ranking
Installation	1
Conformance	1
Compatibility	2
Subtotal	4

Ranking	17

Process	Knowledge Acquisition
Method	Brainstorming
Software	Concept X7

Costs	149 EURO
Disk Space	74,7 MB
License	License for 1 User
Annotation	supporting tablet computers, great functional range

Table 1. Ranking of cost-efficient software products (2006, p. 50ff)

	Ranking	Supporting cost-efficient software products	ISO Ranking	Ranking Survey
Knowledge Identification				
Knowledge Balance	92	no cost-efficient software product, Office similar products		
Balanced Scorecard	89	no cost-efficient software product, Office similar products		
Skandia Navigator	74	no cost-efficient software product, commercial Software		
Market - Asset Value - Method	-5	no cost-efficient software product, Office similar products		
Tobin's q	-15	no cost-efficient software product, Office similar products		
Knowledge Acquisition				
Search Engine	232	Google Desktop Search; MSN Toolbar; Yahoo Dektop Suche	not possible	25; 12; 10
Brainstorming	225	Brainstorming Toolbox; Concept X7	6;17	44; 88
Knowledge Network	203	no cost-efficient software product		
Mind Mapping	195	Free Mind; Think Graph, Tee Tree Office	16; 12; 8	69; 53; 28
eMail System	134	Pegasus Mail; Thunderbird Mail; Amicron Mailoffice 2.0	21; 21; 12	63; 165; 26
Scenario Technique	126	no cost-efficient software product, Office similar products		
System Simulation	98	no cost-efficient software product, commercial Software		
Business Game	91	Gamma	15	75
Synektik	-17	no cost-efficient software product, commercial Software		
Knowledge Distribution				
eMail System	185	Pegasus Mail; Thunderbird Mail; Amicron Mailoffice 2.0	16; 12; 8	63; 165; 26
Handbook FAQs	159	no cost-efficient software product, Office similar products		
Communities of Practice	152	no cost-efficient software product, Office similar products		
Groupware	139	eGRoupware1.2; AlphaAgent 1.6.0; Tiki CMS - Groupware	15; 14; 16	40; 26; 24
Questionnaire	110	Easy Survey	10	61
Best Practice	108	no cost-efficient software product, Office similar products		
Checklist	103	CUEcards 2000	8	128
Lessons Learned	103	no cost-efficient software product, Office similar products		
Knowledge Maps	82	InfoRapid KnowledgeMap	13	69
Story Telling	42	no cost-efficient software product, Office similar products		
Chatroom	29	Skype; MSN, ICQ	not possible	71; 33; 25
Microarticle	2	no cost-efficient software product, Office similar products		
Knowledge Preservation				
Database	242	MySQL; MSDE		86; 44
Mind Mapping	200	Free Mind; Think Graph, Tee Tree Office	16; 12; 8	69; 53; 28
Document Management System	195	Office Manager; UDEX dotNETContact; QVTutto	15; 15; 14	74; 35; 22
Checklist	164	CUEcards 2000	8	128
Content Management	126	CONTEX; ContentKit; VIO MATRIX	16; 13; 13	0; 47; 13

continued on following page

Table 1. continued

Project Review	122	no cost-efficient software product, Office similar products		
Experts System	74	KnowIT; KnowME	10; 7	38; 52
Conceptualization	40	no cost-efficient software product		
Neural Network	-10	no cost-efficient software product, commercial Software		

each method in the likert scale. The ranking of each method is the calculated value based on the likert scale. The "ISO Ranking" illustrates the assessment of the software based on the Quality Model. The absolute frequency with which the software was named by the respondents can be seen in the last column.

Knowledge Balance (92) was ranked highest among the methods for the first process of the *identification of knowledge*. 56% of SMEs think that this is the best method. Further methods are the Balanced Scorecard (89) and the Skandia Navigator (74). The methods Market-Asset Value-Method (-5) and Tobin`s q (-15) were rated by less than 30% to be of good use in SMEs.

Brainstorming (225) and Knowledge Network (203) are popular methods for the *acquisition of knowledge*. Mind Mapping (195), eMail Systems (134), Scenario Technique (126) and System Simulation (98) are also suitable methods for this knowledge process, while Business Games (91) are also a possibility. The method of "Synektik" was rated very low because of its complexity. The absolute star for the acquisition of knowledge was the Search Engine (232) with over 70% for efficient use in SMEs. In this case the Google Desktop Search Engine was the prior selection software. 60% of the respondents chose eMail-Systems which can be supported by the software Thunderbird1.5. For Brainstorming a good tool is Concept X7, while for Mind Mapping the tools Free Mind (42%) and Think Graph (41%) were rated highly. As support software to a Business Game 64% rated Gamma was well

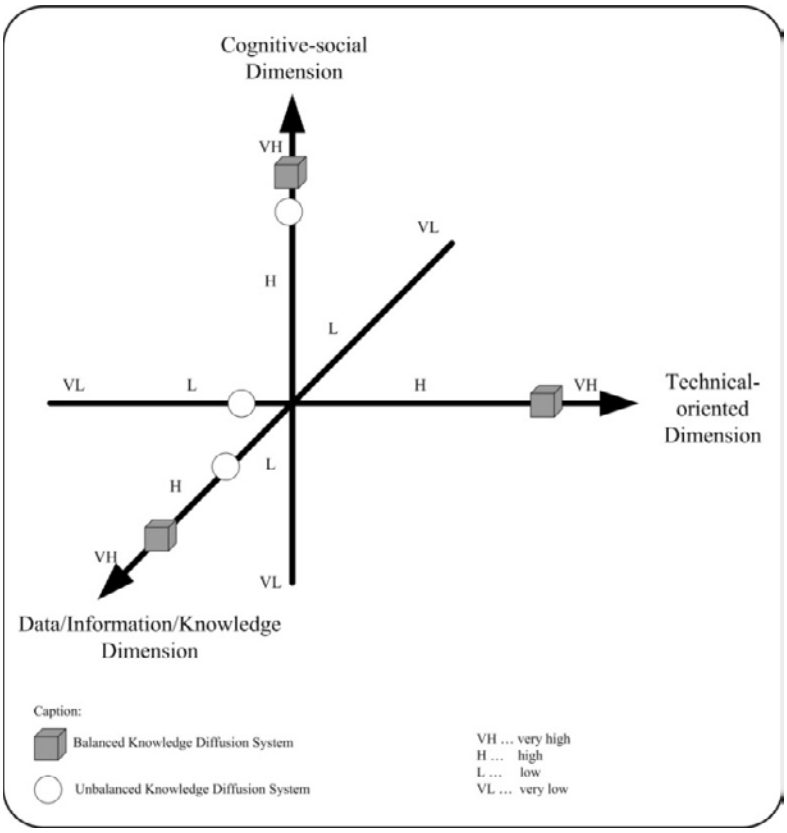
As illustrated in Table 1 the methods eMail-System (185), Handbook FAQs (159), Communities of Practice (152), Groupware (139), Questionnaire (110) and Best Practice (108) are the favorites for the *distribution of knowledge*. It has to be pointed out that the methods Micro Article (2) and Chatroom (29) are rated not as well in the survey. The software products for the methods of transferring knowledge are InfoRapid supporting Knowledge Maps, EasySurvey supporting Questionnaire, Skype and MSN supporting Chatroom, eGroupware1.2 and AlphaAgent1.6.0 supporting Groupware, CUCards 2000 supporting Checklists and Pegasus Mail, Thunderbird1.5 and Amicron Mailoffice 2.0 for the support of eMail-Systems.

Databases (242) are a recognized method of *the knowledge preservation process.* 80% of the SMEs think that they will organize their knowledge with databases. Mind Mapping (200), Document Management System (195) and Checklists (164) are further efficient methods. Content Management Systems (126), Project Review (122), Expert Systems (74) and Conceptualization (40) are methods which can be chosen but are not the favorite choice. Neural Network (-10) is not an adequate method for preserving knowledge in an SME. There were many different software products to support this process. MySQL is the favorite database software followed by the MSDE from Microsoft. Document management can be done by the Office Manager, the UDEX dotNETContact or the QVTutto. There are also software tools for the other methods which are described in Table 1.

CONCLUSION AND SECTION OBJECTIVES

In the future the research area will be expanded from Austria, Germany and Switzerland to other key members of the European Union as well as to the U.S. and Asia-Pacific area. Furthermore, in future research work the authors will focus on knowledge diffusion to the dissemination of explicit knowledge captioned in SMEs from many sources. The diffusion of knowledge through an effective SME website creates benefits not only to the enterprise itself but also to customers and suppliers as well as to new alliances. Empirical studies conducted by Ordanini show that the website is the only solution which has been used by the majority of SMEs, while the adoption of a highly-sophisticated website is a relevant matter for only 10 percent or 20 percent of SMEs. The use of information technology, especially websites, is recognized as a critical success factor for knowledge management initiatives in the SME sector (Wong & Aspinwall, 2005). Wong (2005) sees information technologies as a key enabler for the implementation of knowledge management and considers factors such as the simplicity of technology, ease of use, suitability for users' needs, relevancy of knowledge content, and standardization of a knowledge structure as key factors for knowledge diffusion in the development of a knowledge management systems. Knowledge diffusion through websites is a dominant factor for successful knowledge initiatives. The central research question of this section can be described as follows: How can SMEs spread their knowledge competencies through their websites? For successful knowledge diffusion (Fink & Ploder, 2007a) in SMEs the authors propose a three-dimensional theoretical framework as shown in figure 5: (1)

Figure 5. Three-dimensional framework for knowledge diffusion



Data/Information/Knowledge Dimension, (2) Technical-oriented Dimension and (3) Social-cognitive Dimension.

The major learning objectives of this chapter are:

- Identification and description of contemporary approaches to knowledge management in SMEs.
- Assessing contemporary knowledge processes that are required for successful implementation of knowledge management in SMEs.
- Identification of major knowledge methods supporting the key knowledge processes in SMEs: identification, acquisition, dissemination, preservation referring to the layer concept
- Discussion of the impact of cost-efficient software support for knowledge initiatives.
- Analyzing the relationship of knowledge transfer through SME websites.

In summary, it can be stated that the use of an integration concept can help SMEs leverage their core competencies by promoting the sharing of information and knowledge inside and outside their organization. The impact of cost-efficient software products for knowledge management facilitates business models based on simple knowledge processes. The knowledge process model highlights specific processes in the business where competitive advantages can be achieved and knowledge management systems will have a greater impact. The integration concepts view knowledge management in SMEs as a holistic approach where primary knowledge processes are directly related to knowledge methods and software tools.

REFERENCES

Davenport, T., & Prusak, L. (1998). *Working knowledge: how organizations manage what they know.* Boston, MA: Harvard Business School Press.

Delahaye, D. (2003). Knowledge Management in a SME. *International Journal of Organisational Behaviour, 9*(3), 604-614.

Dezouza, K., & Awazu, Y. (2006). Knowledge Management at SMEs: Five peculiarities. *Journal of Knowledge Management, 10*(1), 32-43.

Dunkelberg, W., & Wade, H. (2007). Overview: Small Business Optimism. *NFIB Small Business Economic Trends,* 1-12.

Edwards, J., & Kidd, J. (2003). Bridging the Gap from the General to the Specific by Linking Knowledge Management to Business Process Management. In V. Hlupic (Ed.), *Knowledge and Business Process Management.* Hershey: Idea Group Publishing.

Fink, K., & Ploder, C. (2007a). A comparative Study of Knowledge Processes and Methods in Austrian and Swiss SMEs. In H. Österle, J. Schelp & R. Winter (Eds.), *Proceedings of the 15th European Conference on Information Systems (ECIS2007).* St. Gallen.

Fink, K., & Ploder, C. (2007b). Knowledge Process Modeling in SME and Cost-Efficient Software Support: Theoretical Framework and Empirical Studies. In M. Khosrow-Pour (Ed.), *Managing Worldwide Operations and Communications with Information Technology.* Hershey: IGI Publishing.

Gillham, B. (2000). *Case Study Research Methods.* London/New York: Continuum.

Kakabadse, N., Kouzmin, A., & Kakabadse, A. (2001). From Tacit Knowledge to Knowledge Management: Leveraging Invisible Assets. *Knowledge and Process Management, 8*(3), 137-154.

Krippendorf, K. (1980). *Content Analysis,* 5. Beverly Hills: Sage Publication.

Laudon, K. C., & Laudon, J. P. (2006). *Management information systems: managing the digital*

firm (9th ed.). Upper Saddle River, NJ: Pearson/ Prentice Hall.

McAdam, R., & Reid, R. (2001). SME and large Organization Perception of Knowledge Management: Comparison and Contrast. *Journal of Knowledge Management, 5*(3), 231-241.

Mentzas, G., Apostolou, D., Young, R., & Abecker, A. (2001). Knowledge Networking: a Holistic Solution for Leveraging Corporate Knowledge. *Journal of Knowledge Management, 5*(1), 94-106.

Mockler, R., & Dologite, D. (2002). Strategically-Focused Enterprise Knowledge Management. In D. White (Ed.), *Knowledge Mapping & Management* (pp. 14-22). Hershey: IRM Press.

Ordanini, A. (2006). *Information Technology and Small Businesses: Antecedents and Consequences of Technology Adoption*. Massachusetts: Edward Elgar Publishing.

Probst, G., Raub, S., & Romhardt, K. (2006). *Wissen Managen*, 5. Wiesbaden: Gabler Verlag.

Rao, M. (2004). *Knowledge Management: Tools and Techniques.* Oxford: Elsevier.

Ruggels, R. (1997). *Knowledge Management Tools.* Boston: Butterworth-Heinemann.

Salojärvi, S., Furu, P., & Sveiby, K. (2005). Knowledge management and growth in Finnish SMEs. *Journal of Knowledge Management, 9*(2), 103-122.

Schwartz, D. (2006). *Encyclopedia of Knowledge Management.* Hershey: Idea Group Publishing.

Street, C., & Cameron, A. (2007). External Relationships and the Small Business: A Review of Small Business Alliance and Network Research. *Journal of Small Business Management, 45*(2).

Trompenaars, F., & Hampden-Turner, C. (2006). *Riding the waves of culture: understanding cultural diversity in business* (2. reprint. with corr. ed.). London: Brealey.

Wang, F., & Plaskoff, J. (2002). An Integrated Development Model for KM. In R. Bellaver & J. Lusa (Eds.), *Knowledge Management Strategy and Technology* (pp. 113-134). Boston: Artech House.

Wong, K. (2005). Critical success factors for implementing knowledge management in small and medium enterprises *Industrial Management & Data Systems, 105*(3), 261-279.

Wong, K., & Aspinwall, E. (2005). An empirical study of the important factors for knowledge-management adoption in the SME sector. *Journal of Knowledge Management, 9*(3), 64-82.

KEY TERMS

Enterprise: Considered to be any entity engaged in an economic activity, irrespective of its legal form. This includes, in particular, self-employed persons and family businesses engaged in craft or other activities, along with partnerships or associations regularly engaged in economic activities. The category of micro, small and medium-sized enterprises (SME) is made up of enterprises which employ fewer than 250 persons and which have an annual turnover not exceeding EURO 50 million, and/or an annual balance sheet total not exceeding EURO 43 million. Within the SME category, a small enterprise is defined as an enterprise which employs fewer than 50 persons and whose annual turnover and/or annual balance sheet total does not exceed EURO 10 million. Within the SME category, a micro enterprise is defined as an enterprise which employs fewer than 10 persons and whose annual turnover and/or annual balance sheet total does not exceed EURO 2 million.

Knowledge Distribution: Can be defined as the transfer of knowledge within and across settings, with the expectation that the knowledge will be "used" conceptually (as learning, enlightenment, or the acquisition of new perspectives

or attitudes) or instrumentally (in the form of modified or new practices.). There are those who see distribution as having other legitimate outcomes. Some of these outcomes include: (1) increased awareness; (2) ability to make informed choices among alternatives and (3) the exchange of information, materials or perspectives.

Knowledge Integration Concepts: Aim to customize knowledge processes and knowledge methods for SMEs in a single enterprise solution platform. Enterprise application such as knowledge management systems are designed to support the SME orientation of business and knowledge processes to that the SMEs can operate efficiently.

Knowledge Management: Can be seen as the overall dealing with knowledge. Knowledge is a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of those who know. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices and norms (Davenport & Prusak, 1998).

Knowledge Management Systems: A fast growing area of corporate software investment. Contemporary technologies such as Portals, Content Management Systems, Search engines, ontologies help managers and employees in their daily decisions and processes. At each level of the organization, knowledge management systems support the major knowledge processes of the business.

Knowledge Methods: Support knowledge processes and are designed to add value to these within organizations. Depending on the identification of industry specific knowledge processes, SMEs have to choose from a knowledge method repository the corresponding knowledge method.

Knowledge Processes: Accelerate the company's business processes while ensuring compliance with the knowledge of the employees. Knowledge processes concentrate on the identification, acquisition, dissemination and preservation of knowledge in order to gain competitive advantages and enhance the value of the company. Organizations have to use the ability to incorporate their knowledge into their business processes.

This work was previously published in Handbook of Research on Enterprise Systems, edited by J.N.D. Gupta, S. K. Sharma, and M.A. Rashid, pp. 185-200, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.5
Adaptive Computation Paradigm in Knowledge Representation:
Traditional and Emerging Applications

Marina L. Gavrilova
University of Calgary, Canada

ABSTRACT

The constant demand for complex applications, the ever increasing complexity and size of software systems, and the inherently complicated nature of the information drive the needs for developing radically new approaches for information representation. This drive is leading to creation of new and exciting interdisciplinary fields that investigate convergence of software science and intelligence science, as well as computational sciences and their applications. This survey article discusses the new paradigm of the algorithmic models of intelligence, based on the adaptive hierarchical model of computation, and presents the algorithms and applications utilizing this paradigm in data-intensive, collaborative environment. Examples from the various areas include references to adaptive paradigm in biometric technologies, evolutionary computing, swarm intelligence, robotics, networks, e-learning, knowledge representation and information system design. Special topics related to adaptive models design and geometric computing are also included in the survey.

INTRODUCTION

Adaptive computing focuses on the methodology and implementation of algorithms and systems that can adjust to different situations and circumstances. An adaptive system may change its own behavior depending on the goals, tasks, and other features of individual users and the environment. Adaptivity is important for ubiquitous and pervasive computing, and as it will be shown in this survey, plays an important role in a variety of traditional as well as emerging areas, such as biometric technologies, evolutionary computing, swarm intelligence, robotics, networks, e-learning, knowledge representation and information system design.

The constant demand for complex applications, the ever increasing complexity and size of software systems, and the inherently complicated

nature of the information drive the needs for developing radically new approaches for information representation and processing. This drive is leading to creation of new and exciting interdisciplinary fields that investigate convergence of software science and intelligence science, as well as computational sciences and their applications. As can be seen from the definition, the driving force behind the need for adaptive paradigm is variety of situations, variability in backgrounds and needs of different user groups or applications. This survey article presents the new paradigm of the algorithmic models of intelligence, based on the adaptive hierarchical model of computation, and presents the algorithms and applications utilizing this paradigm in data-intensive, collaborative environment.

ADAPTIVE METHODS IN TERRAIN MODELING

For a long time, researchers were pressed with questions on how to model real-world objects realistically, while at the same time preserving efficiency, quality and operability requirements. The examples from the area of computer graphics and terrain modeling showcase the concept perfectly. Over the past twenty years, a grid, mesh, TIN, k-d trees, and Voronoi based methods for model representation were developed (Bonnefoi and Plemenos 2000, Gold and Dakowicz 2006, Cohen-Or and Levanoni 1996, Duchaineauy et. al. 1997, Franc and Skala 2002, Iglesis 2002, Kolingerová 2002). Most of these were however static methods, not suitable for rendering dynamic scenes or preserving higher level of details (see Figure 1.). In 1997, first methods for dynamic model representation: Real-time Optimally Adapting Mesh (ROAM) and Progressive Mesh (PM), were developed (Duchaineauy 1997). However, even with the further improvements (Li et. a. 2003), these methods were not capable of dealing with large amount of complex data or significantly varied level of details (see Figure 2.). The main difference between terrain visualized using static and adaptive methods is the size and distribution of the triangles – in Figure 1, it is clearly seen that the patches of similar triangles are used throughout the various terrain features, while Figure 2 uses adaptive methods to decide on the most appropriate triangle sizes based on the curvature and distance from the viewer. However, this method is still not sufficient for dealing with all variety of terrain features, nor it is fast enough to be used in real-time.

Recently, the adaptive multi-resolution technique for real-time terrain rendering was developed (Apu and Gavrilova 2005). The method is characterized by the efficient representation of massive underlying terrain, utilizes efficient transition between detail levels, and achieves frame rate constancy ensuring visual continuity. The method is based on the adaptive loop subdivision and recursive split operation (see Figure 3.), implemented with the use of novel S-Queue operations ordering data structure.

Furthermore, a novel approach based on adaptive dynamic viewer-dependent level of details (LOD), utilizing the above strategy, was developed for real-time terrain rendering. The approach uses mesh regularity operator and LOD control parameters to achieve fast recursive seamless patch stitching, ensure geometric regularity, improve rendering quality, provide multi-resolution storage and allow for rendering and transmission of massive data sets (see Figure 4).

More formally, the process can be described as follows. A mesh M can be viewed as a piecewise linear surface. It is defined as a pair (K, V) where $V \subset \mathbb{R}^3$ is the set of vertices and K is a simplicial complex specifying the connectivity of the mesh simplices (the adjacency of the vertices, edges and faces). A combinatorial k-simplex of K is the $(k + 1)$ element subset of K. Therefore the 0-simplices $\{i\} \in K$ are called vertices, the 1-simpices $\{i, j\} \in K$ are called edges, and the 2-simplices $\{i, j, l\} \in K$ are called faces.

Figure 1. A static mesh (40,000 triangles)

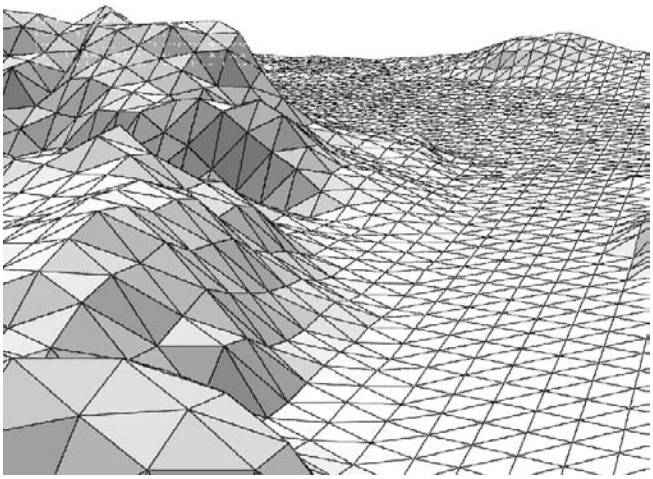


Figure 2. Real-time optimally adaptive mesh (25,000 triangles)

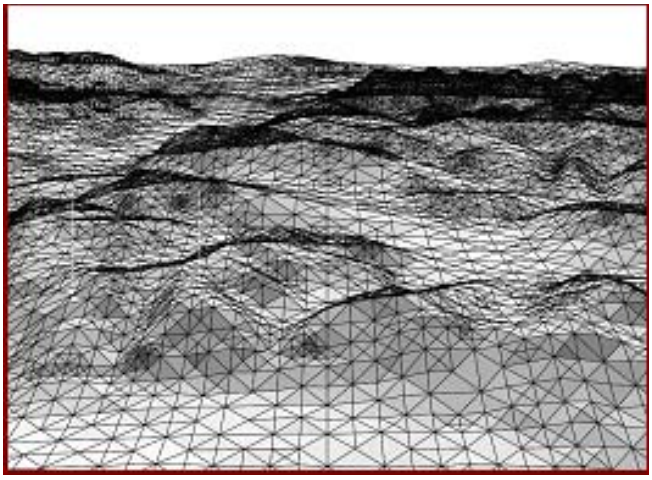


Figure 3. Adaptive loop subdivision and recursive split operations

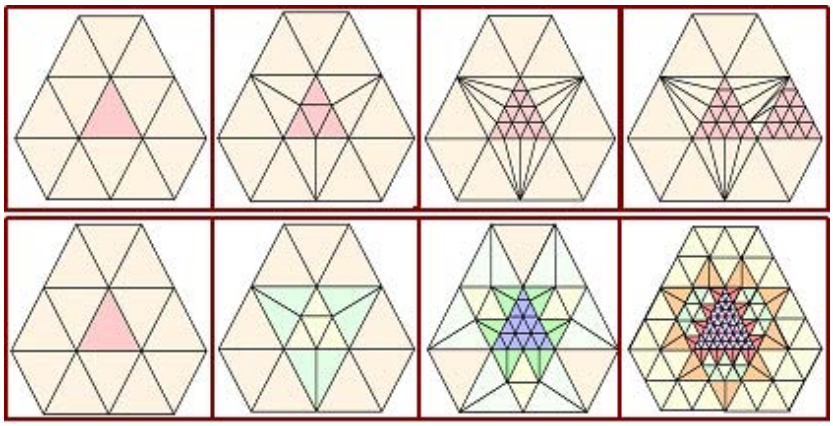
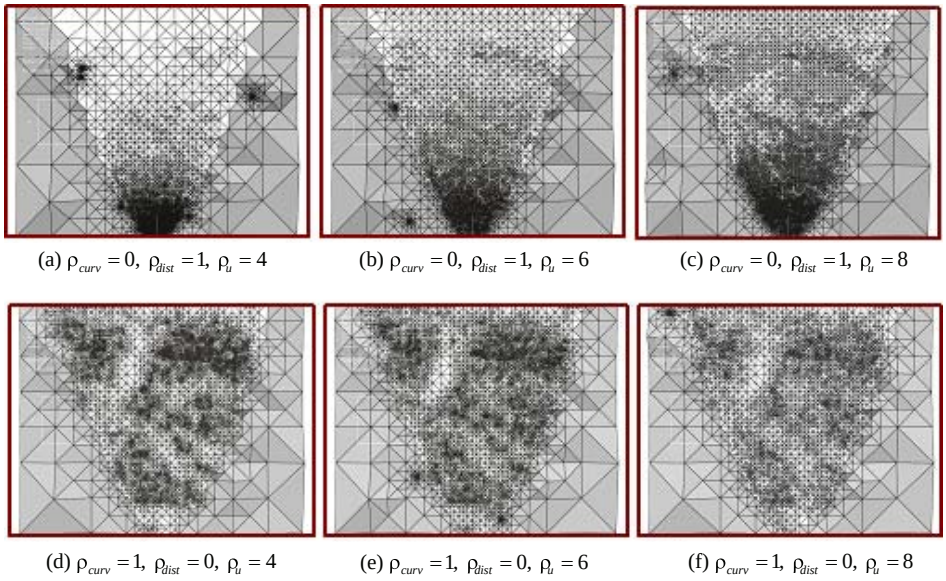


Figure 4. Varied Mesh regularity operator and LOD control parameters



A mesh can be considered as higher dimensional generalization of the concept of graph used in a variety of graphics applications. For instance, a mesh $M = (K, V)$ defines a graph $G = (V, \wp)$ where V is the set of vertices in M and $\wp$ is the 1-simplices of K. Therefore G is the minimum unrestricted topological realization of M. Nevertheless, higher simplices of K (i.e. faces) could be reconstructed from G by means of a convex combinatorial relationship of the transitive closure of $\wp$. For example, a triangulation of the mesh could be obtained from the transitive closure of edges defining a mesh M.

Mesh optimization is the process of reducing a mesh M_0 to a mesh M_k, where M_k contains less number of vertices and geometric primitives (i.e. triangles) than M_0. The goal is to find M_k such that the no other mesh M'_k exists, which represent a better approximation of the mesh M_0. In practical applications, it is sufficient to converge to any of the optimal solutions. There are many variations of mesh optimization techniques. Intuitively, mesh optimization is a compression method for geometric models with large details. In computer graphics, various processes such as regular tessellations, polygonized spline surfaces, polygonized parametric surfaces, polygonized implicit surfaces, range scanned surfaces and subdivision surfaces generate polygonal meshes to render or store the object. In most cases, these meshes consist of a large number of geometric primitives. In order to render them or store them efficiently, one needs to reduce the number of primitives significantly. This process of simplification must be performed in a way such that the loss of details is minimal while conforming as much as possible to the specified criteria.

The novel approach to mesh optimization based on adaptive dynamic viewer-dependent level of detail method was developed in (Apu and Gavrilova 2005). The introduced data structure, called Adaptive Loop Subdivision (ALS), can intuitively be viewed as a special filter that increases the details and smoothness properties of a mesh. It is a subdivision method based on triangle meshes. In general, every triangle in the base mesh M^i is split into four inner triangles (see Figure 4b). Each vertex position is adjusted ac-

cording to a combinatory mask. The refined mesh M^{i+1} contains exactly four times the number of triangles than M^i. This four to one augmentation of LOD is a direct correspondent to the refinement of Haar wavelet.

Standard subdivision technique was first introduced by Loop in 1987 (Loop, 1987) in his Master's Thesis. It became widely acceptable and numerous improvements were suggested subsequently. However, the scheme does not allow local refinements. If one triangle is subdivided without the neighboring triangles, cracks will become visible. The new ALS scheme allows a way to repair those cracks without noticeable visual artifacts through the following conditions imposed on subdivision procedure:

1. If f_i are the faces adjacent to the face f in M^i and $\Im: F(M^i) \rightarrow \mathbb{Z}$ is a function corresponding to the LOD of a face ($F(M_i)$) is the set of faces of M^i) then:

   $$\forall i(\text{neighbor}(f, f_i) \rightarrow 0 \leq |\Im(f) - \Im(f_i)| \leq 1$$

2. A face f can be subdivided if and only if $\aleph(f) = 1$; $\aleph: F(M^i) \rightarrow \{0, 1\}$. Here,

   $$\aleph(f) = \begin{cases} 0; & \text{if f is a T-face} \\ 1; & \text{otherwise} \end{cases}$$

3. Let $\prod: F(M^i) \rightarrow F(M^{i-1})$ be the parent relation ship of a face. That is $f' = \prod(f)$ if and only if f in M^i has been generated by splitting f'. The following condition must hold:

   $$\forall f_i \ \forall f_j = ((\aleph(f_i) = 0) \wedge (\aleph(f_j) = 0) \wedge \text{neighbor}(f_i, f_j) \rightarrow (\Im(f_i) \neq \Im(f_j)) \vee (\prod(f_i) = \prod(f_j)))$$

These constrains are the prime directives of the developed ALS method. They enforce the regularity of the scheme and ensures that no thin triangle is introduced. Their application leads to fine results shown in Figure 4 as well as numerous other advantages. ALS scheme was subsequently successfully used not only for terrain visualization, but in geographical information systems, motion planning and computer simulation applications.

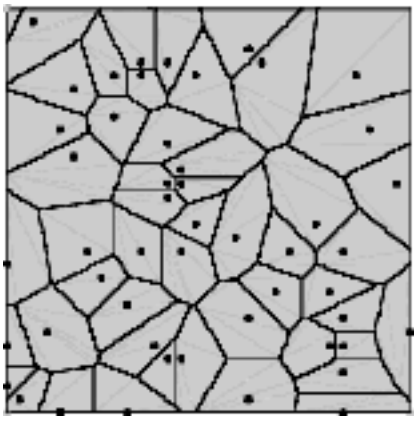
ADAPTIVE METHODOLOGY IN TRADITIONAL APPLICATIONS

Adaptive Geometric Methods

At the same time as adaptive methods were making their way in the area of terrain rendering, the renewed interest to topology-based data structures, Voronoi diagram and Delaunay triangulation in particular, has grown significantly (Okabe et.al. 1992). The key developments on both conceptual and implementation level are regularly presented at the International Symposium on Voronoi Diagrams in Science and Engineering Conference Series. Utilization of these developments in molecular modeling, bioinformatics and robotics promotes further stimulus to research on adaptive and dynamic problems. Thus, article (Gold and Dakowicz 2006) studies dynamic ship navigation visualization system using kinetic Voronoi diagram as an underlying concept. Utilization of the dynamic Voronoi diagram for 3D robot planning and navigation is studies in (Kolingerova 2005). Adaptive Voronoi diagram based approach to swarm simulation is presented in (Apu and Gavrilova 2006). These developments lead the way toward utilization of adaptive hierarchical models in computational geometry.

Recently, some preliminary results on utilization of computational geometry techniques in biometrics began to appear, such as research on image processing using Voronoi diagrams (Asano 2006, Liang and Asano_2004), work on utilizing Voronoi diagram for fingerprint synthesis (Bebis 1999), and studies on 3D modeling of human faces using triangular mesh (Li and Jain 2005). Some interesting results were recently obtained

Figure 5. Planar Voronoi diagram representation



in the BTLab, University of Calgary, through the development of topology-based feature extraction algorithms for fingerprint matching, 3D facial expression modeling and iris synthesis (Wang et. al. 2005, Wecker et. al. 2006, Bhattachariya and Gavrilova 2006).

Adaptive Image Processing and Visualization

Adaptive image processing is one of the most important techniques in visual information processing, especially in image restoration, filtering, enhancement, and segmentation. While existing literature presents some important aspects of the issue, there were no works that would treat the problem from a viewpoint that is directly linked to human perception – until the book "Adaptive Image Processing: A computational Intelligence Perspective Book" appeared (William et. al. 2001). This comprehensive collection of references treats adaptive image processing from a computational intelligence viewpoint, relating neural networks, fuzzy logic, and evolutionary computation to adaptive image processing. Based on the fundamentals of human perception, this book also gives a detailed account of computational intelligence methods and algorithms for adaptive image processing in regularization, edge detection, vision and any area where intelligent visual information processing is required.

Adaptive processing has been tightly linked not only to image processing, but to computer graphics and scientific visualization. In (Lopse et.al. 2002), authors provide an algorithm for computing a *robust adaptive polygonal approximation* of an implicit curve in the plane. The approximation is adapted to the geometry of the curve because the length of the edges varies with the curvature of the curve. Robustness is achieved by combining interval arithmetic and automatic differentiation.

Another example is adaptive visualization area of research, which is often explored in connection with geographical, urban or medical applications. A variety of engineering disciplines use large high-resolution geometric models whose computational requirements exceed current computer hardware capacities. The research presented in (Lu and Hammersley 2000) describes an *adaptive visualization* solution for interactively building such models. While adaptive visualization techniques have conventionally been applied to existing complete models, their method provides adaptive visualization of models while still under construction, through a clever utilization of multiresolution methods.

A common application of adaptive paradigm can be found in the area of medical imaging. A common active contour (*snake*) model is a popular choice for medical imaging applications, however the article by (Shen and Davatzikos 2000) goes further in exploring adaptive paradigm in this context. They propose a clever approach to geometric design and the structure of the model through *adaptive method for carrying out model deformations* in the most reliable way. Specifically, authors suggest to use an attribute vector to characterize the geometric structure around each point of the snake model, which allows to deform to nearby edges with considering geometric structure. They also provide an adaptive-focus statistical model which allows the deformation of the active contour in each stage to be influenced by the most reliable matches. Finally, they propose the deformation mechanism that is robust to local minima and is based on evaluating the snake energy function on segments of the snake at a time, instead of individual points. The approach is novel and unique, and is proven to perform very well experimentally.

Adaptive Information Systems

The current fast evolution in the areas of software, hardware and networks suggests that it will be possible to offer access to information systems through variety of interactive means, including computers, notebooks, PDA's, cellular phones, game consoles, GPS devices etc. Thus, this variety of technologies and information available required creation of a flexible environment to support adaptive interaction and services according to changing requirements, interfaces, devices, communication and user needs.

One of the research initiatives that started in 2002 was devoted specifically to this problem. The *Multichannel Adaptive Information Systems* project (MAIS) was funded by Basic Research Funds of the Italian Department of Education and involved six Universities and industry collaborations. The project research areas were information systems, database systems, human computer interaction, computer networks and telecommunication, hardware design, middleware, management engineering, with the focus on adaptive computing paradigm. Adaptive e-services, adaptive portable devices and adaptive networks were at the focus of the research. The prototype applications of the methods were developed in the areas of tourism, education, and risk management in archeology.

Another relevant project is described in (Doerr 1999). Authors state that one way to increase software system adaptability is to allocate resources dynamically at run-time rather than statically at design time. For example, fine-grained run-time allocation of processor utilization and network bandwidth creates an opportunity to execute multi-modal operations. Thus, this allocation strategy enhances adaptability by combining deterministic and non-deterministic functionality. Authors next showcase that adaptability is essential to improve versatility and decrease lifecycle maintenance costs for embedded real-time systems.

Adaptive Networks

As already seen from the above applications, the driving force behind the need for adaptive paradigm is variety of situations, variability in backgrounds and needs of different user groups or applications. The further expansion of Internet communications, not only for electronic exchange of ideas, but also as a means of collecting a wide range of information, has lead to a variety of other applications besides e-learning. Thus, electronic commerce, Internet transactions, collaborative newsgroups, facebooks, on-line teleconferencing are all rapidly developing. However, one of the problems is the difference in information available to network users. Network systems and services are becoming increasingly complex and diverse, and the processes involved in accessing required information are growing ever more advanced.

As a result, the information disparity that arises from the presence or absence of knowledge about networks and computers is becoming a problem that cannot be overlooked. Another problem is one that is derived from changes and increases in communications traffic.

The key to dealing with the above challenges is *user adaptability* and *adaptability to changes in communication demand*. User adaptability means that the user does not conform to the conditions of the network, but rather the network adapts instantaneously to the user environment and to service needs, which change with every passing moment. Adaptability to changes in communication demand means that the network configuration and equipment functions change dynamically to absorb the introduction of new services and macro-fluctuations in traffic. NTT Laboratories is one of the organizations that combines knowledge, expertise and application research in the area of *adaptive networks*, which constantly change their functions and configurations to respond immediately to changes in the environment, and to be the network platform of the Information Sharing Society of the future. Their key terms to describe the research in adaptive networks is Intellect, Evolution, and Simple & Seamless.

ADAPTIVE METHODS IN EMERGING AREAS

Biometrics and Adaptive Computing

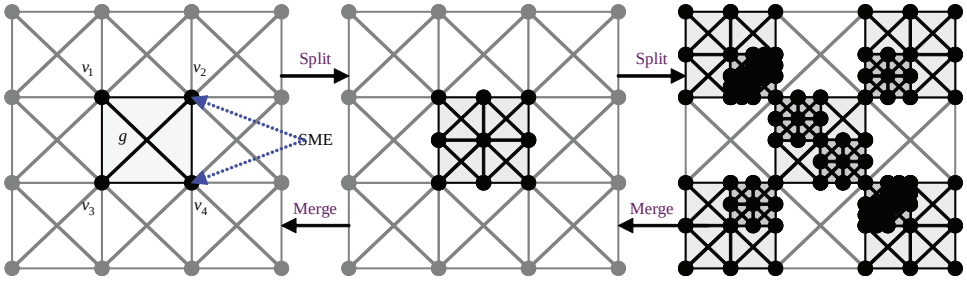
Adaptive techniques have made their way in emerging scientific areas such as *biometric computing*. In information technology, biometric refers to a study of physical and behavioral characteristics with the purpose of person identification. In recent years, the area of biometrics has witnessed a tremendous growth, partly as a result of a pressing need for increased security, and partly as a response to the new technological advances that are literally changing the way we live. Availability of much more affordable storage and the high resolution image capturing devices have contributed to accumulating very large datasets of biometric data. On the other hand, it also created significant challenges driven by the higher than ever volumes and the complexity of the data, that can no longer be resolved through acquisition of more memory, faster processors or optimization of existing algorithms. This justifies the need for the development of a new concept for biometric data storage and visualization based on adaptive paradigm.

It is obvious to anyone who works in the area of biometric computing that the problem is not trivial. It is not enough to simply fill the existing deficiency in data representation and visualization through application of advanced results from the areas of computational geometry and computer graphics. The backbone of the methodology is in the application of adaptive hierarchical data representation to achieve flexible and versatile data representation, fast data retrieval, reliable matching, easy updates and smooth and continuous data processing.

To achieve this objective, we suggest a novel way to represent complex biometric data (e.g. a bitmap, a graphics file, a set of vectors, a polygonal curve) through the organization of the data in a hierarchical tree-like structure. Such organization is similar to *Adaptive Memory Subdivision (AMS)* representation (see Figure 6).

AMS is a hybrid method based on the combination of traditional hierarchical tree structure with the concept of expanding or collapsing tree nodes, depending on the amount of information and level of detail (LOD) that needs to be represented. *Spatial quad-tree* is used to hold the information about the system, as well as the instructions on how to process this information. Expansion is realized through the spatial subdivision technique that refines the data and increases LOD, and collapsing is realized through the merge operation that simplifies the data representation and makes

Figure 6. Adaptive Memory Subdivision (AMS) model



(a) Base Patch LOD=0 Shaded Region is a primitive cluster

(b) After first split, LOD=1

(c) Subsequent refinements of ASM primitive clusters

it more compact. The greedy strategy is used to optimally adapt to the best representation based on the user requirements, amount of available data and resources, and required LOD. This powerful technique enables to achieve compact data representation with required LOD. For instance, it enables to efficiently store and retrieve minor details of the facial image (e.g. scars, wrinkles) or detailed patterns of the compared irises.

Adaptive methods have been studied for increasing hand geometry reliability while new processing algorithms, such as symmetric real Gabor filters, have been used to decrease the computational cost involved in iris pattern recognition (Sanchez-Reillo et. Al. 1999). This work explores adapting these methods to small *embedded systems*, and proposes the design of new biometric systems, where the users template is stored in a portable storage media for added security. Such media could be used to store sensitive information, for instance related to user's health records, and proposed adaptive access methods are devised to avoid the reading of this data unless the biometric verification has been performed.

While the above research is aimed at increase reliability and security of biometric data, another highly interesting direction is merging *adaptive paradigm with multimodal biometric fusion*. Multimodal biometric is intended to utilize biometric information obtained by multiple sensors and from multiple sources in order to increase authentication reliability. The paper by (Veeramachaneni 2003) introduces a new Adaptive Multimodal Biometric Fusion Algorithm(AMBF) algorithm, which is a combination of *Bayesian decision fusion* technique and a *particle swarm optimization* method. A Bayesian framework is typically used to fuse decisions received from multiple biometric sensors. The optimal rule is a function of the error cost and a priori probability of an intruder. This Bayesian framework formalizes the design of a system that can adaptively increase or reduce the security level. Particle swarm optimization searches the decision and sensor operating points (i.e. thresholds) space to achieve the desired security level. The optimization function aims to minimize the cost in a Bayesian decision fusion. The particle swarm optimization algorithm results in the fusion rule and the operating points of sensors at which the system can work. The swarm algorithm can easily handle the scalability issue as the number of sensors increases and efficiently search through the highly large fusion rule search space.

The presented method successfully merges two highly interesting and important paradigms: adaptive method based on evolutionary computing and multimodal biometric system with Beyesian decision rule. As a result, it can successfully address the varying security needs and user access

requirements in a biometric system. The authors report that the adaptive algorithm allows to achieve desired security level and to seemingly switch between different rules and sensor operating points for varying user needs.

The adaptive approach can be successfully sued not only for biometric modeling but also in synthesis of the biometric data. A combination of the method with *multi-resolution approach*, suitable for extracting details of the model at different scales of resolution, is a promising new direction of research which can be used to complement the missing information or to recreate the model. The method performs the high-level detail extraction and capturing of the model characteristics, and then applies this information for synthesis of new biometric data. This novel approach has shown a high potential in a recent study on iris synthesis (Wecker et.al. 2006). Moreover, *adaptive learning method can assist in* examining extracted features, retrieval patterns, and dynamic updates with the purpose of making the model more flexible.

Adaptive Methods in Robotics

Another highly important and rapidly developing area of research, that encompasses artificial intelligence, engineering, vision, geometric processing and decision-making, is robotics. *Adaptive robots* is an area of research that studies the design of robots that function in a changing environment by using high-level cognitive abilities and/or adaptive behaviors. The Dutch AIBO team, composed by research groups from the DECIS Lab and Universities of Amsterdam, Delft, Twente and Utrecht, is the leading force behind research on collaborative robot behavior and intelligent behavior of the team of robots, through robot soccer simulation and applied studies (Wong et. al. 2001). Exploring adaptive paradigm in a variety of areas essential to success of this project, including *adaptive robot vision, adaptive navigation and adaptive learning* are just some of the examples of the current state-of-the art research related to this project. Adaptive methods to improve *self-localization* in robot soccer were devised in (Dahm and Ziegler 2003). The authors utilized adaptive strategies to improve the reliability and performance of self-localization in robot soccer with legged robots. Adaptiveness is the common feature of the presented algorithms and has proved essential to enhance the quality of localization by a new classification technique as well as to increase the confidence level of information about the environment. Cooperative strategy based on *adaptive Q-learning* for robot soccer was developed in (Hwang et.al. 2004). The strategy developed enabled robots to cooperate with each other to achieve the objectives of offense and defense. Through the mechanism of learning, the robots learned from experiences in either successes or failures, and utilized these experiences to improve the performance. The cooperative strategy is based on a hierarchical architecture. An adaptive Q-learning method showed to allow more flexibility in learning that the traditional Q-learning approach, especially in the context of cooperative strategy.

Adaptive Knowledge Representation and Learning

Adaptive Knowledge Representation and Reasoning Conference (AKRR) is one of the unique events devoted completely to the emerging paradigm of *adaptive knowledge representation*. The forum is concerned with all adaptive aspects related to knowledge representation and reasoning. Specifically, such areas as adaptive systems in economic sciences and organizational theory, new generation of semantic web, adaptive systems in medical education, research and practice, and adaptive machine translation are within the conference scope.

The idea of utilizing *adaptive methods in e-learning* is not new. One of the first articles on the subject that appeared in 1998 describes web-based educational applications that are

expected to be used by very different groups of users (Brusilovsky 1998). Thus, authors argue that such a system needs to adapt to users with very different backgrounds, prior knowledge of the subject and learning goals, without human assistance. They next describe an approach for developing *adaptive electronic textbooks* and present InterBook—an authoring tool based on this approach which simplifies the development of adaptive electronic textbooks on the Web.

An extensive research in the area of adaptive learning was undertaken since that time. One example is the research on knowledge representation in the area of learning design and adaptive learning, presented in (Kravcik and Gasevic 2006). The authors deal with learning design and adaptation, and state that the procedural knowledge is highly important. They examine the degree of reusability and interoperability of procedural knowledge in the current adaptive educational hypermedia systems, and discuss several useful strategies and techniques, including informal scripts, system encoding, elicited knowledge, and standardized specifications.

Hierarchical Models in Cognitive Informatics

As it has been seen in the previous sections, adaptive computing heavily relies on hierarchical models of knowledge representing various natural and artificial phenomena. Such models can represent, for instance, complex three-dimensional terrain patterns, evolutionary behaviour of a swarm of living organisms, or an intricate structure of a web-linked virtual library. As it was recently discovered at the front line of the cognitive science and cognitive informatics research, hierarchical cognitive models can be also efficiently use to study internal information and knowledge presentation in human brains (Wang 2007).

It is commonly accepted that memory is the foundation of all forms of natural intelligence. Neural Informatics (NeI) is a branch of cognitive informatics, where memory is recognized as the foundation and platform of any natural or artificial intelligence (Wang 2003). While traditionally the Long-Term Memory (LTM) is perceived as static and fixed in adult brains, recent discoveries in neuroscience and cognitive informatics indicate that LTM is dynamically reconfiguring, particularly at the lower levels of the neural clusters (Wang 2007). The article explains the memory establishment, enhancement, and evolution, which are typical functions of the brain, are not limited to only childhood developmental stage; thus the more complex, dynamic model is necessary to represent such functions. In order to achieve this task, the pioneering theory is presented in the article that is based on the concept of the Hierarchical Neural Cluster (HNC) Model of Memory. Furthermore, the Object-Attribute-Relation (OAR) model is introduced to formally represent the structures of internal information and knowledge acquired and learned in the brain (Wang 2007). The OAR model explains the mechanisms of internal knowledge and information representation, as well as their physical and physiological meanings, and allows to better understand learning mechanisms and help to develop more powerful algorithms for a variety of complex problems faced by scientists everyday.

CONCLUSION

The article presented a comprehensive survey of the new paradigm of the algorithmic models of intelligence, based on the adaptive hierarchical model of computation. It started with the adaptive methods for terrain modeling, and presented methodically the adaptive paradigm in geometric computing, biometric, robotics, image processing and vision, knowledge representations, information systems, e-learning and networks. Illustrations were further provided to explain some concepts related to adaptive information processing and models. Aside from the areas

covered in the survey, there constantly appear a variety of new and emerging applications utilizing adaptive models of computations. It is our hope that this survey inspires readers to further in-depth study of the exciting topic of adaptive computations.

REFERENCES

Apu, R. and Gavrilova, M.L. "Adaptive Spatial Memory Representation for Real-Time Motion Planning," 3IA'2005 Int. Conf. on Comp. Graphics and Artificial Intelligence, France pp.21-32, 2005

Apu, R. and Gavrilova, M.L. "Battle Swarm: An Evolutionary Approach to Complex Swarm Intelligence," 3IA Int. C. Comp. Graphics and AI, Eurographics, Limoges, France, pp. 139-150, 2006

Asano, T. "Aspect-Ratio Voronoi Diagram with Applications," ISVD 2006, IEEE, pp. 32-39, 2006

Bardis,G., Miaoulis, G. and Plemenos, D. "Learning User Preferences at the Declarative and Geometric Description Level," 3IA'2005, Limoges (France), May 11-12, 2005

Bebis G., Deaconu T. and Georiopoulous, M. "Fingerprint Identification using Delaunay Triangulation," ICIIS 99, Maryland, pp. 452-459, 1999

Bhattachariya, P. and Gavrilova, M.L.. "CRYSTAL - A new density-based fast and efficient clustering algorithm", IEEE-CS Press, ISVD 2006, pp. 102-111, Banff, AB, Canada, July 2006

Bonnefoi, P.F. and Plemenos, D. "Constraint satisfaction techniques for declarative scene modelling by hierarchical decomposition," 3IA'2000, Limoges (France), May 3-4, 2000

Brusilovsky, P., Eklund, J. and Schwarz, E., "Web-based education for all: a tool for development of adaptive courseware," Computer Networks and ISDN Systems, Volume 30, Issues 1-7, pp. 291-300, 1998

Capelli R., Maio, D. and Maltoni D. "Synthetic Fingerprint-Database Generation," ICPR 2002, Canada, vol 3, pp 369-376, 2002

Cohen-Or, D. and Levanoni, Y. "Temporal continuity of levels of detail in Delaunay triangulated terrain," Visualization'96, pp. 37-42. IEEE Press, 1996

Dahm, I. and Ziegler, J. "Adaptive Methods to Improve Self-localization in Robot Soccer," LNCS 2752, pps. 393-408, 2003

Doerr, B.S., Venturella, T., Jha, R., Gill, C.D. and Schmidt, D.C., "Adaptive Scheduling for Real-time, Embedded Information," in Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference, 10 pages, 1999

Duchaineauy, M. et. al., "ROAMing Terrain: Real-Time Optimally Adapting Meshes," IEEE Visualization '97, pp. 81-88, 1997

Franc, M. and Skala, V. "Fast Algorithm for Triangular Mesh Simplification Based on Vertex decimation," CCGM2002 Proceedings, Lecture Notes in Computer Science, Springer, 2002

Gavrilova, M.L. Computational Intelligence: A Geometry-Based Approach, Book, in series Studies in Computational Intelligence, Springer-Verlag, to appear in 2008

Gold, C. and Dakowicz, M. "Kinetic Voronoi/ Delaunay Drawing Tools", ISVD 2006, IEEE-CS, pp. 76-84, Banff, Canada, 2006

Hwang, K.S., Tan, S.W. and Chen, C.C. "Cooperative strategy based on adaptive Q-learning for robot soccer systems," IEEE Transactions on Fuzzy Systems Volume: 12, Issue: 4, pp. 569- 576, 2004

Iglesias, A. "Computer Graphics Techniques for Realistic Modeling, Rendering, and Animation of Water. Part I: 1980-88." Int. Conf. on Computational Science (2) 2002, 181-190, 2002

Kawaharada, H. and Sugihara, K. "Compression of Arbitrary Mesh Data Using Subdivision Surfaces," IMA Conference on the Mathematics of Surfaces 2003, 99-110, 2003

Kolingerova I. "Probabilistic Methods for Triangulated Models," 8th Int. Conference on Computer Graphics and Artificial Intelligence 3IA 2005, Limoges, France, 93-106, 2005

Kravcik, M. and Gasevic, D. "Knowledge Representation for Adaptive Learning Design," Proceedings of Adaptive Hypermedia. June, Dublin, Ireland, 11 pages, 2006

Li S., Liu X. and Wu E., "Feature-Based Visibility-Driven CLOD for Terrain," In Proc. Pacific Graphics 2003, pp 313-322, IEEE Press, 2003

Li, S. and Jain, A. Handbook of Face Recognition. Springer-Verlag 2005

Liang X.F. and Asano T. "A fast denoising method for binary fingerprint image," IASTED, Spain, pp. 309-313, 2004

Loop, C. T. Smooth Subdivision Surfaces Based on Triangles. Masters Thesis, University of Utah, Department of Mathematics. 1987

Lopes, H., Oliveira, J.B. and de Figueiredo, L.H. "Robust adaptive polygonal approximation of implicit curves," Computers & Graphics Volume 26, Issue 6, pp. 841-852, 2002.

Lu, H. and Hammersley, R. "Adaptive visualization for interactive geometric modeling in geoscience," the 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media 2000, p.1, Feb. 2000

Luo, Y. and Gavrilova, M.L.. "3D Facial model synthesis using Voronoi Approach," IEEE-CS proceedings, ISVD 2006, pp. 132-137, Banff, AB, Canada, July 2006

Medioni, G. and Waupotitsch, R. "Face recognition and modeling in 3D," IEEE Int. Workshop on Analysis and Modeling of Faces and Gestures, pp. 232-233, 2003

Moriguchi, M. and Sugihara, K. "A new initialization method for constructing centroidal Voronoi Tessellations on Surface Meshes," ISVD 2006, IEEE-CS Press, pp. 159-165, 2006

Okabe, A., Boots, B. and Sugihara, K. Spatial tessellation concepts and applications of Voronoi diagrams, Wiley & Sons, Chichester, England, 1992

Perry, S.W., Wong H.S. and Guan, L. Adaptive Image Processing: A computational Intelligence Perspective, CRC Press, 9 volumes, 272 pages, 2001

Sanchez-Reillo, R., Sanchez-Avila, C. and Gonzalez-Marcos, A. "Multiresolution Analysis and Geometric Measures for Biometric Identification Systems," Secure Networking Proceedings, LNCS, Volume 1740, p. 783, 1999

Shen, S. and Davatzikos, C. "An Adaptive-Focus Deformable Model Using Statistical and Geometric Information," IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 22 No. 8 pp. 906-913 August 2000

Veeramachaneni, K., Osadciw, L.A. and Varshney, P.K. "Adaptive Multimodal Biometric Fusion Algorithm Using Particle Swarm," SPIE, vol 5099, pp. 211, Orlando, Florida, April 21- 25, 2003

Wang, Y. "Cognitive informatics: A new transdisciplinary research field." Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy, 4(2), 115-127, 2003

Wang, Y. "The OAR Model of Neural Informatics for Internal Knowledge Representation in the Brain", Int'l Journal of Cognitive Informatics and

Natural Intelligence, 1(3), 66-77, July-September, 2007

Wang, H., Gavrilova, M.L., Luo, Y. and Rokne, J. "An Efficient Algorithm for Fingerprint Matching", ICPR 2006, Int. C. on Pattern Recognition, Hong Kong, IEEE-CS , 2006

Wecker, L., Samavati, F. and Gavrilova, M.L. "Iris Synthesis: A Multi-Resolution Approach," GRAPHITE 2005, ACM Press, in association with SIGGRAPH, pp. 121-125, 2005

Wong, C.C., Chou, M.F., Hwang, C.P., Tsai, C.H. and Shyu, S.R. "A method for obstacle avoidance and shooting action of the robot soccer," Robotics and Automation, proceedings ICRA. IEEE International Conference, pp 3778- 3782 vol.4 2001

Yanushkevich, S., Wang, P., Srihari, S. and Gavrilova, M.L. Image Pattern Recognition: Synthesis and Analysis in Biometrics, Book, World Scientific, 452 pages, 2006

This work was previously published in the International Journal of Software Science and Computational Intelligence, Vol. 1, Issue 1, edited by Y. Wang, pp. 87-99, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 8.6
Agile Software Methods: State-of-the-Art

Ernest Mnkandla
Monash University, South Africa

Barry Dwolatzky
University of Witwatersrand, South Africa

ABSTRACT

This chapter is aimed at comprehensively analyzing and defining agile methodologies of software development from a software quality assurance perspective. A unique way of analyzing agile methodologies to reveal the similarities that the authors of the methods never tell you is introduced. The chapter starts by defining agile methodologies from three perspectives: a theoretical definition, a functional definition, and a contextualized definition. Then an agile quality assurance perspective is presented starting from a brief review of some of the traditional understandings of quality assurance to the innovations that agility has added to the world of quality. The presented analysis approach opens a window into an understanding of the state-of-the-art in agile methodologies and quality, and what the future could have in store for software developers. An understanding of the analysis framework for objectively analyzing and comparing agile methodologies is illustrated by applying it to three specific agile methodologies.

INTRODUCTION

Agile software development methodologies have taken the concepts of software quality assurance further than simply meeting customer requirements, validation, and verification. Agility innovatively opens new horizons in the area of software quality assurance. A look at the agile manifesto (Agile Alliance, 2001) reveals that agile software development is not just about meeting customer requirements (because even process-driven methodologies do that), but it is about meeting the changing requirements right up to the level of product deployment. This chapter introduces a technique for analyzing agile methodologies in a way that reveals the fundamental similarities among the different agile processes.

As for now, there is a reasonable amount of literature that seeks to describe this relatively new set of methodologies that have certainly changed the way software development is done. Most of the existing work is from the authors of the methodologies and a few other practitioners. What lacks is

therefore a more balanced evaluation comparing what the original intents of the authors of agile methodologies were, to the actual things that have been done through agile methodologies over the last few years of their existence as a group, and the possible future applications.

While most of those who have applied agile methods in their software development projects have gained margins that are hard to ignore in the areas of product relevance (a result of embracing requirements instability) and quick delivery (a result of iterative incremental development), some have not joined this new fun way to develop software due to a lack of understanding the fundamental concepts underlying agile methodologies. Hence, this chapter intends to give the necessary understanding by comprehensively defining agile methodologies and revealing how agile methodologies have taken software quality assurance further than traditional approaches. The second concern resulted from more than three years of research into agile methodology practices where the author discovered that the individual agile methods such as extreme programming, scrum, and lean development etc. are not that different from each other. The apparent difference is because people from different computing backgrounds authored them and happen to view the real world differently. Hence, the differences are not as much as the authors would like us to believe. The evaluation technique introduced here will reveal the similarities in a novel way and address the adoption concerns of agile methodologies. This also reveals what quality in an agile context means.

CHAPTER OBJECTIVES

The objective of this chapter is to introduce you to the fundamentals of analyzing agile methodologies to reveal the bare bones of agile development. After reading this chapter, you will:

- Understand three approaches to the definition of agile methodologies (i.e., a theoretical definition, a functional definition, and a contextualized definition).
- Understand the state-of-the-art in agile methodologies.
- Understand the presented framework for objectively analyzing and comparing agile methodologies.
- Understand the meaning of software quality assurance in an agile context.

BACKGROUND

This section will start by defining agile methodologies based on what people say about agile methodologies, what people do with agile methodologies, and what agile methodologies have done to the broad area of software development.

DEFINING AGILE METHODOLOGIES

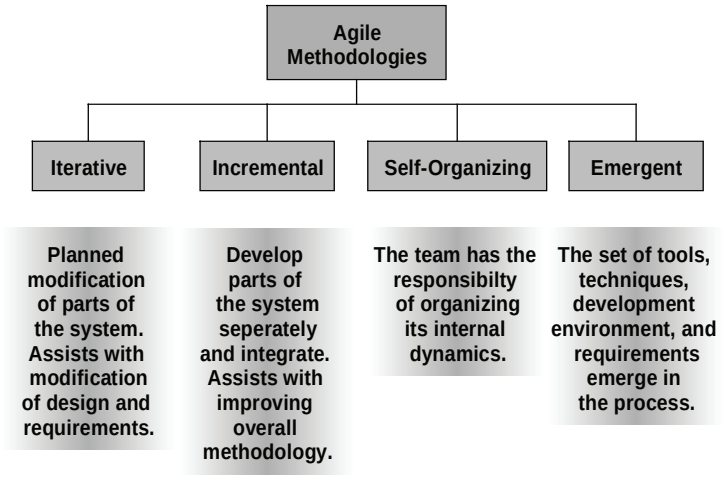
The agile software development methodologies group was given the name “agile” when a group of software development practitioners met and formed the Agile Alliance (an association of software development practitioners that was formed to formalize agile methodologies) in February 2001. The agile movement could mark the emergence of a new engineering discipline (Mnkandla & Dwolatzky, 2004a) that has shifted the values of the software development process from the mechanistic (i.e., driven by process and rules of science) to the organic (i.e., driven by softer issues of people and their interactions). This implies challenges of engineering complex software systems in work environments that are highly dynamic and unpredictable.

THEORETICAL DEFINITION

After the first eWorkshop on agile methodologies in June 2002, Lindvall et al. (2002) summarized the working definition of agile methodologies as a group of software development processes that are iterative, incremental, self-organizing, and emergent. The meaning of each term in the greater context of agility is shown next.

1. **Iterative:** The word iterative is derived from iteration which carries with it connotations of repetition. In the case of agile methodologies, it is not just repetition but also an attempt to solve a software problem by finding successive approximations to the solution starting from an initial minimal set of requirements. This means that the architect or analyst designs a full system at the very beginning and then changes the functionality of each subsystem with each new release as the requirements are updated for each attempt. This approach is in contrast to more traditional methods, which attempt to solve the problem in one shot. Iterative approaches are more relevant to today's software development problems that are characterized by high complexity and fast changing requirements. Linked with the concept of iterations is the notion of incremental development, which is defined in the next paragraph.
2. **Incremental:** Each subsystem is developed in such a way that it allows more requirements to be gathered and used to develop other subsystems based on previous ones. The approach is to partition the specified system into small subsystems by functionality and add a new functionality with each new release. Each release is a fully tested usable subsystem with limited functionality based on the implemented specifications. As the development progresses, the usable functionalities increase until a full system is realized.
3. **Self-organizing:** This term introduces a relatively foreign notion to the management of scientific processes. The usual approach is to organize teams according to skills and corresponding tasks and let them report to management in a hierarchical structure. In the agile development setup, the "self-organizing" concept gives the team autonomy to organize itself to best complete the work items. This means that the implementation of issues such as interactions within the team, team dynamics, working hours, progress meetings, progress reports etc. are left to the team to decide how best they can be done. Such an approach is rather eccentric to the way project managers are trained and it requires that the project managers change their management paradigm all together. This technique requires that the team members respect each other and behave professionally when it comes to what has been committed on paper. In other words management and the customer should not get excuses for failure to meet the commitment and there should be no unjustified requests for extensions. The role of the project manager in such a setup is to facilitate the smooth operation of the team by liaising with top management and removing obstacles where possible. The self-organizing approach therefore implies that there must be a good communication policy between project management and the development team.
4. **Emergent:** The word implies three things. Firstly, based on the incremental nature of the development approach the system is allowed to emerge from a series of increments. Secondly, based on the self-organizing nature a method of working emerges as the team works. Thirdly, as the system emerges and the method of working emerges a framework of development technologies will also emerge. The emergent nature of agile methodologies means that agile

Figure 1. Definition of agility © copyright Ernest Mnkandla PhD thesis University of the Witwatersrand



software development is in fact a learning experience for each project and will remain a learning experience because each project is treated differently by applying the iterative, incremental, self-organizing, and emergent techniques. Figure 1 sums up the theoretical definition of agile methodologies.

The value of agility is in allowing the concepts defined above to mutate within the parameters set by the agile values and principles (For details on agile values and principles see the agile manifesto at http://www.agilealliance.org.. There is always a temptation to fix a framework of software development if success is repeatedly achieved, but that would kill the innovation that comes with agile development.

FUNCTIONAL DEFINITION

Agile methodologies will now be defined according to the way some agile practitioners have understood them as they used them in real world practice.

The term "agile" carries with it connotations of flexibility, nimbleness, readiness for motion, activity, dexterity in motion, and adjustability (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). Each of these words will be explained further in the context of agility in order to give a more precise understanding of the kinds of things that are done in agile development.

- **Flexibility:** This word implies that the rules and processes in agile development can be easily bended to suit given situations without necessarily breaking them. In other words, the agile way of developing software allows for adaptability and variability.
- **Nimbleness:** This means that in agile software development there must be quick delivery of the product. This is usually done through the release of usable subsystems within a period ranging from one week to four weeks. This gives good spin-offs as the customer will start using the system before it is completed.
- **Readiness for motion:** In agile development, the general intention is to reduce

all activities and material that may either slow the speed of development or increase bureaucracy.

- **Activity:** This involves doing the actual writing of code as opposed to all the planning that sometimes takes most of the time in software development.
- **Dexterity in motion:** This means that there must be an abundance of skills in the activity of developing code. The skills referred to are the mental skills that will arm the developers for programming challenges and team dynamics.
- **Adjustability:** This is two fold; firstly there must be room for change in the set of activities and technologies that constitute an agile development process, secondly the requirements, code, and the design/architecture must be allowed to change to the advantage of the customer.

According to Beck (1999), agile methodologies are a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software. These terms will be defined in this context to give a functional perspective of agile development.

- **Lightweight** implies minimizing everything that has to be done in the development process (e.g., documentation, requirements, etc.) in order to increase the speed and efficiency in development. The idea of minimizing documentation is still a controversial one as some assume agility to mean no documentation at all. Such views are however not unfounded because some agile extremists have expressed connotations of zero documentation claiming that the code is sufficient documentation. As agile methodologies approach higher levels of maturity minimizing documentation has evolved to generally imply providing as much documentation as the customer is willing pay for in terms of time and money.
- **Efficient** means doing only that work that will deliver the desired product with as little overhead as practically possible.
- **Low-risk** implies trading on the practical lines and leaving the unknown until it is known. In actual fact, all software development methodologies are designed to reduce the risks of project failure. At times, a lot of effort is wasted in speculative abstraction of the problem space in a bid to manage risk.
- **Predictable** implies that agile methodologies are based on what practitioners do all the time, in other words the world of ambiguity is reduced. This however does not mean that planning, designs, and architecture of software are predictable. It means that agility allows development of software in the most natural ways that trained developers can determine in advance based on special knowledge.
- **Scientific** means that the agile software development methodologies are based on sound and proven scientific principles. It nevertheless remains the responsibility of the academia to continue gathering empirical evidence on agile processes because most of the practitioners who authored agile methodologies seem to have little interest and time to carryout this kind of research.
- **Fun way** because at last developers are allowed to do what they like most (i.e., to spend most of their time writing good code that works). To the developers, agility provides a form of freedom to be creative and innovative without making the customer pay for it, instead the customer benefits from it.

Schuh (2004) defines agile development as a counter movement to 30 years of increasingly heavy-handed processes meant to refashion computer programming into software engineering, rendering it as manageable and predictable as any other engineering discipline.

On a practical perspective, agile methodologies emerged from a common discovery among practitioners that their practice had slowly drifted away from the traditional heavy document and process centered development approaches to more people-centered and less document-driven approaches (Boehm & Turner, 2004; Highsmith, 2002a; Fowler, 2002). There is a general misconception that there is no planning or there is little planning in agile processes. This is due to the fact that the agile manifesto lists as one of its four values the preference for responding to change over following a plan (Agile Alliance, 2001). In fact, planning in agile projects could be more precise than in traditional processes it is done rigorously for each increment and from a project planning perspective agile methodologies provide a risk mitigation approach where the most important principle of agile planning is feedback. Collins-Cope (2002) lists the potential risks as: risks of misunderstandings in functional requirements, risks of a deeply flawed architecture; risks of an unacceptable user interface; risks of wrong analysis and design models; risks of the team not understanding the chosen technology et cetera. Feedback is obtained by creating a working version of the system at regular intervals or per increment according to the earlier planning effort (Collins-Cope, 2002).

Besides dealing with the most pertinent risks of software development through incremental development, agile methodologies attack the premise that plans, designs, architectures, and requirements are predictable and can therefore be stabilized. Agile methodologies also attack the premise that processes are repeatable (Highsmith, 2001; Schwaber & Beedle, 2002). These two premises are part of fundamental principles on which traditional methodologies are built, and they also happen to be the main limitations of the traditional methodologies.

Boehm et al. (2004) view agile methodologies as a challenge to the mainstream software development community that presents a counter-culture movement, which addresses change from a radically different perspective. All agile methodologies follow the four values and 12 principles as outlined in the agile manifesto.

CONTEXTUAL DEFINITION

From these definitions of agile methodologies, a contextual definition can be derived which looks at what agility means in terms of certain specific software engineering concepts. Examples of that would be concepts are software quality assurance, software process improvement, software process modeling, and software project management. Agile methodologies will now be defined according to these concepts. Since this book is specifically focused on agile software quality assurance the definition of agile software quality assurance will be given in more detail.

AGILE SOFTWARE QUALITY ASSURANCE

This section starts by summarizing the traditional definitions of quality and then presents a summary of the work that has been done in the area of agility and quality. References to older literature on software quality are not intended to be exhaustive, but to be simply present a fare baseline for evaluating software quality perspectives in the modern processes. The authors are aware of a number of initiatives in research and academic institutions where evaluation of quality concepts is performed on some agile practices.

DEFINING QUALITY

Have you ever wondered what Joseph Juran generally considered to be a quality legend would have said about agile processes and the quality movement? Well, this is what he said about the

ISO 9000 when he was asked by Quality Digest if he thought ISO 9000 had actually hindered the quality movement; "Of course it has. Instead of going after improvement at a revolutionary rate, people were stampeded into going after ISO 9000, and they locked themselves into a mediocre standard. A lot of damage was, and is, being done" (QCI International, 2002).

According to Juran, quality is fitness for use, which means the following two things: "(1) quality consists of those product features that meet the needs of the customers and thereby provide product satisfaction. (2) Quality consists of freedom from deficiencies" (Juran & Gryna, 1988).

Philip Crosby, who developed and taught concepts of quality management, whose influence can be found in the ISO 9000:2000 standard, which differs from the 1994 standard in the context of each of the eight principles, defines quality as conformance to requirements and zero defects (Crosby, 1984).

ISO 9000 defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs. Where "stated needs" means those needs that are specified as requirements by the customer in a contract, and 'implied needs' are those needs that are identified and defined by the company providing the product. These definitions of quality have a general bias towards the manufacturing industry although they should in general apply to all products, nevertheless, software products are rather complex hence they should be defined in a slightly different way.

Weinberg defines quality simply as "the value to some people" (Weinberg, 1991) and some have expanded on that to mean the association of quality with human assessment, and cost and benefit (Hendrickson, 2004).

Some software engineers have defined software quality as follows:

1. Meyer (2000) defines software quality according to an adapted number of quality parameters as defined by McCall (1977), which are correctness, robustness, extendibility, reusability, compatibility, efficiency, portability, integrity, verifiability, and ease of use.
2. Pressman, who derives his definition from Crosby, defines quality as a "conformance to explicitly stated functional requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software" (Pressman, 2001).
3. Sommerville (2004) defines software quality as a management process concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability, and so on.
4. van Vliet (2003) follows the IEEE definition of quality as stated in the IEEE Glossary of Software Engineering Terminology, which defines quality assurance in two ways as: "(1) A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established operational, functional, and technical requirements. (2) A set of activities designed to evaluate the process by which products are developed or manufactured" (IEEE, 1990). van Vliet's perspective then combines this definition with the analysis of the different taxonomies on quality.
5. Pfleeger (2001) aligns her perspective with Garvin's quality perspective, which views quality from five different perspectives namely; the transcendental meaning that quality can be recognized but not defined, user view meaning that quality is fitness for purpose, manufacturing meaning that quality is conformance to specification, product view meaning that quality is tied to inherent product characteristics, and

the value-based view meaning that quality depends on the amount the customer is willing to pay for the product.

6. Bass (2006) argues that the common practice of defining software quality by dividing it into the ISO 9126 (i.e., functionality, reliability usability, efficiency maintainability, and portability) does not work. His argument is that "in order to use a taxonomy, a specific requirement must be put into a category" (Bass, 2006). However, there are some requirements that may be difficulty to put under any category, for example, "denial of service attack, response time for user request, etc." What Bass (2006) then proposes is the use of quality attributing general scenarios.

From an agile perspective, quality has been defined by some practitioners as follows:

Table 1. Agile quality techniques as applied in extreme programming

Technique	**Description**
Refactoring	Make small changes to code, Code behaviour must not be affected, Resulting code is of higher quality (Ambler, 2005).
Test-driven development	Create a test, Run the test, Make changes until the test passes (Ambler, 2005).
Acceptance testing	Quality assurance test done on a finished system, Usually involves the users, sponsors, customer, etc. (Huo, Verner, Zhu, & Babar, 2004).
Continuous integration	Done on a daily basis after developing a number of user stories. Implemented requirements are integrated and tested to verify them. This is an important quality feature.
Pair programming	Two developers work together in turns on one PC, Bugs are identified as they occur, Hence the product is of a higher quality (Huo et al., 2004).
Face-to-face communication	Preferred way of exchanging information about a project as opposed to use of telephone, email, etc. Implemented in the form of daily stand-up meetings of not more than twenty minutes (Huo et al, 2004). This is similar to the daily Scrum in the Scrum method. It brings accountability to the work in progress, which vital for quality assurance.
On-site customer	A customer who is a member of the development team, Responsible for clarifying requirements (Huo et al., 2004).
Frequent customer feedback	Each time there is a release the customer gives feedback on the system, and result is to improve the system to be more relevant to needs of the customer (Huo et al., 2004). Quality is in fact meeting customer requirements.
System metaphor	Simple story of how the system works (Huo et al., 2004), Simplifies the discussion about the system between customer/ stakeholder/ user and the developer into a non-technical format. Simplicity is key to quality.

McBreen (2003) defines agile quality assurance as the development of software that can respond to change, as the customer requires it to change. This implies that the frequent delivery of tested, working, and customer-approved software at the end of each iteration is an important aspect of agile quality assurance.

Ambler (2005) considers agile quality to be a result of practices such as effective collaborative work, incremental development, and iterative development as implemented through techniques such as refactoring, test-driven development, modelling, and effective communication techniques.

To conclude this section, Table 1 gives a summary of the parameters that define agile quality as specifically applied in extreme programming--a popularly used agile methodology. These aspects of agile quality have eliminated the need for heavy documentation that is prescribed in traditional processes as a requirement for quality. Quality is a rather abstract concept that is difficult to define but where it exists, it can be recognized. In view of Garvin's quality perspective there may be some who have used agile methodologies in their software development practices and seen improvement in quality of the software product but could still find it difficult to define quality in the agile world.

EVALUATING QUALITY IN AGILE PROCESSES

So can we evaluate quality assurance in agile processes? This can be done through:

- The provision of detailed knowledge about specific quality issues of the agile processes.
- Identification of innovative ways to improve agile quality.
- Identification of specific agile quality techniques for particular agile methodologies.

Literature shows that Huo et al. (2004) developed a comparison technique whose aim was to provide a comparative analysis between quality in the waterfall development model (as a representative of the traditional camp) and quality in the agile group of methodologies. The results of the analysis showed that there is indeed quality assurance in agile development, but it is achieved in a different way from the traditional processes. The limitations of Huo et al.'s tool however, are that the analysis:

- Singles out two main aspects of quality management namely quality assurance and verification and validation.
- Overlooks other vital techniques used in agile processes to achieve higher quality management.
- Agile quality assurance takes quality issues a step beyond the traditional software quality assurance approaches.

Another challenge of Huo et al.'s technique is that while the main purpose of that analysis was to show that there is quality assurance in agile processes, it does not make it clear what the way forward is. Agile proponents do not seem to be worried about comparison between agile and traditional processes as some of the more zealous "agilists" believe that there is no way traditional methods can match agile methods in any situation (Tom Poppendieck, personal e-mail 2005).

The evaluation described in this section improves on (Huo et al., 2004) framework by further identifying some more agile quality techniques and then in an innovative way identifies the agile process practices that correspond to each technique. The contribution of this evaluation is the further identification of possible practices that can

Table 2. Mapping software quality parameters to agile techniques

Software Quality Parameters	**Agile Techniques**	**Possible Improvements**
Correctness	Write code from minimal requirements. Specification is obtained by direct communication with the customer. Customer is allowed to change specification. Test-driven development.	Consider the possibility of using formal specification in agile development, Possible use of general scenarios to define requirements (note that some development teams are already using this).
Robustness	Not directly addressed in agile development.	Include possible extreme conditions in requirements.
Extendibility	A general feature of all OO developed applications. Emphasis is on technical excellence and good design. Emphasis also on achieving best architecture.	Use of modeling techniques for software architecture.
Reusability	A general feature of all OO developed applications. There are some arguments against reusability of agile products (Turk, France, & Rumpe, 2002; Weisert, 2002).	Develop patterns for agile applications.
Compatibility	A general feature of all OO developed applications.	Can extra features be added for the sake of compatibility even if they may not be needed? This could contradict the principle of simplicity.
Efficiency	Apply good coding standards.	Encourage designs based on the most efficient algorithms
Portability	Practice of continuous integration in extreme programming.	Some agile methods do not directly address issues of product deployment. Solving this could be to the advantage of agility.
Timeliness	Strongest point of agility, Short cycles, quick delivery, etc.	
Integrity	Not directly addressed in agile development.	
Verifiability	Test-driven development is another strength of agility.	
Ease of use	Since the customer is part of the team, and customers give feedback frequently, they will most likely recommend a system that is easy to use.	Design for the least qualified user in the organization.

be done to improve on the already high quality achievements enjoyed by agile processes.

TECHNIQUES

The parameters that define software quality from a top-level view can be rather abstract. However, the proposed technique picks each of the parameters and identifies the corresponding agile techniques that implement the parameter in one way or another. Possible improvements to the current practice have been proposed by analysing the way agile practitioners work. Of great importance to this kind of analysis is a review of some of the intuitive practices that developers usually apply which may not be documented. You may wonder how much objectivity can be in such information. The point though is that developers tell their success stories at different professional forums and some of the hints from such deliberations have been captured in this technique without following any formal data gathering methodology. The authors believe that gathering of informal raw data balances the facts especially in cases where developers talk about their practice. Once the data is gathered formally, then a lot of prejudices and biases come in and there will be need to apply other research techniques to balance the facts. Tables 2 and 3 summarize the evaluation approach.

In formal software quality management, quality assurance activities are fulfilled by ensuring that each of the parameters listed in Table 2 are met to a certain extent in the software development life cycle of the process concerned. A brief definition of each of these parameters is given according to Meyer (2000):

- **Correctness:** The ability of a system to perform according to defined specification.
- **Robustness:** Appropriate performance of a system under extreme conditions. This is complementary to correctness.
- **Extendibility:** A system that is easy to adapt to new specification.
- **Reusability:** Software that is composed of elements that can be used to construct different applications.
- **Compatibility:** Software that is composed of elements that can easily combine with other elements.
- **Efficiency:** The ability of a system to place as few demands as possible to hardware resources, such as memory, bandwidth used in communication and processor time.
- **Portability:** The ease of installing the software product on different hardware and software platforms.
- **Timeliness:** Releasing the software before or exactly when it is needed by the users.
- **Integrity:** How well the software protects its programs and data against unauthorized access.
- **Verifiability:** How easy it is to test the system.
- **Ease of use:** The ease with which people of various backgrounds can learn and use the software.

SOFTWARE PROCESS IMPROVEMENT

A bigger-picture view of agile processes leads to a notion that agile methods are a group of processes that have reduced the development timeframe of software systems and introduced innovative techniques for embracing rapidly changing business requirements. With time, these relatively new techniques should develop into mature software engineering standards.

SOFTWARE PROCESS MODELING

The agile perspective to software process modeling is that whether formal or informal when

approaches to modeling are used the idea is to apply modeling techniques in such a way that documentation is minimized and simplicity of the desired system is a virtue. Modeling the agile way has led to breakthroughs in the application of agile methods to the development of large systems (Ambler, 2002)

SOFTWARE PROJECT MANAGEMENT

The agile approach to managing software projects is based on giving more value to the developers than to the process. This means that management should strive to make the development environment conducive. Instead of worrying about critical path calculation and Gantt chart schedules, the project manager must facilitate face-to-face communication, and simpler ways of getting feedback about the progress of the project. In agile development there is need to be optimistic about people and assume that they mean good hence give them space to work out the best way to accomplish their tasks. It is also an agile strategy to trust that people will make correct professional decisions about their work and to ensure that the customer is represented in the team throughout the project.

THE AGILE METHODOLOGY EVALUATION FRAMEWORK

All agile methodologies have striking similarities amongst their processes because they are based on the four agile values and 12 principles. It is interesting to note that even the authors of agile methodologies no longer emphasize their methodology boundaries and would use practices from other agile methodologies as long they suit a given situation (Beck & Andres, 2004). In fact, Kent Beck in his extreme programming (XP) master classes frequently mentions the errors of extremism in the first edition of his book on XP (Beck, 1999). A detailed review of agile methodologies reveals that agile processes address the same issues using different real life models.

The evaluation technique presented in this chapter reveals, for example, that lean development (LD) views software development using a manufacturing and product development metaphor. Scrum views software development processes using a control engineering metaphor. Extreme programming views software development activities as a social activity where developers sit together. Adaptive systems development (ASD) views software development projects from the perspective of the theory of complex self-adaptive systems (Mnkandla, 2006).

Tables 3 to 6 summarize the analysis of agile methodologies. Only a few of the existing agile methodologies have been selected to illustrate the evaluation technique. The first column from the left on Tables 3, 4, and 5 lists some methodology elements that have been chosen to represent the details of a methodology. There is a lot of subjectivity surrounding the choice of methodology elements. It is not within the scope of this chapter to present a complete taxonomy of methodologies. For more detailed taxonomies see Avison and Fitzgerald (2003), Boehm et al. (2004), Glass and Vessey (1995), and Mnkandla (2006). Therefore, the elements used here were chosen to reveal the similarities amongst different agile methodologies. The importance of revealing these similarities is to arm the developers caught up in the agile methodology jungle wondering which methodology to choose. While the methodology used in your software development project may not directly lead to the success of a project and may not result in the production of a high quality product use of a wrong methodology will lead to project failure. Hence, there is in wisdom selecting a correct and relevant process. Most organization may not afford the luxury of using different methodologies for each project though that would be ideal for greater achievements. It

also sounds impractical to have a workforce that is proficient in many methodologies. Sticking to one methodology and expect it to be sufficient for all projects would also be naïve (Cockburn, 2000). This evaluation technique therefore gives software development organizations an innovative wit to tailor their development process according to the common practices among different agile methodologies. The advantage is to use many methodologies without the associated expenses of acquiring them.

There is a need to understand in detail each agile methodology that will be analyzed so as to reveal the underlying principles of the methodology. This technique gives the background details as to why the methodology was developed in the first place. An answer to this question would reveal the fundamental areas of concern of the methodology and what fears the methodology addresses. The prospective user of the methodology would then decide whether such concern area is relevant to their project. Identifying what problems the methodology intends to solve is another concern of this evaluation. Some methodologies have a general bias toward solving technical problems within the development process (i.e., extreme programming deals with issues such as how and when to test the code). There are other agile methodologies that solve project management problems (i.e., Scrum deals with issues such as how to effectively communicate within a project). Yet other agile methodologies solve general agile philosophy problems (i.e., Crystal deals with issues such as the size of the methodology vs. the size of the team and the criticality of the project. There may be other agile methodologies that solve a mix of problems right across the different categories mentioned here for example Catalyst puts some project management aspects into XP (see www.ccpace.com for details on Catalyst).

Evaluation of each methodology should also reveal what sort of activities and practices are prevalent in the methodology. This should assist prospective users of the methodology to determine the practices that could be relevant to their given situation. This evaluation technique reveals that some of the practices from different methodologies actually fulfill the same agile principles and it would be up to the developers to decide which practices are feasible in their situation. Therefore, the implication is that at the level of implementation it becomes irrelevant which agile methodology is used, for more on this concept see Mnkandla (2006). Another aspect of agile methodologies revealed by this evaluation technique is what the methodology delivers at the end of the project. When a developer looks for a methodology, they usually have certain expectations about what they want as an output from the methodology. Hence, if the methodology's output is not clearly understood problems may result. For example if the developer expects use of the methodology to lead to the delivery of code and yet the aim of the methodology is in fact to produce a set of design artifacts such as those delivered by agile modeling this could lead to some problems. Finally, this evaluation technique also reveals the domain knowledge of the author of the methodology. In this phase of analysis, there is no need to mention any names of the authors but simply to state their domain expertise. The benefit of revealing the background of the methodology author is to clarify the practical bias of the methodology, which is usually based on the experience, and possible fears of the methodology's author.

Tables 3 to 5 give a summary of the analysis of specific agile methodologies to illustrate how this analysis technique can be used for any given agile methodologies.

ANALYZING SCRUM

Scrum has been in use for a relatively longer period than other agile methodologies. Scrum, along with XP, is one of the more widely used agile methodologies. Scrum's focus is on the fact that "defined and repeatable processes only

Table 3. Analyzing scrum methodology

Elements	Description
Real Life Metaphor	Control engineering.
Focus	Management of the development process.
Scope	Teams of less than 10, but is scalable to larger teams.
Process	Phase 1: planning, product backlog, & design. Phase 2: sprint backlog, sprint. Phase 3: system testing, integration, documentation, and release.
Outputs	Working system.
Techniques	Sprint, scrum backlogging (writing use cases).
Methodology Author (two)	1. Software developer, product manager, and industry consultant. 2. Developed mobile applications on an open technology platform. Component technology developer. Architect of advanced internet workflow systems.

work for tackling defined and repeatable problems with defined and repeatable people in defined and repeatable environments" (Fowler, 2000), which is obviously not possible. To solve the problem of defined and repeatable processes, Scrum divides a project into iterations (which are called Sprints) of 30 days. Before a Sprint begins, the functionality required is defined for that Sprint and the team is left to deliver it. The point is to stabilize the requirements during the Sprint. Scrum emphasizes project management concepts (Mnkandla & Dwolatzky, 2004b) though some may argue that Scrum is as technical as XP. The term Scrum is borrowed from Rugby: "A Scrum occurs when players from each team clump closely together…in an attempt to advance down the playing field" (Highsmith, 2002b). Table 3 shows application of the analysis technique to Scrum.

ANALYZING LEAN DEVELOPMENT

Lean software development like dynamic systems development method and Scrum is more a set of project management practices than a definite process. It was developed by Bob Charette and it draws on the success that lean manufacturing gained in the automotive industry in the 1980s. While other agile methodologies look to change the development process, Charette believes that to be truly agile, there is need to change how companies work from the top down (Mnkandla et al., 2004b). Lean development is targeted at changing the way CEOs consider change with regards to management of projects. LD is based on lean thinking whose origins are found in lean production started by Toyota Automotive manufacturing company (Poppendeick & Poppendeick, 2003). Table 4 shows application of the analysis technique to LD.

ANALYZING EXTREME PROGRAMMING

Extreme programming (XP) is a lightweight methodology for small-to-medium-sized teams developing software based on vague or rapidly changing requirements (Beck, 1999). In the second version of XP, Beck extended the definition of XP

Table 4. Analyzing lean development methodology

Elements	Description
Real Life Metaphor	Manufacturing and product development.
Focus	Change management. Project management.
Scope	No specific team size.
Process	Has no process.
Outputs	Provides knowledge for managing projects.
Techniques and Tools	Lean manufacturing techniques.
Methodology Author	Research engineer at the US Naval Underwater Systems Center, author of software engineering books and papers, advisory board in project management.

to include team size and software constraints as follows (Beck et al., 2004):

- **XP is lightweight:** You only do what you need to do to create value for the customer.
- **XP adapts to vague and rapidly changing requirements:** Experience has shown that XP can be successfully used even for project with stable requirements.
- **XP addresses software development constraints:** It does not directly deal with project portfolio management, project financial issues, operations, marketing, or sales.
- **XP can work with teams of any size:** There is empirical evidence that XP can scale to large teams.

Software development using XP starts from the creation of stories by the customer to describe the functionality of the software. These stories are small units of functionality taking about a week or two to code and test. Programmers provide estimates for the stories, the customer decides, based on value and cost, which stories to do first. Development is done iteratively and incrementally. Each two weeks, the programming team delivers working stories to the customer. Then the customer chooses another two weeks worth of work. The system grows in functionality, piece by piece, steered by the customer. Table 5 shows application of the analysis technique to XP.

A WALK THROUGH THE ANALYSIS TECHNIQUE

Each of the methodology elements as represented in Tables 3 to 5 will be defined in the context of this analysis approach.

METHODOLOGY'S REAL LIFE METAPHOR

This element refers to the fundamental model/metaphor and circumstances that sparked the initial idea of the methodology. For example watching the process followed by ants to build an anthill could spark an idea of applying the same process to software development.

METHODOLOGY FOCUS

The focus of the methodology refers to the specific aspects of the software development process targeted by the methodology. For example, agile

Table 5. Analyzing extreme programming

Elements	Description
Real Life Metaphor	Social activity where developers sit together.
Focus	Technical aspects of software development.
Scope	Less than ten developers in a room. Scalable to larger teams.
Process	**Phase 1:** Writing user stories. **Phase 2:** Effort estimation, story prioritization. **Phase 3:** Coding, testing, integration testing. **Phase 4:** Small release. **Phase 5:** Updated release. **Phase 6:** Final release (Abrahamsson et al, 2002).
Outputs	Working system.
Techniques and Tools	Pair programming, refactoring, test-driven development, continuous integration, system metaphor.
Methodology Authors (two)	1. Software developer (Smalltalk). Strong believer of communication, reflection, and innovation. Pattern for software. Test-first development. 2. Software developer (Smalltalk). Director of research and development. Developed the Wiki. Developed Framework for Integrated Test (Fit).

modeling targets the design aspects of the software development process and also considers issues of how to model large and complex projects the agile way.

METHODOLOGY SCOPE

This element outlines the details to which the methodology's development framework is spelled out. This is where the methodology specifies what it covers within a project. The importance of this parameter is to help the user to identify the list of tasks that the methodology will help manage. Remember a methodology does not do everything but simply gives guidelines that help in the management of a project. The scope of a software development project is relevant in determining the size of the team.

METHODOLOGY PROCESS

This parameter describes how the methodology models reality. The model may be reflected in the life cycle or development process of the methodology. The model provides a means of communications, captures the essence of a problem or a design, and gives insight into the problem area (Avison et al., 2003). The importance of this parameter is that it gives the user a real worldview of the series of activities that are carried out in the development process.

METHODOLOGY OUTPUTS

This parameter defines the form of deliverables to be expected from the methodology. For example if an organization purchased lean development

methodology today, would they get code from application of the methodology, or would they get some documents, etc (Avison et al., 2003). Each agile methodology will give different outputs hence the user can choose the methodology that gives them the output they require.

TECHNIQUES AND TOOLS

This parameter helps the user to identify the techniques and tools applicable to the methodology. Tools may be software applications that can be used to automate some tasks in the development process, or they can be as simple as whiteboards and flip charts. In fact, it is the use of tools that makes the implementation of a methodology enjoyable. Organizations therefore tend to spend a lot of money acquiring tools and training staff on tools. As technology evolves and new tools emerge, more acquisitions and training are usually done. However, most agile methodologies do not specify tools and most agile practitioners use open source tools, which reduces potential costs on software tools.

Each methodology has its own techniques that may be relevant or irrelevant to the problem at hand. Examples of techniques in extreme programming would be pair programming, and the scrum meeting in Scrum methodology. The user then analyzes these techniques in relation to the present project, to determine need for the techniques and include variations that will be part of tailoring the methodology.

METHODOLOGY AUTHOR

This parameter defines the domain knowledge of the methodology author. The benefit of doing this is to clarify the background from which the methodology was conceived. There is no need to mention the name of the author or a detailed biography of the methodology author.

Table 6 summarizes the phase of the analysis where all the practices are brought together and similar practices are identified across different methodologies.

Table 6 classifies the practices using the superscripts 1, 2, 3, 4, and 5. The practices with the same superscript implement the same agile principle.

- "1" represents practices that deal with planning issues such as requirements gathering. The three methods shown here use different

Table 6. Identifying similarities among the practices

	Practices
XP	The planning process[1], small releases[2], metaphor, test-driven development[2], story prioritization[3], collective ownership[3], pair programming[3], forty-hour work week[3], on-site customer[4], refactoring[5], simple design[5], and continuous integration[5].
LD	Eliminate waste[1], minimize inventory[1], maximize flow[2], pull from demand[2], meet customer requirements[2], ban local optimization[2], empower workers[3], do it right the first time[4], partner with suppliers[4], and create a culture of continuous improvement[5].
Scrum	Capture requirements as a product backlog[1], thirty-day Sprint with no changes during a Sprint[2], Scrum meeting[3], self-organizing teams[3], and Sprint planning meeting[4].

terms but the principle is to capture minimal requirements in the simplest available way and start coding.
- "2" represents practices that deal with improvement of quality in terms of meeting the volatile requirements.
- "3" represents practices that facilitate freely working together of developers, effective communication, empowered decision-making, and team dynamics issues.
- "4" represents practices that deal with quick delivery of the product.
- "5" represents practices that deal with agile quality assurance property of ensuring that the product is improved continuously until deployment.

When the similar practices are identified, the developers can then decide to select and tailor some practices to their environment according to relevance, and project and customer priorities. You will notice that the choice of the activities of the development process according to this analysis have shifted from focusing on the individual methodologies to a focus on practices.

ISSUES AND CONTROVERSIES SURROUNDING AGILE DEVELOPMENT

Software Development Common Ground

This section looks at issues that are done in a similar way among different software development methodologies. Most software development processes in use today involve some of the following activities: planning, estimation, and scheduling of the tasks, design, coding, and testing, and deployment and maintenance. What varies among the different processes is the sequence followed in implementing each of the phases, and the level of detail to which each phase is carried out. Some methodologies may implement all of the activities and some partial methodologies may specialize in just a few. The other difference is in the way the process values the people involved in the development activities and what value is attached to the customer in relation to what needs to be done. These differences mark the major boundaries among software development methodologies.

Agile Development Higher Ground

This section looks at issues that are done in a peculiar way by agile methodologies. The role of the domain expert in agile methodologies is rather unique. Software development experts with practical experience in this field have a lot of knowledge that can be classified as *tacit knowledge* due to the fact that it is gained through practice and is not written down in any form. Tacit knowledge is difficult to quantify hence this concept remains quite subjective in the implementation of agile methodologies. However, the strength of using tacit knowledge rests in the team spirit that puts trust on experts to do what they know best within their professional ethics. This in fact is what differentiates the "agile movement" from other development processes. Another hot issue about agile development is the concept of *self-organizing* teams. This concept means that agile development teams are allowed to organize themselves the best way they want in order to achieve the given goals. As a result of applying this concept, managing agile projects becomes different from the traditional approaches to project management. The role of a project manager becomes more of a facilitator than a controller. Detailed discussions on what has become known as "agile project management" can be found in Highsmith (2004) and Schwaber (2004) and at http://finance.groups.yahoo.com/group/agileprojectmanagement/.

Agile methodologies also emphasize on *light documentation*. This concept has been quite controversial since agile methodologies started. The main reason for the controversy is that the

traditional methodologies have always associated documentation with proper planning, software quality assurance, deployment, user training, maintenance, etc. Agile methodologists however, believe that documentation should be minimum because of the associated expenses. In agile methodologies, the general belief is that correctly written code is sufficient for maintenance. The *Test first* technique, which was originally an XP practice and is now widely applied in other agile methodologies is another peculiar agile practice (though its origins may be from earlier processes). The test first technique is a software development approach that implements software design through writing tests for each story before the code is written. The test code then amounts to design artifacts and replaces the need for design diagrams etc.

Challenges Faced by Agile Development

This section looks at issues that are still grey areas to agile methodologies. One of the interesting issues about agility is what is going to happen to the issues of innovative thinking embedded in agile development as the processes attain higher and higher levels of maturity and quality assurance. Are we going to see a situation where agility retires and fails to be agile? Another area of software development that is always heavily debated at agile gatherings is the area of how to cost projects that are developed the agile way. The main difficulty is estimating the cost of an entire project based on iterations. There has been some effort towards dealing with this challenge especial at big agile conferences, for example the extreme programming and agile processes in software engineering held in Europe once per year (see www.XP2006.org). Another example is the Agile Development Conference also held once per year in the USA (see www. agiledevelopmentconference.com).

As agile processes begin to enter grounds such as enterprise architecture, patterns, and software reuse, their software process jacket is getting heavier and heavier and if this is not watched by agile proponents we might have a situation sometime in the future where another software development revolution emerges to maintain the legacy of agility.

THE FUTURE TRENDS OF AGILE SOFTWARE DEVELOPMENT

Agile methodologies are certainly moving toward higher levels of maturity due to a number of things. The first contribution to agile maturity is the availability of comprehensive sources of simple descriptive and analytical information about agile methodologies. The second contribution to agile maturity is the growth in academic research interest in agility, which has resulted in a lot of empirical data being collected and scientifically analyzed to prove and disprove anecdotal data about agile processes. The third contribution to agile maturity is the massive exchange of practical experiences amongst the different practitioners involved in agile software development. The general direction of the agile movement seems to be towards more and more demand for the adoption of agile practices by the larger organizations that have been traditionally associated with traditional processes. There have been reports of higher demands for agile consultancy and training as more and more organizations adopt agile development practices, Poppendieck (personal communication, November 07, 2005) said there was more demand in North America, Europe, and even Japan where his book on lean software development sold more than ten thousand copies. Another interesting development by the agile alliance is their offer to sponsor agile research. This will certainly go a long way in boosting the process maturity of agile methodologies.

CONCLUSION

In this chapter, an overview of agile methodologies was presented without going into the details of describing each existing agile methodology. The focus of the chapter was to provide an informed review of agile methodologies that included a comprehensive definition of what agility and agile quality assurance is all about. The approach to the definition as presented in this chapter was to give a theoretical definition, which is the perspective of those who are philosophical about agile methodologies, a practical definition, which is the perspective of those who are on the development work floors, and a contextual definition, which is a perspective based on the different contexts of the activities of the software development process. In order to enhance understanding of the agile processes, an analysis model was presented. The philosophy of this technique is to cut deep into each given agile methodology and reveal the core values, principles, and practices of the methodology so as to compare the common activities among different agile processes. The aim of doing such an analysis is to provide a technique for striking the balance between these two extremes: "getting lost in the agile methodology jungle and holding onto one methodology." The benefit of using this analysis method is the attainment of a deeper understanding of all the agile methodologies analyzed. This should lay the ground for training and adoption of agile methodologies from a generic point of view rather than worrying about individual agile methodologies.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis. VVT Publications,* (478), 7-94.

Agile Alliance. (2001). *Manifesto for agile software development.* Retrieved May 2, 2005, from http://www.agilemanifesto.org

Ambler, S. W. (2002). *Agile modeling.* John Wiley and Sons.

Ambler, S. W. (2003). *Agile database techniques: Effective strategies for the agile software developer* (pp. 3-18). John Wiley & Sons.

Ambler, S. (2005). *Quality in an agile world. Software Quality Professional, 7*(4), 34-40.

Avison, D. E., & Fitzgerald, G. (2003). *Information systems development: Methodologies techniques and tools.* McGraw-Hill.

Bass, L. (2006, January). Designing software architecture to achieve quality attribute requirements. *Proceedings of the 3rd IFIP Summer School on Software Technology and Engineering* (pp. 1-29). South Africa.

Beck, K. (1999). *Extreme programming explained: Embrace change* (pp. 10-70). Reading, MA: Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change.* Addison-Wesley Professional.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed* (1st ed., pp. 165-194, Appendix A). Addison-Wesley.

Brandt, I. (1983). A comparative study of information systems development methodologies, Proceedings of the IFIP WG8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies. In T. W. Olle, H. G. Sol, & C. J. Tully (Eds.), *Information systems design methodologies: A feature analysis* (pp. 9-36). Amsterdam: Elsevier.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software*, 64-71.

Cohen, D., Lindvall, M., & Costa, P. (2003). *Agile software development* (pp. 11-52). Fraunhofer Center for Experimental Software Engineering, Maryland, DACS SOAR 11 Draft Version.

Collins-Cope, M. (2002). *Planning to be agile? A discussion of how to plan agile, iterative, and incremental developments.* Ratio Technical Library White paper. Retrieved January 20 from http://www.ration.co.uk/whitepaper_12.pdf

Fowler, M. (2000). Put your process on a diet. *Software Development*, *8*(12), 32-36.

Fowler, M. (2002). *The agile manifesto: Where it came from and where it may go.* Martin Fowler article. Retrieved January 26, 2006, from http://martinfowler.com/articles/agileStory.html

Glass, R. L., & Vessey, I. (1995). Contemporary application domain taxonomies. *IEEE Software*, 63-76.

Hendrickson, E. (2004). *Redefining quality,* Retrieved January 12, 2006, from http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=7109

Highsmith, J. (2001). The great methodologies debate: Part 1: Today, a new debate rages: Agile software development vs. rigours software development. *Cutter IT Journal*, *14*(12), 2-4.

Highsmith, J. (2002a). *Agile software development: Why it is hot!* (pp. 1-22). Cutter Consortium white paper, Information Architects.

Highsmith, J. (2002b). *Agile software development ecosystems* (pp. 1-50). Addison-Wesley.

Highsmith, J. (2004). *Agile project management.* Addison-Wesley.

Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004). Software quality and agile methods. *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC04).* IEEE Computer.

IEEE. (1990). *IEEE standard glossary of software engineering terminology.* IEEE Std 610.12.

Juran, J. M., & Gryna, F. M. (1988). *Juran's quality control handbook.* Mcgraw-Hill.

Lindvall, M., Basili, V. R., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., & Zelkowitz, M. V. (2002). Empirical findings in agile methods. *Proceedings of Extreme Programming and agile Methods: XP/agile Universe* (pp. 197-207).

Marick, B. (2001). Agile methods and agile testing. *STQE Magazine*, *3*(5).

McBreen, P. (2003). *Quality assurance and testing in agile projects.* McBreen Consulting. Retrieved January 12, 2006, from http://www.mcbreen.ab.ca/talks/CAMUG.pdf

Meyer, B. (2000). *Object-oriented software construction* (pp. 4-20). Prentice Hall PTR.

Mnkandla, E., & Dwolatzky, B. (2004a). Balancing the human and the engineering factors in software development. *Proceedings of the IEEE AFRICON 2004 Conference* (pp. 1207-1210).

Mnkandla, E., & Dwolatzky, B. (2004b). A survey of agile methodologies. *Transactions of the South Africa Institute of Electrical Engineers*, *95*(4), 236-247.

Mnkandla, E. (2006). *A selection framework for agile methodology practices: A family of methodologies approach*. PhD thesis, University of the Witwatersrand, Johanneburg.

Pfleeger, S. L. (2001). *Software engineering: Theory and practice.* Prentice Hall.

Poppendeick, M., & Poppendeick, T. (2003). *Lean software development: An agile toolkit for software development managers* (pp. xxi-xxviii). Addison Wesley.

Pressman, R. S. (2001). *Software engineering a practitioner's approach*. McGraw-Hill.

QCI International. (2002). *Juran: A life of quality: An exclusive interview with a quality legend.* Quality Digest Magazine. Retrieved January 12, 2006, from http://www.qualitydigest.com/aug02/articles/01_article.shtml

Schuh, P. (2004). *Integrating agile development in the real world* (pp. 1-6). MA: Charles River Media.

Schwaber, K., & Beedle, M. (2002). *Agile software development with SCRUM* (pp. 23-30). Prentice-Hall.

Schwaber, K. (2004). *Agile project management with Scrum.* Microsoft Press.

Sol, H. G. (1983). A feature analysis of information systems design methodologies: Methodological considerations. Proceedings of the IFIP WG8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies. In T. W. Olle, H. G. Sol, & C. J. Tully (Eds.), *Information systems design methodologies: A feature analysis* (pp. 1-7). Amsterdam: Elsevier.

Sommerville, I. (2004). *Software engineering.* Addison-Wesley.

Turk, D., France, R., & Rumpe, B. (2002). Limitations of agile software processes. *Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering* (pp. 43-46).

van Vliet, H. (2003). *Software engineering: Principles and practice.* John Wiley & Sons.

Weinberg, G. M. (1991). *Quality software management* (Vol. 1), *Systems Thinking.* Dorset House.

Weisert, C. (2002). *The #1 serious flaw in extreme programming (XP).* Information Disciplines, Inc., Chicago. Retrieved January 2006, from http://www.idinews.com/Xtreme1.html

This work was previously published in Agile Software Development Quality Assurance, edited by I. Stamelos and P. Sfetsos, pp. 1-22, copyright 2007 by Information Science Reference (an imprint of IGI Global).

Chapter 8.7
Bridging the Gap between Agile and Free Software Approaches: The Impact of Sprinting

Paul J. Adams
Sirius Corporation Ltd., UK

Andrea Capiluppi
University of Lincoln, UK

ABSTRACT

Agile sprints are short events where a small team collocates in order to work on particular aspects of the overall project for a short period of time. Sprinting is a process that has been observed also in Free Software projects: these two paradigms, sharing common principles and values have shown several commonalities of practice. This article evaluates the impact of sprinting on a Free Software project through the analysis of code repository logs: sprints from two Free Software projects (Plone and KDE PIM) are assessed and two hypotheses are formulated: do sprints increase productivity? Are Free Software projects more productive after sprints compared with before? The primary contribution of this article is to show how sprinting creates a large increase in productivity both during the event, and immediately after the event itself: this argues for more in-depth studies focussing on the nature of sprinting.

INTRODUCTION

Agile and Free Software development have received rapid growth in popularity, both as development paradigms and as research topics. In theory they are very different concepts; the latter, strictly speaking, being just a licensing paradigm with implications for code reuse and redistribution.

The interface between Agile and Free Software is very interesting and a fertile area in which not much rigorous research has been carried out to date. Some comparative studies have been made in the past, but given the scarcity of data from

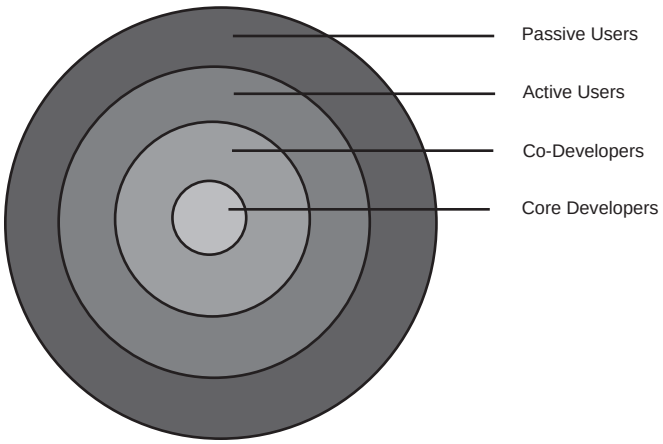
Agile processes, most of the studies have remained on the surface of theoretical discussions (Koch, 2004)(Warsta and Abrahamsson, 2003). Empirical attempts have been also made to measure, on an empirical basis, the degree of *agility* within other development paradigms (Adams, Capiluppi and deGroot, 2008).

This article examines and compares the Free Software and Agile approaches by observing typical Agile practices when deployed within Free Software teams: in particular, it reports on the Plone and KDE PIM projects, where sprinting (Beck, 1999) is commonly used by developers to focus the activity for a limited period of time. Sprinting allows developers to meet in person, get to know each other and create the basis for collaborating more effectively in a distributed environment (During, 2006). In previous works, the PyPy and the Zope projects have been reported to use sprinting regularly within their projects (Sigfridsson, Avram, Sheehan and Sullivan, 2007), and its use is advocated as an *"applied (...) idea of Agile development to the very difficult problem of distributed software development"* (Goth, 2007).

What past literature has not provided yet is a quantitative evaluation of the impact of sprinting on productivity of developers: what has instead been reported is that traditional productivity metrics could fail in capturing the effects of the interactions among developers within sprints (Goth, 2007). In order to tackle this issue, this article explores the use of automatic measures to determine the productivity of developers both before and after the sprinting efforts. A research hypothesis has been formulated as follows: when quantitatively evaluating sprinting, the productivity of developers will display higher values after a sprint than before it. If the null hypothesis can be rejected, this result could prove useful to others in the Free Software communities, encouraging them to adopt this practice and to focus their efforts within a constrained period of time to increase their productivity.

This article is structured as follows: Section 2 introduces the context of the work, explaining how the Agile and Free Software paradigms share some of their process characteristics. Section 3 reports on the methodology, the attributes and the definitions used throughout the article. Section 4 describes how sprinting is accomplished within the two reported case studies, while Section 5 summarises the main findings of measuring the effects of sprinting on developers productivity. Since this work reports on empirical analysis of public data, Section 6 will report on the threats to validity. Finally Section 7 will conclude the article, and illustrate avenues of further research.

Figure 1. The open development model



SPRINTS WITHIN THE AGILE DEVELOPMENT

As an activity, sprinting has its roots within the Agile development, specifically the SCRUM development model (Schwaber and Beedle, 2001). The SCRUM model is not a method per se, in that it does not prescribe specific practices to be followed in the release cycle. Within SCRUM, the principal period of development is focused during the sprints. Typically, a sprint would be a short period of development, typically 4 to 6 weeks (although they can be shorter). Within this period, developers would work to solve a specific and well-focused problem, such as the addition of a new set of functionality. Normally a sprinting team is collocated, with a dedicated manager who monitors progress on a daily basis in a short stand-up meeting.

As described, this model is in great contrast to the open development approach, as found within many Free Software projects. Again, the Free Software model does not prescribe specific practices, just how contributions should be handled. Originally Eric Raymond argued for what he called the "bazaar'' model of development (Raymond, 1999). In this model contributions to the code may be made by anyone whilst concurrently this openness leads to many people discovering and fixing bugs. In practice this is not what is found (Crowston, Annabi, Howison and Massango, 2004). Instead it is more common to have a model with a "cathedral'' component at its center, managed by core developers. This model is shown in Figure 1. It has been also shown that some Free Software projects show transitions between one stage to the other, and that this transition can be started by explicit actions of the core developers of a Free Software project (Capiluppi and Michlmayr, 2007).

At the centre of the model are the core developers, who are typically long-term contributors to a project and drive its development. They are the *de facto* management of Free Software developments. Other contributors to a project typically move in towards the centre of the model from the outside, where they began as users of the software.

Despite the commercial nature of the Agile approach, sprinting practices have been adopted by Free Software teams: given the differences as described above, and due to the unique nature of this model, Free Software developers have so far preferred to maintain the Free Software name instead of using the much more structured "Agile'' term to define their approach (Goth, 2007), even when blended with tools coming from other development approaches.

EMPIRICAL APPROACH

The following concepts and attributes have been used to extract data from the two Free Software projects, and used to compare the effectiveness of their sprinting practices:

- **Commit:** the atomic action of developers checking in one or more files (being source code or other) into a central repository. Both the Plone and the KDE PIM projects store the source code and other artifacts within similar Configuration Management Systems (subversion), therefore the same scripts were used to extract the relevant information out of each project.
- **Lines of code (LOCs):** in this study, the number of lines of code (LOCs) of each commit is also recorded. They will be used in conjunction with the commits to describe the output produced by Free Software developers.
- **Productivity:** in its most basic definition, it evaluates the amount of work produced during an observed period. Considering the information stored in configuration management systems, the productivity was in first instance evaluated by taking the amount of commits in a day. Secondly, the LOCs of the

commits were also extracted, to provide a clearer picture of the productivity of developers during sprints.

- **Sprint duration:** since their inception, sprints are considered one of the core practices within the SCRUM development model (Beck, 1999). Short periods of focused development work, preceded by a precise set of requirements both by stakeholders and final users, are at the core of the practice. Within this work, the sprints are characterised by evaluating the time (in days) when the sprints were held. The measurement of productivity was evaluated not only in each of the sprint periods, but also one week (*i.e.*, 7 days) before and one week after those events.

KDE PIM and Plone

The KDE website[1] hosts a large number of Free Software projects under a common name which together form both a desktop environment and associated application software, primarily for Unix-like operating systems. The complete project repository is roughly 50GB large and has grown steadily, along with contributors, since the project's launch over ten years ago. The KDE repository has somewhere in the order of 300 projects, ordered by application domain, one of which is Personal Information Management (PIM). KDE as a whole lists more than 1500 developers as committers in its Subversion CMS; considering the KDE PIM project alone, an overall of 380 committers are traceable in its evolution history.

Plone[2] is a Free Software content management system based upon the Zope web application server. Written in Python, it can run on all major platforms. As a project it is much smaller than KDE with 163 accounts in the project repository. Although Plone is focussed on one particular product, development is subdivided into teams working on specific issues within the system: css, programmed logic, work flow management, etc.

GQM: Goal Question Metric

The Goal-Question-Metric (GQM) method evaluates whether a goal has been reached, by associating that goal with questions that explain it from an operational point of view, and providing the basis for applying metrics to answer these questions (Basili, Caldiera and Rombach, 1994). The aim of the method is to determine the information and metrics needed to be able to draw conclusions on the achievement of the goal.

In the following, we applied the GQM method to first identify the overall goal of this research; we then formulate a number of hypotheses related to the two Free Software projects and their sprinting methods; and finally we collected adequate productivity metrics to determine whether the goal was achieved.

- **Goal:** The long-term objectives of this research are both to assess the presence, and to evaluate the efficacy of Agile processes within a Free Software development paradigm. If confirmed on these initial case studies, the results should be made public to other Free Software projects, in order to take advantage of Agile practices and tools.
- **Question:** The aim of this study is to establish the efficacy of an Agile practice (sprinting) within the Free Software paradigm. The productivity of the two Free Software systems will be studied: based on the "productivity'' and "sprint duration'' attributes defined above, two research hypotheses were formulated:

  HP1: *There is a difference between the average productivity of Free Software projects and the productivity as achieved during the sprints.*

  HP2: *In the presence of sprinting, there is a difference in the productivity of Free Software developers between the periods*

immediately before and immediately after the selected sprints.

- **Metrics:** based on the number of sprints actually performed during the lifecycle of the two projects, 6 sprints were selected from each project, and the productivity evaluated during the week before the sprint, during the sprint duration, and during the week after the sprint.

A summary of the research hypotheses is displayed in Table 1: the null and the alternative counterparts are formulated, as well as the metrics used to assess the hypothesis, and the type of test conducted to evaluate the hypothesis.

Sprinting in Free Software Projects

As one might expect, the implementation of sprinting varies between different Free Software projects. Even when the activities are similar, the frequency and motivation can be very different. A common value for all the observed sprints in the Free Software approach has been the location of these sprints: being it a distributed development, the face-to-face meetings have been typically run alongside main Free Software conferences, or developers meet-ups (During, 2006). In this section the sprinting practices of Plone and KDE PIM are described: one of the authors has been involved in the development of the KDE PIM, while directly collaborating with several developers from the other.

Sprinting in Plone

Since early 2003 there have been 32 Plone project sprints. Typically these involve less than 10 developers. However, as sprints have become more regular, so has the number of sprinters increased with meetings growing to be as large as 120 registrants[3]. These larger meetings are a-typical in that they are not, in themselves, one sprint alone. Instead, they are a gathering of many sprint meetings. Here, this larger type of meeting is described based upon observation of a sprint held in October 2007.

Each team, within the larger event, works on an individual subcomponent of the entire system: work flow, cascading style sheets, ZODB, etc. These teams set their own daily targets and share them with the entire sprint team in a short meeting at the beginning of the day. A similar meeting is held at the end of the day in order to convey progress made. In general, these Plone sprints are focused on the rapid introduction of new functionality to the system and optimisation of existing functionality. Some Plone sprints have also had a shared focus on system documentation. Six sprints (out of 32) were randomly selected between 2003 and 2007, with sprint durations varying from 3 to 8 days. A summary of the sprints' dates (in ISO format) and their durations is displayed in Table 2.

Table 1. Summary of the research hypotheses and the applied tests

Hypo	Null	Alternative	Metrics	Test
HP1	Productivity during sprints is larger than the baseline productivity	Productivity is larger during sprints	Commits-per-day (C)	$C_{sprints} > C_{baseline}$
HP2	Productivity after the sprints is larger than productivity after the sprints	Productivity after the sprints is equal to before	Commits-per-day (C)	$C_{after} > C_{before}$

Sprints in KDE-PIM

The sprints of this project have been held annually since 2003. As with the Plone sprints they are generally larger than an Agile sprint, typically with around 15 developers. However, unlike Plone sprints, there is no structure (explicit or otherwise) to each day and the primary focus of the event is not on coding, although this is one of the activities[4].

The annual KDE PIM plays a specific role within the product release cycle. KDE PIM development, like KDE as a whole, is based upon a 6 month release cycle for major releases. Minor releases for bug fixing and small amounts of new functionality happen monthly. The KDE PIM sprints occur at the beginning of one of these release cycles and is primarily focused with planning the next release.

In order to be consistent with the Plone case, six sprints were selected between 2003 and 2008, with reported shorter sprint durations, varying from 3 to 4 days. This corresponds to all of the sprints happened within the KDE PIM project: details of these sprint dates are in Table 2.

Sprint Impact: Setting Base Rates

The impact that a particular event may have on the development cycle is inevitably hard to assess without insider knowledge of what may be happening within that project at that time. There are, however, simple metrics that can be applied to public data sources in order to produce results which are indicative of the impact of an event (*i.e.*, sprinting) without needing to know specifically what has occurred. As reported above, the Configuration Management servers of the two studied projects have been analysed to extract the amount of commits recorded in the development activities. The number of commits per day was then used as a means of assessing productivity before and after development sprints, as well as within the sprint period.

As a start, a simple means of establishing any trends in commit rates is to plot the commit rates for the entire project history. Example for KDE PIM and Plone development are given in Figure 2 and Figure 3.

Figure 2 shows the distribution of commits-per-day in the Plone project. From this distribution,

Table 2. Summary of the dates of the sprints in Plone and KDE PIM

Project / Event		Start Date	End Date	Duration (days)
Plone	Sprint 1	2003-05-09	2003-05-11	3
	Sprint 2	2004-09-16	2004-09-19	4
	Sprint 3	2005-02-20	2005-02-27	8
	Sprint 4	2005-03-24	2005-03-27	4
	Sprint 5	2006-04-23	2006-04-29	7
	Sprint 6	2007-02-17	2007-02-20	4
KDE PIM	Sprint 1	2003-01-03	2003-01-05	3
	Sprint 2	2004-01-02	2004-01-05	4
	Sprint 3	2005-01-06	2005-01-09	4
	Sprint 4	2006-01-06	2006-01-08	3
	Sprint 5	2007-01-12	2007-01-15	4
	Sprint 6	2008-02-01	2008-02-03	3

it is also possible to evaluate an overall average commit rate of 10 commits per day, and a median value is 6. Since the distribution of Figure 2 is clearly not uniform, using the global average and median as base rates could produce spurious results. It was therefore decided to use different base rates of productivity for the various sprints: the base rates were in fact computed only during the year when the sprint happened, and are summarised in Table 3.

Similar to the Plone plot of commits, Figure 3 shows that largely due to an increase in activity between late 2003 and early 2006, KDE PIM has an average commit rate of 10 (31,214 commits assessed over 2,853 days), with a median value of 8. As there have only ever been 6 KDE PIM sprints, they were all assessed in the same manner as the Plone sprints. The dates are shown in Table 2. As seen above for the Plone system, different base rates were used to assess whether the effect of sprinting could be visible on the productivity of developers (Table 3).

Productivity During the Sprints

In order to tackle the first research hypothesis (HP1), an approach to measuring the impact of sprints upon project development is to compare the commit rate during the sprints to the base rate established in the previous section. In the

Figure 2. Growth of commits-per-day in plone

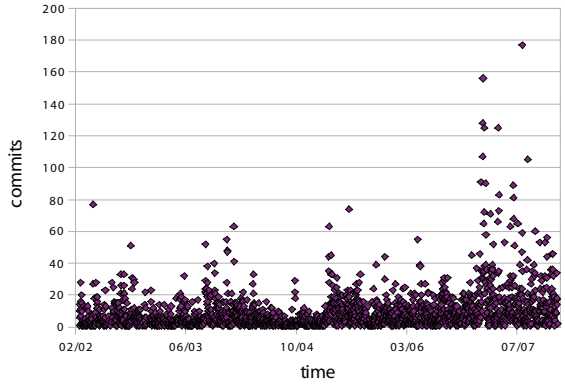


Figure 3. Growth of commits-per-day in KDE PIM

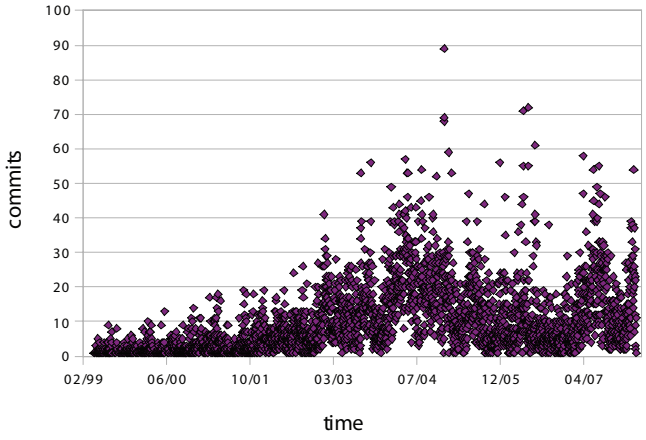


Table 3. Different base rates of commits-per-day for the two projects

	Plone			KDE PIM		
Base Rates	Year	Median	Average	Year	Median	Average
Sprint 1	2003	4	5,97	2003	12	12,9
Sprint 2	2004	4	4,75	2004	20	20,57
Sprint 3	2005	6	7,97	2005	11	13,26
Sprint 4	2005	6	7,97	2006	7	9,67
Sprint 5	2006	7	8,32	2007	12	16,42
Sprint 6	2007	12	19,1	2008	15	20,37

next subsections, the productivities of the two selected projects are evaluated during the six sprints, and compared to the base rates of the appropriate year. The test of the hypothesis HP1 is based on the principle of majority: if the majority of sprints achieve a larger productivity with respect to the yearly base rate, the null hypothesis will be rejected.

Effects of Sprinting – Plone

Figure 4 shows the commits per day for the 6 randomly selected sprints and the context weeks before and after: a vertical line divides the context week before the sprint from the sprint itself. Since the sprints have different durations, no common vertical line is displayed between the end of the sprint and the context week after it. As an example, the data for sprint 4 in Figure 4 displays 18 points: 7 points for the week before the sprint, 4 for the sprint duration, and 7 for the week after.

For each sprint, the average productivity and the median were recorded during each sprint duration, and compared with the base rates of the productivity within Plone. A summary of these results is shown in Table 4. As visible, in general the productivity during the six sprints in Plone has not increased, when compared with the base rates (depicted in bold within the Table), apart from two events where the values are greatly larger. One of the possible explanations is that Plone sprints have a large amount of participants turning up for the meetings, and coordination problems could arise which could make the productivity decrease.

The amount of lines of code in a day (i.e., LOCs-per-day) was also evaluated during the sprints of Plone, in order to test whether the commits-per-day and the LOCs-per-day trends followed the same pattern. Table 5 summarises the results: the Plone case shows that the average LOCs-per-day metrics follow the same pattern also seen in commits-per-day: in the presence of more commits, the Plone system experienced an increased productivity in LOCs, as compared to the week immediately after the sprint.

Effects of Sprinting: KDE PIM

Shows the commits per day for the 6 randomly selected sprints and the context weeks before and after. As done above, the productivity of KDE PIM developers was recorded during the sprints, and evaluated against the base rates (Table 4). As visible in the table, the majority of the sprints in KDE PIM achieve a larger productivity (either as medians, averages, or both) than the base counterparts. Differently from the Plone case, KDE PIM sprints tend to gather the same developers to work on the selected requirements, and this could be reflected on the improved productivity during these events. KDE PIM sprints are generally starting with a commit rate lower than that of the project base rate. However a crucial difference between the KDE PIM and Plone sprints is that KDE PIM

sprints are not only more consistent at improving productivity, but they are also successfully raising the commit rate above the base.

As done for the Plone case, the amount of LOCs-per-day in KDE PIM was evaluated during the sprints, as compared to the week immediately after the sprint, and the results reported in Table 5. The KDE PIM case shows that the LOCs-per-day metric follows the same pattern of commits-per-day: the three sprints where data of LOCs was available (sprint 4, 5 and 6) achieved a higher rate of LOCs-per-day than the weeks immediately after them. Sprints 1, 2 and 3, due to the recent conversion to Subversion, have no data available regarding the LOCs involved in the commits.

Productivity after the Sprints

The study of the first research hypothesis above showed a mixed picture: the development of Plone does not achieve (overall) an increased productivity during sprints with respect to its base rates of effort. Instead, the KDE PIM productivity during sprints largely differs from its average, showing a very focused development process during certain periods of the project's lifecycle.

The second research hypothesis was designed in order to assess how the process of development reacted after the sprinting event, and to ascertain whether the productivity before was impacted by the event itself, showing a clear increase after the event. For each sprint the average commits per day, and their medians, were noted for the week immediately before and immediately after the sprint. As a base line for this hypothesis, one would expect an increased productivity after the event, when compared with the period just before it. The results of this analysis are shown in Table 6.

Productivity after the Sprints: Plone

Comparing the base rate medians and averages with the sprints' achievements, Table 6 shows that the productivity has not steadily increased after the event, as compared with the period before the event: sprint 1, 2 and 3 clearly show that the overall activity, in terms of commits per day, not only dropped after the sprints, compared to just before; but it also decreased to lower levels than the overall base rates for the Plone project itself.

By analysing the remaining sprints, two additional findings were instead revealed: first, the high productivity accomplished during the sprints 4 and 6, was not paired to a similar productivity after the sprint effort (see Table 4). Second, in either the median or average, or both the measurements, an increase of productivity is detected after the event. An explanation for that was given by Plone insiders, revealing that sprints were more and more effective, as long as the use of this practice became clearer and more diffused. In summary, these results are indicative of sprints improving the productivity of the project as a whole, but the null hypothesis of HP2 could not be rejected for Plone: in the majority of cases, the productivity of developers is not larger after sprints as compared to before the sprints.

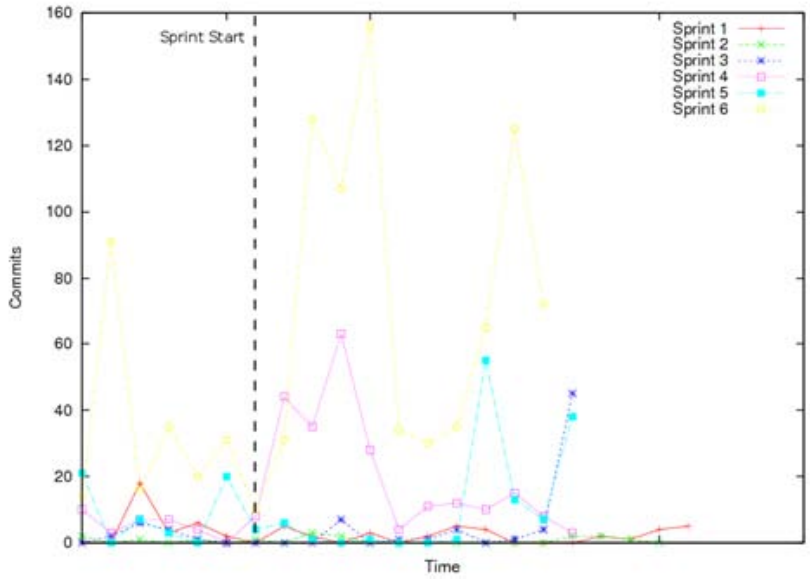
Productivity after the Sprints: KDE PIM

Table 6 confirms for KDE PIM what was already learned from the analysis of the sprint effort. All the studied sprints experienced an increase of either the median or the average productivity, or both. In the majority of the sprints, differently from the Plone case, the week after these sprints also accomplished a higher productivity than the observed base rates for the project itself.

An interpretation for that is the increased focus of the development work as a result of the sprint events. Therefore the null hypothesis could not be rejected: in the majority of cases, the productivity of KDE PIM developers after the sprints was higher than before.

As a more general message, Plone and KDE PIM share the same pattern: after the sprints

Figure 4. Commits during plone sprints



developers increase their efforts as compared to before the sprints. Also, the sprinting events serve as gatherings of exceptional productivity: even simple metrics like the one proposed in this article can be used to share knowledge, motivate and reinforce standard rates of effort.

Threats to Validity

The following aspects have been identified which could lead to threats to validity of the present empirical study:

- **Productivity:** This metric as discussed above (number of commits per day) is limited by its fragility; there are a large series of events which may affect this figure. The major consideration in the applicability of this metric is the requirement of other community knowledge in order to provide qualitative explanation for the measurements. Commit-per-day is a sensitive measurement and can be affected by many external forces, e.g.~lower commit rates are probable on public holidays. Commits-per-day can also be affected by many internal forces, e.g.~reaction to a security flaw in the codebase may increase the commit rate. As a result the observer must be careful to understand the full context of commit rates within the project and around the dates of the events being assessed.
- **Context Periods:** The commit examination period, *i.e.*~one week before and one week after the sprint, could be problematic. Some developers could skew the distribution with many exogenous causes: they could profuse very little effort just before the sprint as they prepare for travel, or on the other hand, some developers might do a lot beforehand to make their code look better.
- **KDE PIM Timings:** The evaluation of KDE PIM sprints is problematic due to their timing. Each KDE PIM sprint occurs within around the first week of a new year. It is possible that the productivity for the week before a KDE PIM sprint is reduced in this new year period.

Figure 5. Commits during KDE PIM Sprints

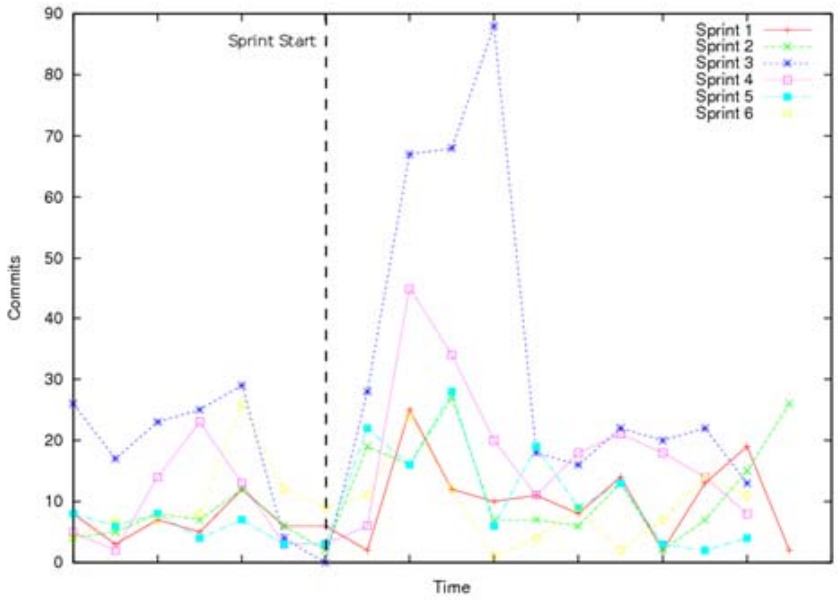


Table 4. Average commits-per-day during plone and KDE PIM Sprints

		Median		Average		Test of HP1 – Commits-per-day	
		Baseline	Sprint	Baseline	Sprint		
Plone	Sprint 1	**4**	3,5	**5,97**	3.5	x	x
	Sprint 2	**4**	2,5	**4,75**	2,5	x	x
	Sprint 3	**6**	2.5	**7,97**	3.25	x	x
	Sprint 4	**6**	39.5	**7,97**	42.5	√	√
	Sprint 5	**7**	1	**8,32**	2.25	x	x
	Sprint 6	**12**	117.5	**19,1**	105.5	√	√
KDE PIM	Sprint 1	**12**	13	**12,9**	14	√	√
	Sprint 2	**20**	18.5	**20,57**	18.25	x	x
	Sprint 3	**11**	68,5	**13,26**	63.75	√	√
	Sprint 4	**7**	35	**9,67**	29.33	√	√
	Sprint 5	**12**	20	**16,42**	19	√	√
	Sprint 6	**15**	28	**20,37**	28.71	√	√

FURTHER WORK AND CONCLUSION

This article provided a research study into the increasing phenomenon of mixed software development paradigms, when blending together some aspects, practices or shared tools. From the theoretical standpoint, both the Agile and Free Software methodologies have been studied in the past, and several shared concepts have been identified. The Agile paradigm has been deemed as the "commercial'' counterpart of the Free Software approach in many respects, and Free Software developers have been reported in

Table 5. LOCs-per-day in the sprints, as compared with the seven days immediately after the sprints. For the first three KDEPIM sprints, and the fifth Plone sprint, no LOCs data is available from the SVN servers.

Average				
		During the sprint	7 days after the sprint	Test of HP2 – LOCs per day
Plone	Sprint 1	0.63	8.29	x
	Sprint 2	3.33	0	√
	Sprint 3	32.18	28.83	√
	Sprint 4	458.84	0	√
	Sprint 5	N/A	N/A	-
	Sprint 6	70.22	55.59	√
KDE PIM	Sprint 1	N/A	N/A	-
	Sprint 2	N/A	N/A	-
	Sprint 3	N/A	N/A	-
	Sprint 4	173.92	84.87	√
	Sprint 5	151.05	95.92	√
	Sprint 6	102.61	17.24	√

Table 6. Summary of results for the test of hypothesis 2

		Before		After		Test of HP2	
		Median	Average	Median	Average		
Plone	Sprint 1	4.5	7.25	3.5	3.5	x	x
	Sprint 2	1	1.25	0	0	x	x
	Sprint 3	3	3.25	2	2.71	x	x
	Sprint 4	7	6.5	11	15	√	√
	Sprint 5	7	5	5	13.5	x	√
	Sprint 6	20	31	38	57	√	√
KDE PIM	Sprint 1	7	7.71	12	12	√	√
	Sprint 2	7	7.29	8	8.43	√	√
	Sprint 3	24	18.71	21	20.57	x	√
	Sprint 4	6	10	19	16.71	√	√
	Sprint 5	7	6.57	5	8.57	x	√
	Sprint 6	11	13.14	28	28.71	√	√

the past to have adopted Agile techniques and practices. Still, no rigorous studies have been carried out to evaluate the results and the effects when a development approach (*i.e.*, Free Software) incorporate a practice (sprinting) belonging to another paradigm (*i.e.*, Agile).

Recent empirical research has looked beyond the usual data sources for indicators of quality within Free Software development projects. As such, measuring aspects of the developer community has become of interest (SQO-OSS, 2008). Within this article, commits-per-day was

presented as a simple metric for developers productivity and it was shown how it can be used to measure the impact of events within the release cycle, specifically developer sprints. Through this metric, it was also possible to characterise an Free Software project based on its "base rates'', *i.e.* the global average and median of the accomplished productivity.

Two Free Software projects were selected for study, Plone and KDE PIM: the rationale of this selection was based on a reported use of the sprinting practices in the two projects, and insider knowledge was available due to one author being part of one of the projects. The first project had a larger number of sprinting events, attended by an increasing amount of developers. The second had sprint events just once in a year, and overall only 6 events have been recorded. Two research hypothesess were formulated: first, the productivity during the sprinting events increases with respect to the base rates observed in a Free Software project. Second, the sprinting event has a follow-up effect on productivity, and the amount of commits per day increases after the events, compared to the period before.

Results and insider knowledge showed a composite picture: the Plone project had just 2 sprints during which the productivity increased over the base rates, whereas all the KDE PIM events produced a larger amount of commits per day than the average. It was reported also that Plone meetings hosted an increasing numbers of developers, which could hint a higher cost of coordination of efforts. On the other hand, results showed that within Plone, the sprint events have lately started to show an increasing productivity after the event (compared to before), whereas the earlier meetings did not achieve the same levels. The sprints in KDE PIM, instead, always increased the rate of commits per day: less frequent, and more focused meetings have been deemed responsible for this pattern.

Further work has been identified in further studying the quality of the alleged increased productivity: the code committed during the events should be analysed in order to assess its quality compared to the average baseline within the project. Its complexity should also be assessed: Free Software developers should consider whether to commit large amounts of new code, but with higher complexity, or to commit smaller portions, but with an overall lower complexity. assessed: Free Software developers should consider whether to commit large amounts of new code, but with higher complexity, or to commit smaller portions, but with an overall lower complexity.

REFERENCES

Cameron, Lynne (2003). Challenges for ELT from the expansion in teaching children. *ELT Journal, 57,* 105 – 112.

P. Adams, A. Capiluppi, and A. de Groot. Detecting agility of Free Software projects through developer engagement. In Proceedings of the 4th International Conference on Open Source Systems, 2008.

K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, October 1999.

A. Capiluppi and M. Michlmayr. From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In J. Feller, B. Fitzger- ald, W. Scacchi, and A. Silitti, editors, Open Source Development, Adoption and Innovation, pages 31–44. International Federation for Information Processing, Springer, 2007.

K. Crowston, H. Annabi, J. Howison, and C. Massango. Effective work practices for software engineering: Free/libre open source development. In ACM Workshop on Interdisciplinary Software Engineering Research, 2004.

B. During. Sprint driven development: Agile methodologies in a distributed open source project

(pypy). In P. Abrahamsson, M. Marchesi, and G. Succi, editors, XP, volume 4044 of Lecture Notes in Computer Science, pages 191–195. Springer, 2006.

G. Goth. Sprinting toward open source development. IEEE Softw., 24(1):88–91, 2007.

S. Koch. Agile principles and open source software development: A theoretical and empirical discussion. In Extreme Programming and Agile Processes in Software Engineering: Proceedings of the 5th International Conference XP 2004, number 3092 in Lecture Notes in Computer Science (LNCS), pages 85–93. Springer Verlag, 2004.

E. Raymond. The Cathedral and the Bazaar, chapter The Cathedral and the Bazaar. O'Reilly & Associates, Inc., 1999.

K. Schwaber and M. Beedle. Agile Software Development with Scrum. Prentice Hall, October 2001.

A. Sigfridsson, G. Avram, A. Sheehan, and D. K. Sullivan. Sprint-driven development: working, learning and the process of enculturation in the pypy community. In J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, editors, OSS, volume 234 of IFIP, pages 133–146. Springer, 2007.

The SQO-OSS Project Consortium. D7 – Novel Quality Assessment Techniques, February 2008.

J. Warsta and P. Abrahamsson. Is open source software development essentialy an agile method? In 3rd Workshop on Open Source Software Engineering, 2003.

F.Wilcoxon. Individual comparisons by ranking methods. Biometrics Bulletin, 1(6):80–83, 1945.

ENDNOTES

1 http://www.kde.org/

2 http://plone.org/

3 Plone Naples sprint: http://www.openplans.org/projects/plone-conference-2007/sprint

4 The details here are based upon information derived from an interview held in 2008 with Adriaan de Groot, Vice President of KDE e.V.

This work was previously published in International Journal of Open Source Software & Processes, Vol. 1, Issue 1, edited by S. Koch, pp. 58-71, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 8.8
Agile SPI: Software Process Agile Improvement—A Colombian Approach to Software Process Improvement in Small Software Organizations

Julio A. Hurtado
University of Cauca Colombia, Colombia

Francisco J. Pino
University of Cauca Colombia, Colombia

Juan C. Vidal
University of Cauca Colombia, Colombia

César Pardo
University of Cauca Colombia, Colombia

Luís Eduardo Fernández
University of Cauca Colombia, Colombia

ABSTRACT

This chapter presents Agile SPI, a framework in which the main goal is to motivate small and medium size enterprises (SMEs) towards improving and certifying their software development processes. This framework was born in the SIMEP-SW project where a software process improvement model for supporting process improvement in the

Colombian software industry context was built. We present Agile SPI, its origin, development, principles, architecture, main components, and the initial experiences.

INTRODUCTION

Nowadays, the software industry represents an important economical activity; it offers different possibilities for business and it aims to be a great opportunity for developing countries. In Latin American countries, the software industry is usually immature; companies face an undisciplined process and that means quality is unpredictable (Mayer&Bunge, 2004). Not only will it be impossible to plan and manage quality without a mature environment, but also when we achieve it, we will not know why and we could not repeat it (Hurtado, Pino, & Vidal, 2006). The Latin American software industry has grown smoothly, so the generation of strategies for achieving the software process improvement (SPI) environment that would allow organizations to take advantage of effective software processes. Software quality assurance through software process improvement is one of the strategies software companies could engage in with two goals: the first one is to improve the quality process so that they can get into a new market, and the second one is the need for their processes, like administrative units, to become more efficient and effective (Pino, Garcia, Ruiz, & Piattini, 2006).

One of the characteristics of the Latin American software industry is that it is mainly formed by small and medium size enterprises (SMEs). Most of these companies did not have a defined software process improvement project basically due to the great initial investment required and the disadvantage of their personnel competitiveness on the software process areas. The special characteristics of small companies cause that processes improvement programmes must be applied of a way particular and visibly different from the typical way the great organizations do it, this is not as simple as the fact to consider these programmes as versions to minor scale of great organizations (Richardson, 2001; Storey, 1982).

Agile SPI (Hurtado et al., 2006) is a framework SPI based on a strategy for institutionalization the software process in the small organizations context. For the process to be sustained, process behaviour needs to be integrated into the organization's culture. A process is institutionalized when it is followed consistently and performed naturally by everyone involved in performing the process activities. This will happen when the SME has in place a framework for software process improvement appropriate, for that reason Agile SPI is composed by three components:

- Two light models, a light reference model and an evaluation model, with a set of processes which are typically used by the Colombian small software organizations.
- Agile process for supporting a project SPI. A process SPI model for guiding a SPI program of way agile. A process definition model for supporting the implementations of improvements.

In the context of the SIMEP_SW[1] project, a pilot experience was carried out in order to validate the theoretical results of this research. Some programs SPI in a Colombian software development companies was implemented according to the guidelines suggested by Agile SPI. The results were analysed for validating our model and improving it.

In the second section is presented the background. The third section presents the Agile SPI origins including a description of the SIMEP-SW project and its developing. Agile SPI, its principles, architecture and main components are presented in the fourth section. The study cases are presented in the fifth section. Finally, the sixth section presents the main conclusions and describes perspectives.

BACKGROUND

The software industry is characterised for a fast product innovation but not by the process refinement. Software process improvements are required to increase the productivity of software companies. Generally, it is the aim to increase the quality of the produced software and to keep budget and time. Quality models for software process improvements were developed in context of large organisations and multinational companies. In the Latin American software industry, there are many problems in a SPI efforts because they are based on models created for others contexts (geographical, human, sizes of organizations, technologies used, process users and others factors). In some countries like Colombia, most of the organizations are small, with undisciplined behaviour, ad hoc process and absence of process focus. The same situation is presented in most of the countries of the world, even in development countries exists a great capacity in small enterprises. The models and technologies associated to the process must be adapted to the organization's needs. The quality models usually do not consider the factors of success for the companies, such as profitability, competitiveness, strategy of market and satisfaction of users (Conradi & Fuggetta, 2002).

The traditional frameworks focuses the efforts for the discipline and the risk reduction. Nowadays, the software products looks for the competitiveness: faster, better, and cheaper. Inside the focus of SPI models, Conradi and Fuggetta (2002) present three differences: process assessment, process refinement, and process innovation. Models like CMMI contribute to assessment and refinement focus, however the Quality Improvement Paradigm (QIP) contributes to innovation focus through the reuse concept. Printtzel and Conradi (2001) presented a taxonomy in order to compare the SPI frameworks based on some causal relations between process quality and process quality. They compare CMMI, ISO, ISO/IEC 15504, QIP, Experience Factory (EF), Gode Question Metric (GQM), and Software Process Improvement for better Quality (SPIQ). This strategy was important for defining the causal relations for Agile SPI.

Latin American countries have been concerned in recent years about software quality and development processes in their own industry, becoming as a main feature in increasing product quality (Bedini, Llamosa, Pavlovic, & Steembecker, 2005). For example, the "MoProSoft" model from Mexico and the "MR.MPS" from Brazil, amongst others that could be mentioned.

In Mexico, the MoProSoft model has been developed - "Modelo de Procesos para la Industria de Software" (Oktaba, 2005) (Processes Model for the Software Industry). This model is based on ISO 9001:2000, ISO/IEC 15504-2:1998 y CMM. MoProSoft aims to provide the software industry in Mexico with a model based on the best international practices. This model is at the same time easy to understand, simple to apply and economical to adopt. It seeks to assist organizations in standardizing their practices, in the assessment of their effectiveness and in the integration of ongoing improvement. MoProSoft defines three process categories: organizational, management, and operation. It specifies three parts for each process categories: a general definition, practices, and a guide for adjustments. A tenet of its improvement strategy is that the organization should establish its own strategy for the setting up of the processes defined by the model. The processes should evolve in alignment with the suggestions for improvement. The organization's strategic plan will be reached with increasingly ambitious goals being set all the time. In this way the organization can reach maturity progressively, by this ongoing and continual improvement in its processes.

In Brazil, the MPS.BR project (Weber, Araújo, Rocha, Machado, Scalet, & Salviano, 2005) has been developed. Its basis lies in ISO/IEC 12207:2002, CMMI e ISO/IEC 15504:2003. The MPS.BR project has two models: a Refer-

ence Model for the software improvement process– MR_MPS along with a Business Model for the software improvement process– MN_MPS. MN_MPS defines the elements and interactions involved in the certification of the organization by implementing MR_MPS in two ways: first one for an organization and second one for a group of organizations together (thus managing to make it more affordable for SMEs). MR_MPS is made up of maturity levels, along with an assessment model. The maturity level is organized in two dimensions: capability and process. The process maturity is classified into seven levels: optimized, managed quantitatively, defined, almost completely defined, partially defined, managed, and partially managed. Process areas are attributed to each maturity level based on the levels of CMMI. This is so as to ensure a gradual and fully appropriate implementation in Brazilian SMEs. The implementation level of the practices, associated with each process area, is evaluated by means of indicators.

In the previous models have been set out explicitly no improvement strategy for guiding an improvement process. Agile SPI improvement strategy is based on providing the organization with an agile process which supports the basis for addressing a SPI Program.

In Blowers and Richardson (2006); Garcia, Graettinger, and Kost (2006); Scott, Jeffery, Carvalho, D'Ambra, and Rutherford (2001); Wangenheim, Weber, Rossa Hauck, and Trentin (2006), among others, are observed others related efforts to software process improvement in small software organizations.

Agile SPI was developed inside to SIMEP-SW project (An Integrated System for the Improvement of Software Development Processes), between the 2004 to 2006 years, supported by the University of Cauca, Colciencias and some Small enterprise of South Colombian.

Agile SPI - Process is an agile process of software processes improvement, which can be used as guide for the implementing of a software process improvement program in SMEs. Agile SPI - Process takes as premise the precepts of the Agile Manifesto and the requirements for a light SPI, which have been adapted to the necessities of a software processes improvement program for small software enterprises. These are:

1. The highest priority is to satisfy the client's necessity through the early and continuous delivery of significant improvements to the development process because of Agile SPI - Process provides a light and agile process of software processes improvement.
2. There are not stable requirements of improvement. For this reason, the diagnosis is a key phase. Even so, requirements of improvement that arise will be prioritized and welcomed as it is feasible to carry out them.
3. To give frequently improvements of the software process.
4. An improvement program with Agile SPI - Process should be based on the effective collaboration among the consultants, the improvement group, the top management, the development group, the SQA group, marketing and other dependences related with the SPI project.
5. To build projects around motivated individuals toward the improvement of individual, group, and organizational processes. To give them the opportunity and the support that they need and to offer to them trust so that they carry out the tasks.
6. The most efficient and effective form of communicating round trip information inside an improvement team is through face-to-face conversation.
7. The maturity of process, as the average performance of the projects, should be the main measure of the progress improvement. The base measurements to measure the performance are the productivity and the quality.

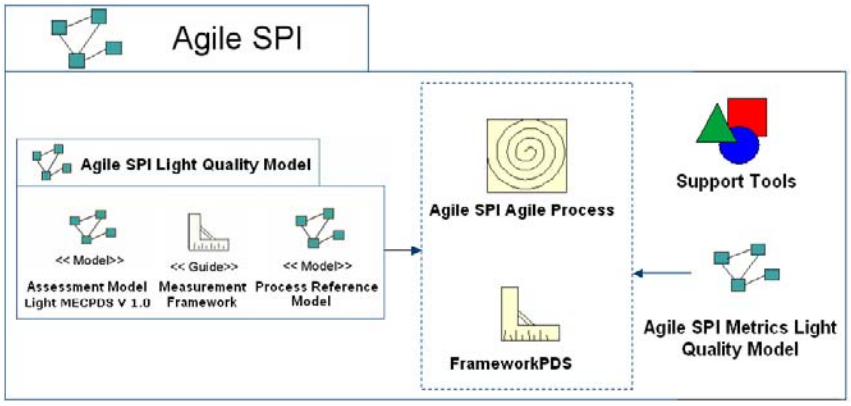
8. Agile SPI - Process promotes the sustained development. The work must be continuous and indefinite.
9. Agile SPI - Process promotes a technical and management infrastructure, appropriate to support the process improvement.
10. Agile SPI - Process promotes the conformation of a dynamic organizational infrastructure, based on objectives, not in control strategies.
11. Agile SPI - Process promotes the continuous learning as a key discipline. The objective of this discipline is to allow knowing the work, to meditate about this and to adjust the work through short and concise iterations.
12. Agile SPI - Process promotes the effective conformation of the teams proposed by its infrastructure, it worries about the quality of the human work to carry out.

Agile SPI aims to support process improvement for the Colombian software industry. Its main goal was to motivate small and medium size companies towards improving and certifying their development processes. The framework must include practical recommendations for process implementation, in order to facilitate their internalisation by a person when following the process and institutionalisation by organizations when everyone follows the common process and the process discipline is enforced, and also it should consider a tool for process definition. In the context of the SIMEP-SW project, a pilot experience was carried out in order to apply the theoretical results of this research. Inside these project researchers of University of Cauca and some small enterprise worked for to analyse, design, and proof the different models and strategies for improve the local industry in the Popayán and Cali cities. Some strategies was adopted in addition to the Agile Model, requirements for a company to achieve a maturity level CMMI using: agile practices, adoption of reuse strategies like software product line and adoption of agile methods for implementing some practices, and a collaborative focus for improve the humans process. The most important contribution of SIMEP-SW project was its participation in the project Competisoft Project[2]. This new project supported by CYTED aims "To increase the competitiveness level of the Latin American Software SMEs by means of the creation and diffusion a methodologic framework.

Framework Agile SPI for Software Process Improvement

Agile SPI is a framework based on models lightweight, international standards, agile improvement, and agile practices. Agile SPI promotes an improvement with lacking of agility, promotes the agility with quality and recognizes in the innovation the most promissory source for the improvement of all the aspects that involves a process: people, methods, techniques, and tools. The SMEs will be able to create its processes, if these do not exist, following its principles and needs, or will be able to follow the cycles and the models, if it is supporting a process previously defined and wants to improve it. The improvement focuses in the quality of the product and the productivity of the organization, from which to organize by disciplines it aims towards the securing of the quality of the product and that to maintain the process agile and to innovate in their components is the principles that move the productivity. The discipline give reliability in the process, a mature process offers to certain degree of reliability on the accomplishment of a project in the organization as far as the reach of its goals of quality and costs. The agility and the innovation are present to support the feature in which the traditionally mature processes never will consider, under this approach the process must be agile with the purpose of being able to be reconciled to a SME with its time and resource restrictions. And of course, the process must be

Figure 1. Agile SPI architecture



defined and institutionalised, and prudentially controlled.

Agile SPI Architecture

Agile SPI is an integral model for the Software Process Improvement that exhibits the following characteristic: the main focus is SME. In this kind of organisations there are a great interest in innovating as far as products or services, but not as far as the organization processes. The processes are ad-hoc, light or chaotic, there is not great motivation in improving this aspect. Agile SPI was created like a framework for supporting improvement in software SME. Usually, the quality assurance is more focused to the test than to have a complete program of quality control. In the best of the cases the quality assurance is focused to control processes instead of facilitating its improvement. The academy is looking for new forms of development, nevertheless is more focused in the technologies than in the processes or methodologies. The improvement of process is considered a remote work, reaching for a great companies. It isn't known that the improvement is possible from teams and its members up to enterprise level. The universities that distribute the work related to the improvement show the models of quality, their interpretation and all the required infrastructure to do it, this can motivate to work to improve the state-of-the-art, but it don't to change the state of the practice.

The Agile SPI architecture has been influenced by the structure of models defined by SEI and ISO, but considering others quality models of processes well-known internationally. Agile SPI structure has a main components of an improvement program: an improvement guide and models for supporting, Agile SPI - Light Quality Model and Agile SPI - Light Metrics Model. There are two dynamic elements of the static structure: the conceptual model Framework SPD (Software Process Definition) and the improvement process. The Figure 1 depicted the Agile SPI architecture.

Due to the context SME, Agile SPI is a complete framework (it is not a model that says what makes lack, but like complementing the process). It is functional, its application facilitate the complete operation of an improvement program. It search be understandable, to guarantee his learning and application: therefore it is simple and clear. It aims to be usable, that implies that the strategy must be clear, intuitive, flexible and adjustable to the needs. It aims to motivate, that implies that the improvements must be visible in short periods, it is necessary the sufficient motivation to guarantee

that the improvement project has continuity. It aims to implement a model for applying in the SME, that is to say, to apply the model must be viable (economically) and feasible (attainable). The flexibility of Agile SPI is based on its models independence. Agile SPI can be used in different contexts with several models.

The framework Agile SPI presents the following components:

- A guide for an improvement program called Agile SPI Process. It is a process that guides the efforts of a SME towards the adjustment of a software process adequate to its necessities. This process is the framework of reference for the management of the improvement projects, the framework includes a method, models, infrastructure, techniques, and the tools of support.
- A light quality model for Agile SPI. Agile SPI Light Quality Model integrated process and product, and that guides the organization of the people and the teams, the disciplines and the areas of work associated to the definition, application and improvement of the process towards a defined maturity level. Defining a set of processes which are typically used by the Colombian SME's. It allows to identify and diagnose problems of the industry as far as the process and that allows to planning the improves according to a process reference model.
- A light measurement model for Agile SPI. Agile SPI Light Measurement Model allows to measure: the performance of the process in the projects in which it is applied, to improve the estimations of the projects through the measurement of the effort, the maturity of the this and improvement of the process within the framework of a program SPI.
- A conceptual and technological frame for the definition, visualization and application of processes, Agile SPI - Conceptual Framework. This conceptual frame is based on metamodel SPEM - Software Process Engineering Metamodel, this frame is the conceptual base on which SPI and the tools of support are supported to all the models of Agile SPI. This conceptual framework allows to relate elements of process, with the elements of the quality model, with the model of evaluation, the measurement model. For example, the disciplines concept is a separating element of areas of the process and the structure of Agile SPI are defined based on this concept.

Agile SPI Focused on Disciplines and Process Components

Agile SPI allows to organize an improvement project, very integrated with the development projects, and each organizational and technical change is to handle through an experience with the purpose of visualizing in an isolated way, if the improvement really has been done. The improvement project follows a process of defined by Agile SPI - Agile Process, this is organized by small iterations, in which the different disciplines participates in a greater or smaller degree of intensity. Agile SPI is based on the concept of discipline like representative areas of process like approach of assurances. In the approach of refinement of the process Agile SPI defines the concept of capacity, which usually measures the quality degree of a discipline with respect to a quality model or process model. See Figure 2.

Agile SPI define three dimensions for improvement: capability, innovation, and agility with the following causal relationship:

- F1 (Discipline)→F2 (Capacity)→ Quality(Process)→ Quality (Product)
- F1' (Component Process)→F2'(Innovation in Component)→Quality (Process) → Quality (Product)
- F1'' (Discipline)→F2''(Agility on Discipline)→Performance(Process)→Productivity

(Project).

- F1’’’(Component Process)→F2’’’ (Learning in component)→Quality (Process) && Performance (Process)→Quality (Product) & Productivity (Project).

Therefore F1 + F1’+ F1’’+ F1’’’ ⇔ Quality(Product) & Productivity (Project)

The process components are created, evolved and replaced according to the results of the evaluation and the priorities of each organization. The improvement is visualized by disciplines and components of process. The improvements are organized by iterations that lean in the technology of processes to manage the configuration of process.

The Agile SPI: Process Life Cycle

Agile SPI – Process is an agile and light process of software processes improvement, which can be used as guide for the execution of a software processes improvement program in small and medium enterprises (SMEs). Light because enterprises like the SMEs which possess certain characteristics as: low resources, light processes, small human resource, limited economic availability, and so forth, need a model that supports an improvement program that consider the real characteristics of their industry, besides offering quick results in their improvement programs.

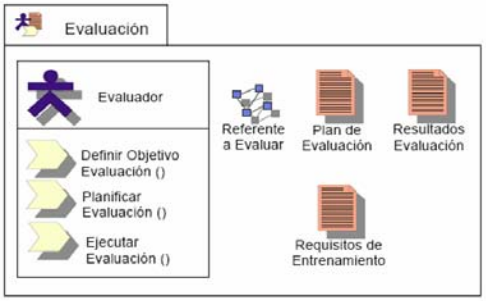
Agile SPI – Process is an iterative and incremental process is based on improvement cases, which has the feature of throwing quick results of improvement because it allows to create mini improvement programs that include improvement cases inside a global improvement program. The improvement cases are atomic units of improvement in the processes areas that have been selected to be improved either because the enterprise follows a certification or because for it its priority is to improve a specific process.

Obtaining quick improvement results will allow in consequence to the improvements to be visible from the early phases of the improvement project, more agile and quicker as the mini improvement programs finish depending on the prioritize criteria that the enterprise has defined previously. This seeks to maintain the personnel’s motivation toward the improvement program, through permanent improvement results, removing the risks of the project in the first phases, to focus the major effort in the areas that the enterprise considers more important for its business. The life cycle is highly influenced by the iterative and incremental life cycle models present in many development processes such as RUP, XP, Scrum, among others; since the improvement projects cover extensive requirements and they impact the whole organizational structure of any enterprise, some characteristics of these models have been adapted to create a complete, agile, and less bureaucratic improvement process and sensitive to the related activities with the management (meetings, documents, infrastructure, etc.).

Agile SPI – Process allows the parallelism between iterations or improvement programs, being very advantageous because improvements can be developed in processes areas where an evident independence exists. In the certification processes, the appraisal before the beginning of an improvement program is very important, since this allows estimating which processes areas the enterprise has and which is their maturity level. Currently many tools exist with which the process areas of an enterprise can be appraised according to a particular quality model, for example: CMMI (SEI, 2002) in its continuous or staged version, ISO/IEC 15504 (ISO_15504-2, 2004; ISO_15504-5, 2006), ISO 9001-2000 (ISO_9001, 2000), and others. In this aspect Agile SPI – Process can be used independently of the quality model and evaluation method, for example this it can be used if a continuous or staggered CMMI has been selected or if the choice is ISO 15504.

Figure 2. Example of discipline in Agile SPI framework.

Discipline **Evaluation** *from* **Agile SPI Process** *(e.g. apply to* **Discipline ABC.***)*
Apply to **Process Component:** *{Description - Artifacts - Roles - Workflow }* **Disciplina ABC**
Based on **Quality Model Component:***{Description, Goals, Practices, Measurement , verification }* **Disciplina ABC.**
With **Evaluation Component:** *{ Measurement, verification }* **Disciplina ABC.**
Measurement Component: {Productivity, defects} **Disciplina ABC.**
Composed by:
{Roles: Evaluator - Artifacts: Referent Quality Models and sub artifacts, evaluation plans, evaluation results, training requirements - Roles: evaluator - Workflow: Define evaluation objective, planning evaluation, execute evaluation}



Agile SPI – Process also includes, documents, and explains a set of disciplines to any improvement process that can be applied in smaller or bigger measure in each one of the phases in which several iterations can be developed, for this we were based on the Software Development Unified Process. When we identify in Agile SPI – Process the disciplines to be developed in each one of the improvement process phases, we are assuring that personnel involved in the improvement program will be able to visualize in major detail the behaviours and activities that should be developed inside an improvement program. We have considered as vital disciplines in any improvement program: Training, Improvement Management, Evaluation, Analysis, Design, Installation, Process Configuration Management and Learning.

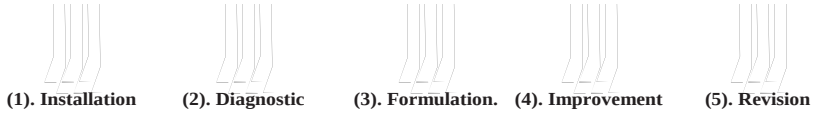
The Agile SPI - Process Phases

Agile SPI - Process describes a process of software processes improvement in five phases, next we will see each one of them:

The Figure 3 presents to Agile SPI – Process phases: installation, diagnostic, formulation, improvement, and revision of the program.

- **Phase 1 —Installation:** This is the beginning phase for Agile SPI - Process. Motivation should exist in the organization to undertake a plan of improvement of its processes. In this phase a proposal of improvement is created based on the business needs, which will help to guide to the organization through each one of the following phases, this proposal must be approved by the management to guarantee this way the assignment of the necessary resources for the improvement project. During this phase some objectives also are defined, which are established from the enterprise needs. Besides a feature very important in Agile SPI - Process is offering a guide in the improvement of software processes, also provides a management infrastructure, which describes the way in which committed people are organized inside the improvement effort; this infrastructure organizes the improvement

Figure 3. Agile SPI: Process phases modeled under SPEM



effort keeping in mind a management team (MT), a processes technology team (PTT), and improvement teams (ITs); these teams have been influenced by the infrastructure proposed by IDEAL, complementing it with the creation of effective groups proposed by the methodology TSP (Team Software Process), adapted by Agile SPI - Process like TSPI (Team Software Process Improvement) and some of the features in the administration of a project using the SCRUM methodology.

- **Phase 2—Diagnostic:** In this phase, a program has already begun toward the improvement of processes and the work that here is realized is fundamental for the realization of the following phases. An appraisal is realized to know the general state of the enterprise processes, besides an analysis of the results that will allow establishing the priority of the improvement cases, allowing this way to create one of the main products of this phase known as "improvement general guide or plan."
- **Phase 3—Formulation:** In this phase, the most high-priority cases of improvement (1 or 2) are taken to improve according to the results of the appraisal made in the previous phase and the planning of a first improvement iteration is realized, this with the purpose of finding a measure of the effort that serves as base for the estimate of the effort that the rest of the improvement project will take to carry out.
- **Phase 4 —Improvement:** In the improvement phase of Agile SPI - Process the whole effort of the improvement cases is managed based on the estimate made in the improvement execution plan created in the previous phase and consequently the plans corresponding to the different iterations of the process areas to improve or to create are developed. A document should exist where it is registered the execution of the test pilots, the evaluation of the new process areas or the new improvement that has been realized. If the pilot plans have been developed satisfactorily it is necessary to create acceptance and institutionalisation plans of the new processes in the enterprise.
- **Phase 5 —Revision:** In this phase, a feedback is made before starting the beginning phase again. In this phase all the learned lessons and the metrics developed to measure the accomplishment of the objectives serve like knowledge base or source of information for people involved in the following improvement cycle. With all the gathered information the realized work should be evaluated and all the elements related with the execution of the program SPI should be corrected or adjusted, how for example the established infrastructure, the used methods, the communication channels and if the solutions to the identified problems were the appropriate ones.

Iterations in Agile SPI - Process and Their Correspondence with the SCRUM Development Process

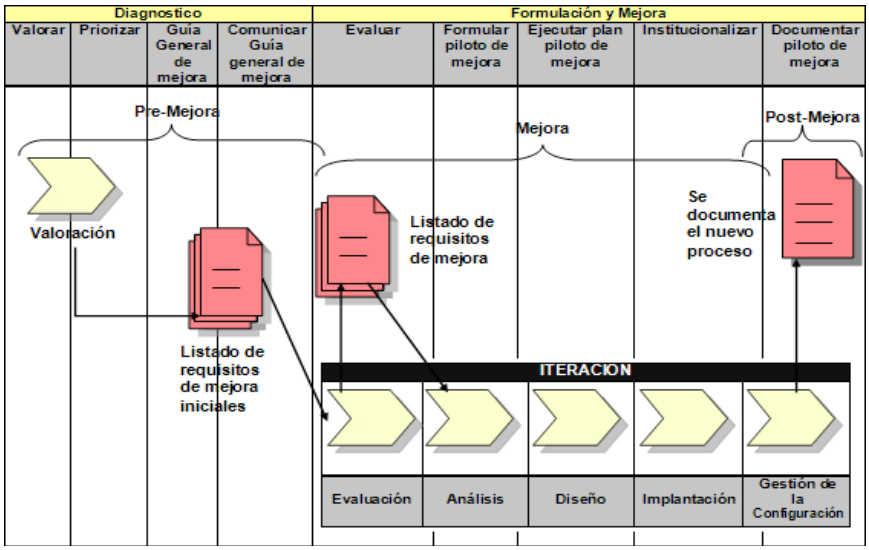
Iteration in Agile SPI - Process is a mini cycle improvement that allows advancing the development and management of a set of improvement cases in an independent way. The iteration is the integrative concept between phases and disciplines. The phases can be decomposed in time and space (teams) for iterations, and an iteration, being itself a guide of improvement, is defined starting from a set of disciplines according to the phase where it is and to the characteristics of the improvement project. Iterations in Agile SPI - Process is a very important part in the Software Processes Improvement because this way independent improvements can be developed, and so to deliver quicker improvements. The key resides in developing iterations in areas that are independent of others, this way the work in them can be achieved in parallel, and the work of improvement that in them is developed doesn't cause problems; nevertheless it is necessary to keep in mind that a dependence can exist among areas, in that case it is necessary to study which is the impact that could create a improvement case and based on this to order the way of how improvements will be developed in the other improvement cases.

The iterations for the improvement cases in the diagnostic, formulation, and improvement phases can be developed in a similar way to the Scrum development process with its sprints, in each one of them there are three phases: pregame, game, and postgame, which we have called them pre-improvement, improvement, and post-improvement.

Improvement cases can be seen as Sprint Backlog (Improvement Requirements List), which have been divided in smaller improvement tasks, and these in their entirety and respective correspondence would conform a specific area that would be seen as a prioritised list of features required by the improvement, which has been obtained thanks to the appraisal achieved to the enterprise; in relation to the SCRUM development process this list of features is created starting from the client's requirements and is called Product Backlog (Initial Improvement Requirements List).

In Figure 4 is showed how the iterations in Agile SPI - Process are developed. In the pre-improve-

Figure 4. Iterations in Agile SPI – Process



ment phase the appraisal discipline is developed for the creation of the product accumulation or delay registry (initial improvement requirements list). After prioritising the areas to improve, an improvement requirements list is created through the evaluation of each area, which allows defining an order for each one of the improvement iterations to execute. It is important the assistance of dependences net to order in a detailed way the improvements to achieve in the selected areas. The dependences net allow identifying the improvement cases or the number of iterations to develop for each area. The improvement cases are the activities that compose the area.

After identifying the improvement cases that compose the area and ordering them through dependences net, the next is the analysis and design of the new or improved process and its respective installation and documentation.

Agile SPI - Process also adapts and proposes techniques and practices for the teams conformation and management, it exemplifies the net of dependences that can exist between areas or practices that they compose depending of the quality model and the way of treating this characteristic. It also documents and identifies the milestones and workproducts resultants of these and it proposes some control and management templates for the improvement process, all this inside a guide that doesn't seek to be an extensive model but a friendly, easy to use, and agile process of software processes improvement independent of the quality model and evaluation methods, and the most important: adapted to the characteristics of the Colombian software industry and in general Latin America.

At the moment, Agile SPI - Process is in its verification stage, we are achieving improvement programs in some small enterprises of our region, which have demonstrated from the beginning a great interest to improve their processes after they knew the advantages that it can bring to the quality of their products, as well as the benefit of having an international certification. With this process for the software processes improvement we seek to help to the enterprises not only of our region but also of Latin America to reach a level that allows them to compete with companies like the American and European in software development.

STUDY CASE: APPLYING AGILE SPI IN A SMALL SOFTWARE ORGANIZATION

This section aims to present the applying the framework Agile SPI in the organization SIDEM Ltda (http://www.sidemltda.com) - Colombia.

SIDEM Ltda.

Sidem Ltda. is an organization in Cauca's Valley, from the Cali City, Colombia. This organization is dedicated to production, integration, maintenance, supporting and consulting of Information Systems, using multiplatform design to support the constant challenges of productive processes.

Nowadays, Sidem Ltda. account with more of 300 clients in Colombia, using his management and financial solution. These solutions allowed to this organization to be classified by the "Cámara de Comercio de Occidente" like enterprising industrialists of the Cauca's Valley and to position itself like one of the best organizations in the Colombian south.

Experience with SIDEM Ltda.

Previous View to the Software Improvement Process

At the beginning six years ago, Sidem Ltda. was a small organization with four people, which one developed several activities. There were not defined roles or positions in a specific way and the software process was not documented. Two

years ago, Sidem Ltda. is growing because account with 18 persons: one person is in charge to assign activities of client support, 11 development engineers, 2 persons in marketing and management area, 2 people in quality area, and 2 persons in management jobs.

In Sidem Ltda there are not areas defined in a clear way, we just found defined one organizational structure of the organization with some activities and persons assigned to them, overlapping among them and without documents. For that reason, there are chaos and immaturity process without relation. Almost all the time, the activities are not finished in time, budget and quality, because it is necessary to attend other problems and activities, which wastes too much time.

As a first step, Sidem Ltda created a Quality Team composed by 2 people. This team was created because:

- Defining software development activities
- Madding documents about development process
- Establishing control activities on the development process.

But, due to problems inside Sidem Ltda, the Quality Team finished madding another activities and that first effort was lost.

Any way, we founded another effort for improving process. Sidem Ltda there was implemented two projects with adoption UP Methodology and UML. The results obtained were a very good results for the team project. These results showed a better management and planning of the activities.

Initial Phase

First, we made an install meeting with Sidem Ltda. The Sidem manager presented the organization and necessities. Also, he was concerned about immature process, bad development methodologies, problems with estimating time and cost and chaos generated in management and development of projects.

The SIMEP-SW Team explained that an improvement project could be one of the most important solutions and decisions that Sidem Ltda should take and support. Consequently, the organization decided to initiate an improvement project based on the components of the Framework Agile SPI.

The SIMEP-SW Team proposed: 2 researchers to develop the appraisal on CMMI, 1 researcher in charged to obtain the development process implicit in the organization and 2 researchers in charge to manage all the Improvement Program based on Agile SPI Process, 1 researcher as a leader team.

At the same way, Sidem Ltda assigned an Improvement Team, working as a management bridge for supporting all the necessities during the Improvement Program. Fits, schedules, meetings, and necessary material in each meeting program. Each meeting should be programmed with two weeks before, with the objective of having a suitable management time.

When the two teams were ready for starting activities, we started a training about:

- Bases of Agile SPI Process.
- Reference Model CMMI.
- Presenting study cases.

After of training to the people, the SIMEP-SW Team prepared a plan or proposal of improvement with: objectives of improvement, the identified necessities of business, the necessary resources, roles and people, strategies in the development of the objectives, a schedule of work, and possible risks in the course of the project. Finally, we presented the improvement program to Sidem Ltda and we installed it.

Diagnose Phase

In order to know the current state of Sidem Ltda the SIMEP-SW Team made an appraisal in Level 2 CMMI, using SPQA Web Tool, which we would obtain an overview in the process capacity. With this process baseline, we could apply evaluations more detailed. In according to the results and recommendations generated by SPQA.WEB Tool:

- In a general way, acceptable requirement management. Then it is necessary an improvement of this area because this allows to collect and to control requirements, and to avoid deviations throughout the development of products.
- Project Planning, Project tracing and subcontract management are in a low degree of implementation. In consequence, the organization cannot control its development processes.
- Quality Assessment, configuration management and metrics are not implemented in the organization.

With the appraisal results, it was established a meeting with the objective to obtain the areas that should be improve. Finally, we obtained the next conclusion:

CMMI Engineering Area

- First Iteration: Management Requirements and Development Requirements.
- Second Iteration: Technical Solution and Product Integration.
- Third Iteration: Validation and Verification

Later, we began the training activities about:

- CMMI Engineering Area
- Management Requirements and Development Requirements

The work product about to improvement program was updated and new dates were specified. We planned evaluation activities in process areas selected. With the evaluation results, we designed a improvement plan based on the priorities selected by Sidem Ltda. Consequently, we generated a work product, which was communicated in all the organization, the objective was to establish a feeling of responsibility in the areas selected.

Formulation Phase

Nowadays, the Improvement Project in the organization Sidem Ltda is in this phase.

Once we made formulation of improvement cases, we will prepare the design of improved process. We will implement pilot proof. After studying of the impact, if the improvement is positive and advisable for the Sidem Ltda Process, we will deployment in formal way all of the new improved process.

This study case presents the results of one first stage or cycle for the first period of a improvement project. In this study case until now the deployment of Agile SPI Process has been made during eight weeks. Two weeks for Installing, two weeks in diagnose phase and four weeks for the formulation phase. We hoped that the Improvement Project will continue and that always represent one of the primary targets of the Sidem Ltda.

Learned Lessons

The following lessons have been learned as a result of application Agile SPI Process:

- The management leader must agree in applying Agile SPI - Process and promoting.
- Communicate to all organization about the applying Agile SPI - Process.
- Develop tasks in a smooth way.
- It is important to obtain results quickly to maintain motivation in the improvement program.

- The improvement process must be planned, be managed and the necessary resources for their development are due to assign.
- Good communication between organization team leader and improvement team leader.
- If the personnel available in the organization is limited, then to make sure that parity in the assigned work exists.
- There is a confusion between processes and structure.
- The organization not always knows clearly his processes.
- Many organizations have a implicit process, it is necessary to document.
- Training about modelling of the business process and the development process.
- It is not necessary planning objectives of improvement that will not be carried out.
- If an improvement first cycle has not all support and commitment, it is better to choose not to generate negative experiences in organization with unsuccessful improvement programs.
- The improvement process does not have to be left, to be suspended, or diminished because of other events, this it must be considered of greater or equal importance than the projects or diverse situations that can be presented in the organization.

CONCLUSIONS AND PERSPECTIVES

The software process in the organizations requires evolution and maturity to approach to its different stakeholders and continues improvement and Assessment. Therefore, they have arisen different kinds of Frameworks normally named like Quality Models and Improvements methods for supporting SPI strategies. In this paper we had presented Agile SPI, a framework based on models lightweight, international standards, agile improvement, and agile practices. Agile SPI is mainly influenced by the SMEs, the agile manifest, Conradi-Fuggetta thesis, and the existent and well-know models. Agile SPI includes a flexible infrastructure based on Discipline concept of SPEM and define five contexts for applying itself. The initial improvement is measured by the product quality and the project productivity, and then by the process capability and agility. Agile SPI is differenced of others Frameworks due and this is complete respect to the models, is flexible due a permits the inclusion of other models and was designed for SME industry. Agile SPI respect to other Latin-American initiatives is different because include a improvement model. This model permits apply the framework to an improvement program.

ACKNOWLEDGMENT

This work has been funded by the following projects: SIMEP_SW financed by Colciencias and University of the Cauca; COMPETISOFT (506AC287) financed by CYTED and MECENAS (PBI06-0024) granted by the “Junta de Comunidades de Castilla-La Mancha.”

REFERENCES

Bedini, A., Llamosa, A., Pavlovic, M., & Steembecker, K. (2005). *Quality software map of South America*. In *Proceedings of the 1st International Research Workshop for Process Improvement in Small Settings*, Pittsburgh, PA (pp. 216-227).

Blowers, R., & Richardson, I. (2006). The capability maturity model (SW and integrated) tailored in small indigenous software industries. In *Proceedings of the 1st International Research Workshop for Process Improvement in Small Settings*, Pittsburgh, Carnegie Mellon University (pp. 175-181).

Conradi, R., & Fuggetta, A. (2002, July/August). Improving software process improvement. *IEEE Software, 19*(4), 92-99.

Garcia, S., Graettinger, C., & Kost, K. (2006). *Proceedings of the 1st International Research Workshop for Process Improvement in Small Settings, 2005* (Special report CMU/SEI-2006-SR-001), Pittsburgh, Software Engineering Institute. Retrieved December 14, 2007, from http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06sr001.pdf

Hurtado, J., Pino, F., & Vidal, J. (2006). *Software process improvement integral model: Agile SPI* (Tech. Rep. No. SIMEP-SW-O&A-RT-6-V1.0). Popayán, Colombia, Universidad del Cauca-Colciencias. ()

ISO_15504-2. (2004). *ISO/IEC 15504-2:2003/ Cor.1:2004(E). Information technology: Process assessment, Part 2: Performing an assessment.* Geneva: International Organization for Standardization.

ISO_15504-5. (2006). *ISO/IEC 15504-5:2006(E). Information technology: Process assessment, Part 5: An exemplar process assessment model.* Geneva: International Organization for Standardization.

ISO_9001. (2000). *ISO 9001:2000. Quality management systems: Requirements.* Geneva: International Organization for Standardization.

Mayer&Bunge. (2004). *Panorama de la Industria del Software en Latinoamérica*. Brasil: Mayer&Bunge Informática LTDA. Retrieved December 14, 2007, http://www.mbi.com.br/200409_panorama_industria_software_america_latina.pdf

Oktaba, H., (2005). *Modelo de Procesos para la Industria de Software - MoproSoft - Versión 1.3, Agosto de 2005. NMX-059/01-NYCE-2005.* Ciudad de México: Organismo Nacional de Normalización y Evaluación de la Conformidad - NYCE. Retrieved December 14, 2007, from http://www.normalizacion-nyce.org.mx/php/loader.php?c=interes.php&tema=21

Pino, F., Garcia, F., Ruiz, F., & Piattini, M. (2006). A lightweight model for the assessment of software processes. In *Proceedings of the European Systems & Software Process Improvement and Innovation (EuroSPI 2006)*, Joensuu, Finland (pp. 7.1-7.12).

Printzell, C., & Conradi, R. (2001). *A taxonomy to compare SPI frameworks.* Paper presented at the Software Process Technology 8th European Workshop (EWSPT 2001), Witten, Germany (Vol. 2077, pp. 217-235). Springer.

Richardson, I. (2001, September). Software process matrix: A small company SPI model. *Software Process: Improvement and Practice, 6*(3), 157-165.

Scott, L., Jeffery, R., Carvalho, L., D'Ambra, J., & Rutherford, P. (2001). Practical software process improvement: The IMPACT Project. In *Proceedings of the Australian Software Engineering Conference* (pp. 182-189).

SEI. (2002). *CMMI for systems engineering/ software engineering* (Version 1.1). Pittsburgh: Software Engineering Institute (SEI). Retrieved December 14, 2007, from http://www.sei.cmu.edu/cmmi/

Storey, D. J. (1982). *Entrepreneurship and the new firm*. Croom Helm.

Wangenheim, C. G. v., Weber, S., Rossa Hauck, J. C., & Trentin, G. (2006, January). Experiences on establishing software processes in small companies. *Information and Software Technology,* pp. 1-11.

Weber, K., Araújo, E., Rocha, A., Machado, Scalet, D., & Salviano, C. (2005). Brazilian software process reference model and assessment method. *Computer and Information Sciences, 3733,* 402-411. Berlin/Heidelberg: Springer.

ENDNOTES

1. SIMEP_SW: An Integrated System for Software Process Improvement
2. COMPETISOFT (Process Improvement for Promoting Iberoamerican Software Small and Medium Enterprises Competitiveness) project financed by CYTED.

This work was previously published in Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies, edited by H. Oktaba; M. Piattini, pp. 177-192, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 8.9
Machine Learning and Value-Based Software Engineering

Du Zhang
California State University, USA

ABSTRACT

Software engineering research and practice thus far are primarily conducted in a value-neutral setting where each artifact in software development such as requirement, use case, test case, and defect, is treated as equally important during a software system development process. There are a number of shortcomings of such value-neutral software engineering. Value-based software engineering is to integrate value considerations into the full range of existing and emerging software engineering principles and practices. Machine learning has been playing an increasingly important role in helping develop and maintain large and complex software systems. However, machine learning applications to software engineering have been largely confined to the value-neutral software engineering setting. In this paper, the general message to be conveyed is to apply machine learning methods and algorithms to value-based software engineering. The training data or the background knowledge or domain theory or heuristics or bias used by machine learning methods in generating target models or functions should be aligned with stakeholders' value propositions. An initial research agenda is proposed for machine learning in value-based software engineering.

INTRODUCTION

Software engineering research and practice thus far are mainly conducted in a value-neutral setting where each artifact in software development such as a requirement, a use case, a test case, a defect, and so forth, is treated as equally important during a software system development process (Boehm, 2006a). There are a number of shortcomings of such value-neutral software engineering (Biffl et al. 2006): (1) its exclusion of economics, management sciences, cognitive sciences, and humanities from the body of knowledge needed to develop successful software systems;

(2) its delimitation of software development by mere technical activities; and (3) its failure to explicitly recognize the fact that software systems continue to satisfy and conform to evolving human and organizational needs is to create value. Value-based software engineering (VBSE) is to integrate value considerations into the full range of existing and emerging software engineering principles and practices so as to increase the return on investment (ROI = (benefits–costs)/costs) for the stakeholders and optimize other relevant value objectives of software projects (Biffl et al. 2006; Boehm, 2006a, Wang, 2007).

Machine learning (ML) has been playing an increasingly important role in helping develop and maintain large and complex software systems. However, machine learning applications to software engineering have been largely confined to the value-neutral software engineering setting (Zhang, 2000; Zhang & Tsai, 2003; Zhang & Tsai, 2005; Zhang & Tsai, 2007, Wang, 2008). In this paper, the general message we attempt to convey is to apply ML methods beyond the value-neutral software engineering setting and to VBSE. The training data or the background knowledge or domain theory or heuristics or bias used by ML methods in generating target models or functions for software development and maintenance should be aligned with stakeholders' value propositions (SVPs) and business objectives. Even though the transition to VBSE from the traditional value-neutral setting is necessarily evolutionary because not all the theories, infrastructures, methodologies and tools for VBSE have been fully developed yet, there are a number of agenda items for VBSE (Boehm, 2006a).

The goal of the road map in VBSE is to make software development and maintenance decisions that are better for value creation (Boehm, 2006a). On the other hand, the hallmark of ML is that it results in an improved ability to make better decisions. VBSE offers a fertile ground where many software development and maintenance tasks can be formulated as ML problems and approached in terms of ML methods. The purpose of this paper is to describe an initial research agenda for ML applications to VBSE with regard to the identified areas in VBSE (value-based requirement engineering, architecting, design and development, verification and validation, planning and control, risk/quality/people managements, and a theory of VBSE (Boehm, 2006a)).

The rest of the paper is organized as follows. Section 2 offers an overview of the related work. Section 3 highlights some important concepts in VBSE. In Section 4, we describe an initial research agenda for ML applications in VBSE. Finally Section 5 concludes the paper with remark on future work.

RELATED WORK

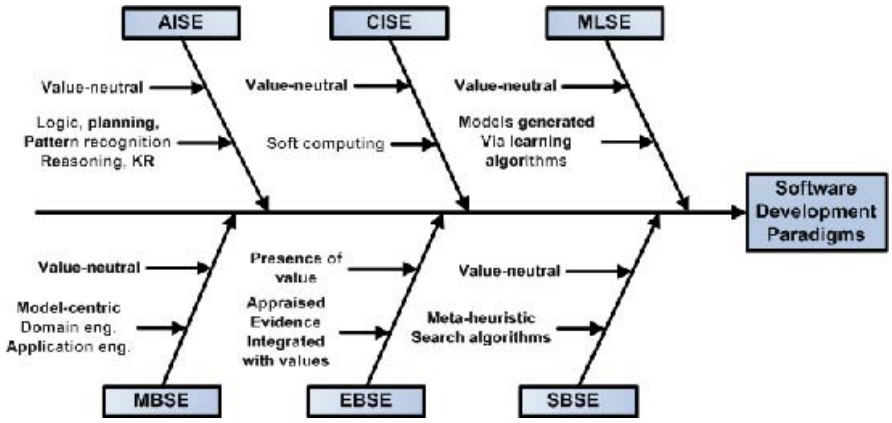
In this section, we provide a brief account for some of the major and emerging software development paradigms which are related to the main theme of this paper. The intent is to highlight the state-of-the-art in the software development landscape and to delineate differences between the existing approaches and the one advocated in this paper.

Besides machine learning in (value-neutral) software engineering (MLSE), there are a number of related and emerging software development paradigms: search-based software engineering (SBSE), evidence-based software engineering (EBSE), model-based software engineering (MBSE), artificial intelligence in software engineering (AISE), and computational intelligence in software engineering (CISE). Figure 1 highlights their similarities and differences.

MLSE

ML falls into the following broad categories: *supervised* learning, *unsupervised* learning, *semi-supervised* learning, *analytical* learning, *reinforcement* learning, and *multi-agent* learning.

Figure 1. Emerging software development paradigms



Each of the categories in turn includes various learning methods. Supervised learning deals with learning a target function from labeled examples. Unsupervised learning attempts to learn patterns and associations from a set of objects that do not have attached class labels. Semi-supervised learning is learning from a combination of labeled and unlabeled examples. Analytical learning relies on domain theory or background knowledge to learn a target function. Reinforcement learning is concerned with learning a control policy through reinforcement from an environment. Multi-agent learning is an extension to single-agent leaning. There are of course many emerging learning methods such as argument based machine learning, interactive learning, and so forth.

In software development, there are processes, products and resources (Fenton & Pfleeger, 1997). Processes are collections of software related activities, such as constructing specification, detailed design, or testing. Products refer to artifacts, deliverables, documents that result from a process activity, such as a specification document, a design document, or a segment of code. Resources are entities required by a process activity, such as personnel, software tools, or hardware. The aforementioned entities have internal and external attributes. Internal attributes describe an entity itself, whereas external attributes characterize the behavior of an entity (how the entity relates to its environment).

A partial list of ML applications in value-neutral software engineering includes (Zhang, 2000; Zhang & Tsai, 2003; Zhang & Tsai, 2005; Zhang & Tsai, 2007): (1) Predicting or estimating measurements for either internal or external attributes of software development processes, products, or resources (e.g., quality, size, cost, effort). (2) Discovering either internal or external properties of the processes, products, or resources (e.g., loop invariants, objects, normal operation boundary, equivalent mutants, process models, architecture information, aspects). (3) Transforming products to accomplish some desirable or improved external attributes (e.g., serial programs to parallel ones, improving software modularity). (4) Synthesizing or generating various products (e.g., evolutionary testing for

generating test data, project management rules and schedules, design repair knowledge, design schemas, programs/scripts/agents). (5) Reusing products or processes (e.g., similarity computing, locating/adopting software to specifications, generalizing program abstractions, clustering of components). (6) Enhancing processes (e.g., extracting specifications from software, acquiring specification consistent with scenarios). (7) Managing products (e.g., managing knowledge for software development process).

There were many different ML methods utilized in the aforementioned applications (instance-based learning and case-based reasoning, neural networks, decision trees, genetic algorithms, genetic programming, inductive logic programming, explanation-based learning, Bayesian learning, concept learning, analytic learning, support vector machines, multiple instance learning, active learning, clustering, association rules, and expectation maximization) (Zhang, 2000; Zhang & Tsai, 2003; Zhang & Tsai, 2005; Zhang & Tsai, 2007). A common property in the existing ML applications is that the software engineering issues were tackled solely from technical or logical perspectives (involving mappings and transformations, for instance) without the value dimension being taken into consideration (e.g., how to increase ROI for the stakeholders and optimize other relevant value objectives of software projects). The training data or the background knowledge or domain theory or heuristics or bias used by the ML methods in generating target functions did not contain any value propositions.

SBSE

SBSE treats software development tasks as a search problem with regard to a set of constraints and a search space of possible solutions (Clark et al, 2003; Harman & Jones, 2001). It relies on evolutionary algorithms, gradient ascent/descent, particle swarm intelligence, simulated annealing, tabu search or colony optimization techniques to tackle the software development or maintenance tasks. So far its applications have included the following areas in software engineering: requirement engineering, project planning, cost estimation, maintenance, reverse engineering, refactoring, program comprehension, service oriented tasks, quality assessment, and testing (structural, functional, non-functional, state-based properties, robustness, stress, security, mutation, regression, interaction, integration, and exception). Value considerations are not explicitly incorporated into the search process.

EBSE

EBSE is geared toward improving the decision making process related to software development and maintenance by integrating current best evidence from research with practical experience and human values (Dyba, Kitchenham & Jorgensen, 2005; Kitchenham, Dyba & Jorgensen, 2004). There are five main steps in EBSE as delineated in (Dyba, Kitchenham & Jorgensen, 2005; Kitchenham, Dyba & Jorgensen, 2004): (1) Translate a relevant problem or need of information into an answerable question. (2) Glean the literature for the best available evidence that can be used to answer the question. (3) Assess the evidence for its validity, impact, and applicability. (4) Combine the appraised evidence with practical experience, and stakeholders' values and circumstances to make decisions. (5) Evaluate performance and find ways to improve it. One important strength of EBSE is that it does take into consideration the SVPs.

MBSE

MBSE is centered on software models, modeling, and model transformation technologies. It is a disciplined approach to developing and extending a product family. The software models provide the necessary information to support, economically and effectively, future changes to a software

product family. By focusing on models that capture and consolidate developers' understanding of a family of software products, reusable assets can be developed that satisfy a wide variety of uses and can be utilized to analyze existing software to quickly compose or synthesize new solutions for subsequent products in a product family (Brown, Iyengar & Johnston, 2006; MBSE, 2008; Sendall & Kozacaynski, 2003). The goal is to achieve the benefits of reuse, shorter time to market, product maintainability and higher quality. However, value considerations are not prominently factored into the paradigm.

MBSE consists of two parallel engineering processes: domain engineering and application engineering, and sanctions the concepts of product families, a production system, and software assets (the reusable resources needed in application engineering such as domain models, software architectures, design standards, communication protocols, code components and application generators).

Domain engineering is a process of analysis, specification and implementation of software assets in a domain which are used in the development of multiple software products. Application engineering is a process that develops software products from software assets.

Many organizations have model-based development paradigm in place: Microsoft's Software Factory (Greenfield & Short, 2004), Lockheed Martin's Model Centric Software Development (Waddinggton & Lardieri 2006), and NASA JPL's Defect Detection and Prevention (Feather et al, 2008).

CISE

In CISE (or software engineering with computational intelligence), soft computing techniques such as fuzzy sets, neural networks, genetic algorithms, genetic programming and rough sets (or combinations of those individual technologies) are utilized to tackle software development issues recently [Khoshgoftaar, 2003a; Khoshgoftaar, 2003b; Lee, 2003a; Lee, 2003b; Pedrycz & Peters, 1998). The results have been largely confined to the value-neutral setting.

AISE

The application of some general artificial intelligence techniques to software engineering (AISE) has produced some encouraging results (Mendonca & Sunderhaft, 1999; Mostow, 1985; Partridge, 1998; Rich & Waters, 1986; Tsai & Weigert, 1993). Some of the successful AI techniques include: knowledge-based approach, automated reasoning, expert systems, heuristic search strategies, temporal logic, planning, and pattern recognition. Again the results thus far have been obtained in the value-neutral setting.

VBSE

The essence in VBSE is that the approach aims at aligning software development and maintenance with customer requirements and strategic business objectives. It offers a framework where SVPs are incorporated into the technical and managerial decisions made during software development and maintenance (Biffl et al. 2006; Grunbacher et al, 2006).

Value includes product, process and resource attributes. Value attributes include: profits (generated from products), strategic positioning in market share, utility, relative worth, reputation, customer loyalty, innovation technology, cost reduction, quality of life, improved productivity.

An emerging agenda of issues in VBSE has been proposed in (Boehm, 2006a), that includes the following areas:

- Value-based requirements engineering. The key objectives include recognition of success-critical stakeholders, elicitation of SVPs, and reconciliation of SVPs.

- Value-based architecturing. The goals are to iron out the discrepancy between a system's objectives and achievable architectural solutions.
- Value-based design and development. The goals are to ensure that a software system's objectives and its value considerations are embodied in the software's design and development practices.
- Value-based verification and validation. The objectives are to ascertain that a software solution meets its value objectives and that V&V tasks are sequenced and prioritized as investing activities.
- Value-based planning and control. The objectives in this area are to incorporate the value delivered to stakeholders into the product planning and control techniques.
- Value-based risk management. How to factor the value considerations into principles and practices for risk identification, analysis, prioritization, and mitigation is the main focus in this area.
- Value-based quality management. The goals are to prioritize desired software quality considerations with respect to SVPs.
- Value-based people management. The tasks involve building stakeholder team, manage expectations, and reconcile SVPs.

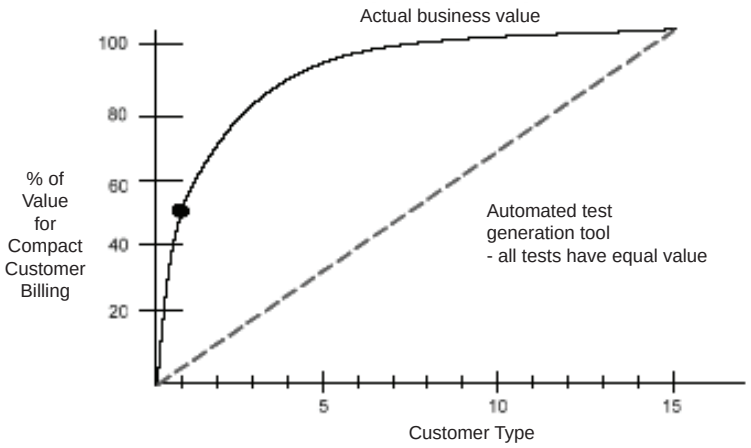
A number of concepts need to be in place to facilitate ML applications in VBSE. Of the important and useful concepts is the one about Pareto modules.

Figure 2 depicts a reported study in (Bullock, 2000) where the dotted line reflects the value-neutral practice in which an automated test data generation tool assumes that all tests have the same value. The Pareto curve for the empirical data, on the other hand, displays the actual business value where one of the fifteen customer services accounted for 50% of all billing revenues.

We refer to module(s) that realizes a service of such a high positive impact on the system's ROI as *Pareto modules.* They are the most important modules of a software system with regard to its product value. How modules contribute to a product's overall value hinges on reconciled SVPs.

For a given software system Ω, we can define a set $\mathbf{M}_{\Omega}$ of modules, a valuation function υ, and a value set $\boldsymbol{V}$ as follows:

Figure 2. Pareto distribution for varying test case value



$\mathbf{M}_\Omega = \{m_i \mid m_i \in \Omega\}$;
$\upsilon: \mathbf{M}_\Omega \rightarrow [0, 1]$;
$\boldsymbol{V} = \{\upsilon(m_i) \mid m_i \in \mathbf{M}_\Omega\}$

The valuation function υ can be defined by SVPs and has the following properties:

- $0 < \upsilon(m_i) \leq 1$, for all i;
- $\Sigma\, \upsilon(m_i) = 1$.

We define a partially ordered set $(\boldsymbol{V}, \leq)$ where $\leq$ is a binary order relation on $\boldsymbol{V}$ and satisfies reflexivity, anti-symmetry and transitivity for all elements in $\boldsymbol{V}$. We say that $\upsilon(m_p)$ is a *principal element* for $(\boldsymbol{V}, \leq)$ if we have the following:

$\forall \upsilon(m_j) \in \boldsymbol{V}\ [\upsilon(m_j) \leq \upsilon(m_p)]$[1]

We use ρ to denote the *principal module* as specified by $\upsilon(m_p)$. We define a *principal-element-ordered subset* $\boldsymbol{V}_{[mi, \rho]}$ of $\boldsymbol{V}$ and its cumulative value $\mu_{[mi, \rho]}$ as follows:

$\boldsymbol{V}_{[mi, \rho]} = \boldsymbol{V} - \{\upsilon(m_k) \mid \upsilon(m_k) \leq \upsilon(m_i)\}$
$\mu_{[mi, \rho]} = \Sigma\, \upsilon(m_j) \in \boldsymbol{V}_{[mi, \rho]}$

Now we are in a position to formally define the concept of Pareto modules.

Definition 1. *Given a threshold value* $\tau \in (0, 1]$, *we identify a principal-element-ordered subset* $\boldsymbol{V}_{[mi, \rho]}$ *such that* $\tau = \mu_{[mi, \rho]}$. *Modules in* $\boldsymbol{V}_{[mi, \rho]}$ *are referred to as Pareto modules with regard to* τ.

If $\tau < \mu_{[mi, \rho]}$ but removing any m_j from $\mu_{[mi, \rho]}$ would result in $\tau > \mu_{[mi, \rho]}$, then the condition of $\tau = \mu_{[mi, \rho]}$ is relaxed to that of $\tau \leq \mu_{[mi, \rho]}$.

In practice, to identify the Pareto modules, we can apply the Pareto principle as follows: (1) arrange modules according to the descending order of their value contributions to the total product value; (2) calculate cumulative percentages of contribution with regard to all the modules; (3) draw a line graph based on the cumulative percentages and modules involved; and (4) identify the most valuable modules with regard to an established threshold value.

The next set of useful concepts pertains to modules with varying defect densities. Item four on the software defect reduction top ten list in (Boehm & Basili, 2001) indicates that "*About 80% of the defects come from 20% of the modules, and about half the modules are defect free*." Thus, there are three types of modules here in terms of defect density measure: *defect-intensive modules* that refer to those 20% of the modules causing the bulk (80%) of defects, *defect-prone modules* that are the next 30% of modules containing the remaining 20% of defects, and *defect-free* modules that include the rest 50% of modules. In practice, there are different ways to specify the criteria for defect-intensive and defect-prone modules. Here we describe a possible way to define modules with varying defect densities that is based on the software Constructive Quality Model COQUALMO (Huang & Boehm, 2006; Steece, Chulani & Boehm, 2002).

In COQUALMO, there are two components: a defect introduction sub-model that estimates the rates at which software requirements, design and code defects are introduced, and a defect removal sub-model. Let KSLOC stand for thousand source line of code. The calibrated baseline (nominal) defect introduction rates DIR_{nom} in COQUALMO are given in Table 1 (Boehm et al, 2000; Huang & Boehm, 2006; Steece, Chulani & Boehm, 2002):

Thus the total of the nominal defect introduction rate for a software system is 60 defects/KSLOC. Multiplying the baseline rates with the size of a software system provides the total number of defects introduced in each of the three categories (requirements, design, and coding) and summing them up returns the total number of nominal defects introduced into a software system. We use $\mathbf{NDI}_\Omega$ to denote it[2]. Scaling $\mathbf{NDI}_\Omega$ down to the module level (e.g., if a module m has a size of 100 SLOC, then its nominal coding defects are

Table 1. Nominal defect introduction rates

Type of Defects	**DIR_{nom}**
Requirements defects	DIR_{nom}(req) = 10/KSLOC
Design defects	DIR_{nom}(des) = 20/KSLOC
Coding defects	DIR_{nom}(cod) = 30/KSLOC

scaled down to 3 accordingly), let $\mathbf{NDI}_m$ denote the nominal defects introduced into a module *m* and $\mathbf{TDI}_m$ the actual total number of defects in *m*. we can define defect-intensive and defect-prone modules using $\mathbf{NDI}_m$ as follows.

Definition 2. *A module m is defect-intensive if its defect introduction rates (in some or all three categories) are higher than the nominal rates DIR_{nom}. Therefore, we have* $\mathbf{TDI}_m > \mathbf{NDI}_m$.

Definition 3. *A module m is defect-prone if its defect introduction rates (in some or all three categories) are lower than the nominal rates DIR_{nom}. Therefore, we have* $\mathbf{TDI}_m \leq \mathbf{NDI}_m$.

There is another dimension about the nature of defects. Item five on the software defect reduction top ten list in (Boehm & Basili, 2001) states that "*About 90% of the downtime comes from, at most, 10% of the defects.*" We refer to those 10% of the defects as *impact defects*.

Definition 4. *Impact defects (or high risk defects) are those defects that result in loss of human life or high financial loss. This translates into the Required Reliability (RELY) ratings of Extra High, Very High and High according to the Constructive Cost Model (COCOMO) II (Boehm et al, 2000), and rough Mean Time Between Failures (MTBF) of one million hours, 300K hours and 10K hours, respectively (Boehm et al, 2004; Huang & Boehm, 2006).*

Definition 5. *Non-impact defects are those defects that result in moderate recoverable loss, easily recoverable loss or slight inconvenience. They have the RELY ratings of Nominal, Low and Very Low, and the MTBF of 300 hours, 10 hours and 1 hour, respectively (Huang & Boehm, 2006).*

The definitions of the aforementioned concepts will pave the way for ML to be utilized in various VBSE agenda issues. Further effort will be needed to fully develop concepts that accommodate comprehensive ML applications in VBSE.

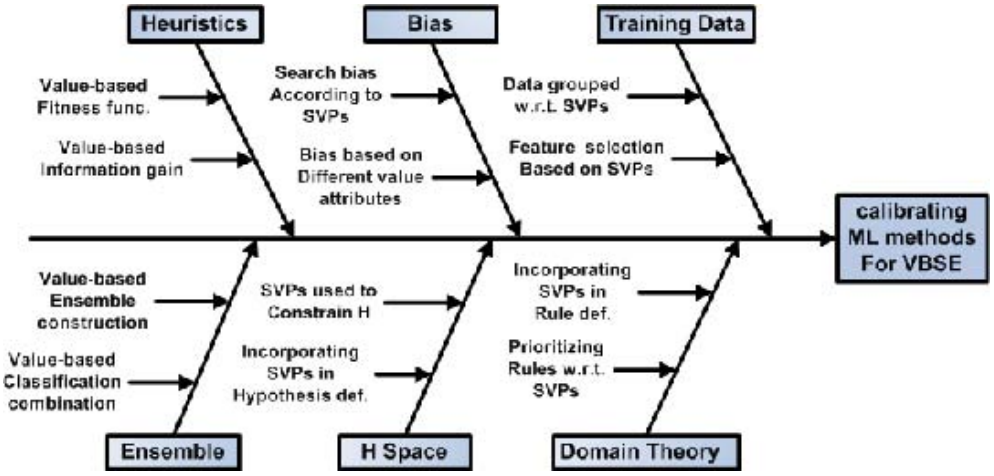
RESEARCH AGENDA FOR ML IN VBSE

In this section, we first discuss some general issues on how to calibrate ML methods for VBSE tasks. Using Boehm' VBSE agenda in (Boehm, 2006a) as a roadmap, we then describe some preliminary agenda items on how ML can help with the goals, objectives and tasks in VBSE.

How to Calibrate ML Methods

ML methods formulate various general hypotheses, models and target functions through either

Figure 3. Calibrating ML methods for VBSE



observed training data, or some background knowledge or domain theory, or a combination of both. The generalization process during leaning also hinges on certain adopted bias or heuristics.

To calibrate ML methods for VBSE tasks, the fundamental issue is how to incorporate SVPs from the business value level into the technical level details of ML model generation process. Specifically, this translates into the following issues: how to use SVPs to select data features and to group training data, how to incorporate SVPs into domain knowledge representation, how to prioritize rules, based on SVPs, in domain knowledge during model generation, how to include SVPs in defining search bias, how to use different value attributes in defining domain-specific biases for the search process, how to utilize SVPs in defining hypotheses and constraining hypothesis space, how to factor SVPs into ensemble construction and classification combination process when ensemble learning is used to generate models, and how the value concept plays a role in defining ML method-specific heuristics (e.g., fitness function, information gain measure).

Value-Based Requirements Engineering

For the objectives in value-based requirements engineering, techniques such as business case analysis, requirements prioritization and release prioritization have proven to be effective (Boehm, 2006a).

ML methods can be utilized to assist business case analysis, and requirements and release prioritization. Specifically, ML methods can be used to predict or estimate software cost, software size, software development efforts, and release prioritization and timing. These prediction, estimation or cost models would help stakeholders gain insight on what capabilities are not feasible with regard to budget, schedule and technology constraints, which features of a system are most important and attainable, and which aggregate of capabilities will meet stakeholders' critical needs

given the resource constraints. This in turn will assist stakeholders in prioritizing and reconciling potential conflicting value propositions. Possible ML methods for generating the models include: decision trees, Bayesian learning, neural networks, genetic algorithms, genetic programming, case-based reasoning, and inductive logic programming.

Value-Based Verification and Validation

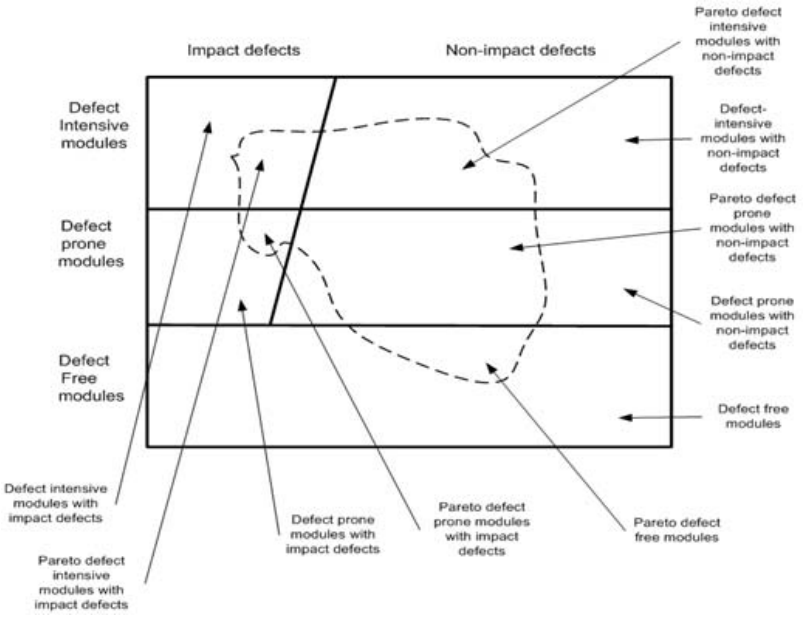
The key techniques in value-based V&V are value-based and risk-based testing techniques (Boehm, 2006a). Central to those techniques is how to align SVPs with the technical level details in test construction and execution and how to sequence and prioritize testing activities as investing activities (Ramler, Biffl & Grunbacher, 2006).

Since we have the following concepts in place: Pareto modules, defect-intensive and defect-prone modules, impact and non-impact defects (see Section 3), we can use an SVPs-based approach to identify modules of the aforementioned types and decompose the overall test data generation process for a given software system into a sequence of test data generation cycles with each focusing on some specific testing objective(s). Based on the types of modules and the types of defects, we have a landscape of possible testing scenarios as shown in Figure 4.

The dotted area indicates Pareto modules that can also be defect-intensive, defect-prone, or defect-free modules, and that can intersect with impact or non-impact defects.

From a value-based standpoint, to improve the return on investment, we want to make sure to first maximize the success rate of Pareto modules and

Figure 4. Possible testing scenarios



to minimize the chance of having impact defects that devastate the value contribution. Afterwards, attention can be focused on non-impact defects and non-Pareto modules. Thus, a prioritization of cycles can be generated that is driven by the value consideration and allows the most critical modules with regard to SVPs to be thoroughly tested first. For instance, we may have the following priority groups: (1) the top priority is to thoroughly test Pareto modules with impact defects; (2) the next priority is to devote attention to modules with impact defects; (3) the third priority group is to deal with Pareto modules with non-impact defects or defect-free Pareto modules; and (4) the last priority group consists of modules with non-impact defects or defect-free modules[3].

For each cycle, a number of ML methods can be utilized to generate test cases for different classes of modules. Some possible ML methods include: genetic algorithms (McMinn, 2004), genetic programming, inductive logic programming and rule-based active learning.

There are a number of issues here: how many cycles are needed, what goes to each cycle, and how cycles are prioritized.

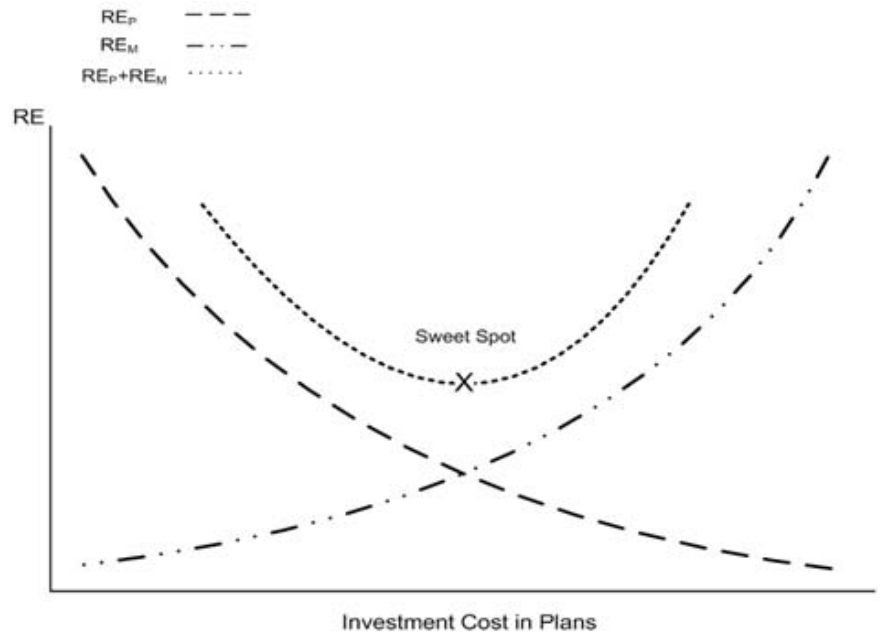
Value-Based Risk Management

There are a number of techniques for value-based risk management: the risk-based "how-much-is-enough" techniques, the risk-based analysis for project predictability, the risk-based simulation, and the risk-based testing techniques (Boehm, 2006a).

A pivotal concept in risk management is the *risk exposure* (***RE***) involved in a prescribed course of actions. ***RE*** is defined as follows:

$$\boldsymbol{RE} = \boldsymbol{P}(\mathcal{L}) \times \boldsymbol{S}(\mathcal{L})$$

Figure 5. Sweet spots for different risk exposure profiles



where $\boldsymbol{P}(\mathcal{L})$ is the probability of loss $\mathcal{L}$, and $\boldsymbol{S}(\mathcal{L})$ is the size of loss. $\mathcal{L}$ can be defined based on any value attribute as discussed in Section 3. In the risk exposure profile analysis (Boehm, 2006b), there is a dichotomy between planning and market share as the value attribute: inadequate planning results in little delay to capture market share but high ***RE*** due to oversights and rework (RE_P in Figure 5); excessive planning reduces the chance of major problems but at the expense of high ***RE*** because of time-to-market delays (RE_M in Figure 5).

ML methods can be used to help find the "sweet spot" for different risk profiles and different risk exposure profiles (see Figure 5). Depending on the circumstances, either inductive learning, or analytical learning, or a combination of inductive and analytical learning can be deployed.

Value-Based Design and Development

To ensure that a system's objectives and its value considerations are embodied in the software's design and development practices, the software traceability techniques play an important role (Boehm, 2006a). During the software development process, many artifacts are produced and maintained: documents, requirements, design models, test scenarios, and so forth. Trace dependencies are to identify relationships among those artifacts and the quality of the trace dependencies should reflect the value of the artifacts they attempt to bridge. This is vital for a number of reasons, from documentation, program understanding, impact analysis, consistency checking, reuse, quality assurance, user acceptance, error reduction, cost estimation, to customer satisfaction.

ML methods can be used to establish value-based trace dependencies among different artifacts. Methods such as instance-based learning (case-based reasoning), inductive logic programming, rule-based learning would lend themselves to the task.

Value-Based Quality Management

ML methods can be used to generate predictive models for identifying high risk or fault prone components as an integral part of the quality management. Because of the need to align desired quality properties with SVPs, value considerations should be, directly or indirectly, involved in defining or contributing to those quality properties. SVPs should also help prioritize the desired quality factors.

ML methods that are appropriate for the task include: decision trees, genetic programming, neural networks, case-based reasoning, inductive logic programming, and concept learning.

CONCLUSION

VBSE offers a new software development paradigm that recognizes the importance of business and customer value considerations. It tackles the decision making process in software development and maintenance from a value-based perspective. In this paper, we discuss the issue of ML applications to VBSE. Because ML applications to software engineering thus far have been largely confined to the value-neutral setting, we reviewed the landscape in the field and took a closer look at the emerging agenda for VBSE to find out how ML can be positioned to play a larger role in VBSE. We propose some guideline on how to calibrate ML methods to accommodate the value considerations that are so critical in accomplishing VBSE agenda items. Using Boehm' VBSE roadmap as a guide, we describe some preliminary agenda items on how ML can help with the goals, objectives and tasks in VBSE.

The take-home message of this work is two-fold: VBSE offers a ROI-conscious approach to software development and maintenance, and ML has an active and important role to play in various agenda items in VBSE.

The viability of ML applications in VBSE hinges on the outcomes of empirical studies, which will be pursued as our future work. How to solidify SVPs into various ML methods is an open issue worth studying.

REFERENCES

Biffl, S., et al. (2006). *Value-Based Software Engineering.* Berlin: Springer.

Boehm, B., et al. (2000). *Software Cost Estimation with COCOMO II.* New Jersey: Prentice Hall.

Boehm, B., & Basili, V. R. (2001). Software Defect Reduction Top 10 List. *IEEE Computer*, 34(1), 135-137.

Boehm, B., et al. (2004). The ROI of Software Dependability: the iDAVE Model. *IEEE Software*, 21(3), 54-61.

Boehm, B. (2006a). Value-Based Software Engineering: Overview and Agenda. In S. Biffl et al (Ed.), *Value-Based Software Engineering* (pp. 3-14). Berlin: Springer.

Boehm, B. (2006b). Value-Based Software Engineering: Seven Key Elements and Ethical Considerations. In S. Biffl et al (Ed.), *Value-Based Software Engineering* (pp. 109-132). Berlin: Springer.

Brown, A., Iyengar, S., & Johnston, S. (2006). A Rational Approach to Model-Based Development. *IBM Systems Journal*, 45(3), 463-480.

Bullock, J. (2000). Calculating the Value of Testing. *Software Testing and Quality Engineering*, May/June issue, 56-62.

Clark, J. et al. (2003). Reformulating Software Engineering as A Search Problem. *IEE Proceedings – Software*, 150(3), 161-175.

Dyba, T., Kitchenham, B. A., & Jorgensen, M. (2005). Evidence-Based Software Engineering for Practitioners. *IEEE Software*, 22(1), 58-65.

Feather, M., et al. (2008). A Broad, Quantitative Model for Making Early Requirements Decisions. *IEEE Software*, 25(2), 49-56.

Fenton, N. E., & Pfleeger, S. L. (1997). *Software Metrics.* Boston: PWS Publishing Company.

Greenfield J., & Short, K. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Indianapolis, IN: Wiley Publishing.

Grunbacher, P., Koszegi, S. & Biffl, S. (2006). Stakeholder Value Proposition Elicitation and Reconciliation. In S. Biffl et al (Ed.), *Value-Based Software Engineering* (pp. 133-154). Berlin: Springer.

Harman, M. & Jones, B. (2001). Search-Based Software Engineering. *Information and Software Technology*, 43(14), 833-839.

Huang, L., & Boehm, B. How Much Software Investment is Enough: A Value-Based Approach. *IEEE Software*, 23(5), 88-95.

Khoshgoftaar, T. (2003a). *Software Engineering with Computational Intelligence*, Berlin: Kluwer.

Khoshgoftaar, T. (2003b), Special Issue on Quality Engineering with Computational Intelligence. *Software Quality Journal*, 11(2).

Kitchenham, B. A., Dyba, T., & Jorgensen, M. (2004). Evidence-Based Software Engineering. In *Proceedings of International Conference on Software Engineering*, Edinburgh, pp.273-281.

Lee, J. (2003a). *Software Engineering with Computational Intelligence*, Berlin: Springer-Verlag.

Lee, J. (2003b). Special Issue on Software Eng with Computational Intelligence. *Information and Software Technology*, 45(7).

McMinn, P. (2004). *Search-based Software Test Data Generation: A Survey. Software: Testing, Verification and Reliability*, 14(2), 105-156.

Mendonca, M. & Sunderhaft, N. L. (1999). Mining Software Engineering Data: A Survey. DACS State-of-the-Art Report, http://www.dacs.dtic.mil/techs/datamining/.

Mostow, J. (1985). Special issue on artificial intelligence and software engineering. *IEEE Trans. SE*, 11(11), 1253-1408.

MBSE (accessed 2008). http://www.sei.cmu.edu/mbse/index.html.

Partridge, D. (1998). *Artificial Intelligence and Software Engineering*, Boston: AMACOM.

Pedrycz, W., & Peters, J. F. (1998). *Computational Intelligence in Software Engineering*. Singapore: World Scientific Publisher.

Ramler, R., Biffl, S., & Grunbacher, P. (2006). Value-Based Management of Software Testing. In S. Biffl et al (Ed.), *Value-Based Software Engineering* (pp. 225-244). Berlin: Springer.

Rich, C. & Waters, R. (1986). *Readings in Artificial Intelligence and Software Engineering*, San Francisco: Morgan Kaufmann.

Sendall, S. & Kozacaynski, W. (2003). Model Transformation: the Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5), 42-45.

Steece, B. Chulani, S., & Boehm, B. (2002). Determining Software Quality Using COQUALMO. In *Case Studies in Reliability and Maintenance*, W. Blischke and D. Murphy (Ed.), Hoboken, NJ: John Wiley & Sons.

Tsai, J. J. P. & Weigert, T. (1993). *Knowledge-Based Software Development for Real-Time Distributed Systems*, Singapore: World Scientific Pub.

Waddinggton, D. & Lardieri, P. (2006). Model-Centric Software Development. *IEEE Computer*, 39(2), 28-29.

Wang, Y. (2007). *Software Engineering Foundations: A Software Science Perspective*. CRC Book Series in Software Engineering, Vol.2. NY: Auerbach Publications.

Wang, Y. (2008). The Theoretical Framework and Cognitive Process of Learning, *ACM Transactions on Autonomous and Adaptive Systems*, 2(4), Dec.

Zhang, D. (2000). Applying Machine Learning Algorithms in Software Development. *Proceedings of Monterey Workshop on Modeling Software System Structures*, Santa Margherita Ligure, Italy, pp.275-285.

Zhang, D. & Tsai, J. J. P. (2003). Machine Learning and Software Engineering. *Software Quality Journal*, 11(2), 87-119.

Zhang, D. & Tsai, J. J. P. (2005). *Machine Learning Applications in Software Engineering*, Singapore: World Scientific Publishing Co.

Zhang, D. (2006). Machine Learning in Value-Based Software Test Data Generation. Proceedings of the *Eighteenth IEEE International Conference on Tools with AI*, Washington DC, pp.732-736.

Zhang, D. (2008). A Value-Based Framework for Software Evolutionary Testing. Submitted for publication.

Zhang, D. & Tsai, J. J. P. (2007). *Advances in Machine Learning Applications in Software Engineering*, Hershey, PA: Idea Group Publishing.

ENDNOTES

[1] If there are several principal elements in ***V***, we can use other criteria to designate one for the discussion.

[2] Even though a better estimate calls for adjusting the total defect number in each category with a different calibration constant and a different quality adjustment factor (an aggregate of 22 defect introduction drivers about the characteristics of platform, product, personnel and project) (Boehm, et al. 2000), for our purpose in identifying modules with different defect density, we simply use nominal defect introduction estimate as the measure.

[3] The reason to include defect-free modules (Pareto or non-Pareto) is because of the need to test for other performance related criteria.

This work was previously published in the International Journal of Software Science and Computational Intelligence, edited by Y. Wang, Volume 1, Issue 1, pp. 112-125, copyright 2009 by IGI Publishing (an imprint of IGI Global).

Chapter 8.10
An Operational Semantics of Real-Time Process Algebra (RTPA)

Yingxu Wang
University of Calgary, Canada

Cyprian F. Ngolah
University of Calgary, Canada, & University of Buea, Republic of Cameroon

ABSTRACT

The need for new forms of mathematics to express software engineering concepts and entities has been widely recognized. Real-time process algebra (RTPA) is a denotational mathematical structure and a system modeling methodology for describing the architectures and behaviors of real-time and nonreal-time software systems. This article presents an operational semantics of RTPA, which explains how syntactic constructs in RTPA can be reduced to values on an abstract reduction machine. The operational semantics of RTPA provides a comprehensive paradigm of formal semantics that establishes an entire set of operational semantic rules of software. RTPA has been successfully applied in real-world system modeling and code generation for software systems, human cognitive processes, and intelligent systems.

INTRODUCTION

Real-time process algebra (RTPA) is a denotational mathematical structure and a system modeling methodology for describing the architectures and behaviors of real-time and nonreal-time software systems (Wang, 2002, 2003, 2006a, 2006b, 2007a, 2008a-c). RTPA provides a coherent notation system and a rigorous mathematical structure for modeling software and intelligent systems. RTPA can be used to describe both *logical* and *physical* models of systems, where logic views of the architecture of a software system and its operational platform can be described using the

same set of notations. When the system architecture is formally modelled, the static and dynamic behaviors that perform on the system architectural model, can be specified by a three-level refinement scheme at the system, class, and object levels in a top-down approach. Although CSP (Hoare, 1978, 1985), the timed CSP (Boucher & Gerth, 1987; Fecher, 2001; Nicollin & Sifakis, 1991), and other process algebra (Baeten & Bergstra, 1991; Milner, 1980, 1989) treated a computational operation as a process, RTPA distinguishes the concepts of meta processes from complex and derived processes by algebraic process operations.

Definition 1: *Operational semantics of a programming language or a formal notation system is the semantics perceived on a given virtual machine, known as the abstract reduction machine, that denotes the semantics of programs or formal system models by its equivalent behaviors implemented on the reduction machine.*

One way to define an operational semantics for a language or formal notation system is to provide a state transition system for the language, which allows a formal analysis of the language and permits the study of relations between programs (Jones, 2003; Plotkin, 1981; Schneider, 1995). An alternative way is to describe the operations of the language on an abstract deductive machine whose operations are precisely defined (Sloneger & Barry, 1995; Winskel, 1993). In operational semantics, the reduction machine is a virtual machine that is adopted for reducing a given program to values of identifiers modeled in the machine by a finite set of permissible operations (Louden, 1993; McDermid, 1995).

This article presents a comprehensive operational semantics for RTPA on the basis of an abstract reduction machine, which defines inference rules for repetitively reducing a system model in RTPA into the computational values of identifiers and data objects. The abstract syntaxes of RTPA are introduced, and the reduction machine of RTPA is elaborated. Based on these, the operational semantics of 17 RTPA meta processes and 17 RTPA process relations are systematically developed. A comparative analysis of a set of comprehensive formal semantics for RTPA may be referred to (Wang, 2007a). The *deductive semantics* of RTPA is presented in Wang (2006a, 2008b). The *denotational semantics* of RTPA is reported in Tan and Wang (2008).

THE ABSTRACT SYNTAX OF RTPA

On the basis of the process metaphor of software systems, abstract processes can be rigorously treated as a mathematical entity beyond sets, relations, functions, and abstract concepts. RTPA is a denotational mathematical structure for denoting and manipulating system behavioral processes (Wang, 2002, 2003, 2006a, 2006b, 2008a-c). RTPA is designed as a coherent algebraic system for software and intelligent system modeling, specification, refinement, and implementation. RTPA encompasses 17 meta processes and 17 relational process operations.

Definition 2: *RTPA is a denotational mathematical structure for algebraically denoting and manipulating system behavioural processes and their attributes by a triple, that is:*

$$RTPA \triangleq (\mathfrak{T}, \mathfrak{P}, \mathfrak{R}) \quad (1)$$

where

- $\mathfrak{T}$ is a set of 17 primitive types for modeling system architectures and data objects;
- $\mathfrak{P}$ a set of 17 meta processes for modeling fundamental system behaviors;
- $\mathfrak{R}$ a set of 17 relational process operations for constructing complex system behaviors.

Detailed descriptions of $\mathfrak{T}$, $\mathfrak{P}$, and $\mathfrak{R}$ in RTPA will be extended in the following subsections (Wang, 2007a).

The Meta Processes of Software Behaviors in RTPA

RTPA adopts the foundationalism in order to elicit the most primitive computational processes known as the *meta processes*. In this approach, complex processes are treated as derived processes from these meta processes based on a set of algebraic process composition rules known as the *process relations*.

Definition 3: *A meta process in RTPA is a primitive computational operation that cannot be broken down to further individual actions or behaviors.*

A meta process is an elementary process that serves as a basic building block for modeling software behaviors. *Complex processes* can be composed from meta processes using *process relations*. In RTPA, a set of 17 meta processes has been elicited from essential and primary computational operations commonly identified in existing formal methods and modern programming languages (Aho, Sethi, & Ullman, 1985; Higman, 1977; Hoare et al., 1986; Louden, 1993;

Table 1. The meta processes of RTPA

No.	Meta Process	Notation	Syntax
1	Assignment	:=	$y\mathbb{T} := exp\mathbb{T}$
2	Evaluation	$\hat{}$	$\hat{}_{\mathbb{T}} exp\mathbb{T} \rightarrow \mathbb{T}$
3	Addressing	$\Rightarrow$	$id\mathbb{T} \Rightarrow \text{MEM}[ptr\mathbb{P}]\ \mathbb{T}$
4	Memory allocation	$\Leftarrow$	$id\mathbb{T} \Leftarrow \text{MEM}[ptr\mathbb{P}]\ \mathbb{T}$
5	Memory release	$\not\Leftarrow$	$id\mathbb{T} \not\Leftarrow \text{MEM}[\perp]\mathbb{T}$
6	Read	$\gtrdot$	$\text{MEM}[ptr\mathbb{P}]\mathbb{T} \gtrdot x\mathbb{T}$
7	Write	$\lessdot$	$x\mathbb{T} \lessdot \text{MEM}[ptr\mathbb{P}]\mathbb{T}$
8	Input	$\|\gtrdot$	$\text{PORT}[ptr\mathbb{P}]\mathbb{T} \|\gtrdot x\mathbb{T}$
9	Output	$\|\lessdot$	$x\mathbb{T} \|\lessdot \text{PORT}[ptr\mathbb{P}]\mathbb{T}$
10	Timing	$\overset{@}{=}$	$@t\mathbb{TM} \overset{@}{=} \S t\mathbb{TM}$ $\mathbb{TM} =$ yy:MM:dd | hh:mm:ss:ms | yy:MM:dd:hh:mm:ss:ms
11	Duration	$\triangleq$	$@t_n\mathbb{TM} \triangleq \S t_n\mathbb{TM} + \Delta n\mathbb{TM}$
12	Increase	$\uparrow$	$\uparrow(n\mathbb{T})$
13	Decrease	$\downarrow$	$\downarrow(n\mathbb{T})$
14	Exception detection	!	! (@*e*S)
15	Skip	$\otimes$	$\otimes$
16	Stop	$\boxtimes$	$\boxtimes$
17	System	§	§(*SysID*ST)

Wilson & Clark, 1988; Woodcock & Davies, 1996). Mathematical notations and syntaxes of the meta processes are formally described in Table 1.

Lemma 1: *The essential computing behaviours state that the RTPA meta process system $\mathfrak{P}$ encompasses 17 fundamental computational operations elicited from the most basic computing, that is:*

$$\mathfrak{P} = \{:=, \blacklozenge, \Rightarrow, \Leftarrow, \not\Leftarrow, \gg, \ll, |\gg, |\ll, @, \triangleq, \uparrow, \downarrow, !, \otimes, \boxtimes, \S\} \quad (2)$$

As shown in Lemma 1 and Table 1, each meta process is a basic operation on one or more operands such as variables, memory elements, or I/O ports. Structures of the operands and their allowable operations are constrained by their types as described in previous sections. It is noteworthy that not all generally important and fundamental computational operations, as shown in Table 1, had been explicitly identified in conventional formal methods. For instances, the evaluation, addressing, memory allocation/release, timing/duration, and the system processes. However, all these are found necessary and essential in modeling system architectures and behaviors.

Process Operations of RTPA

Definition 4: *A process relation in RTPA is an algebraic operation and a compositional rule between two or more meta processes in order to construct a complex process.*

A set of 17 fundamental process relations has been elicited from fundamental algebraic and relational operations in computing in order to build and compose complex processes in the context of real-time software systems. Syntaxes and usages of the 17 RTPA process relations are formally described in Table 2. Deductive semantics of these process relations may be referred to (Wang, 2006a, 2007a, 2008a).

Lemma 2: *The software composing rules state that the RTPA process relation system $\mathfrak{R}$ encompasses 17 fundamental algebraic and relational operations elicited from basic computing needs, that is:*

$$\mathfrak{R} = \{\rightarrow, \curvearrowright, |, |\ldots|\ldots, R^*, R^+, R^i, \circlearrowleft, \rightarrowtail, \|, \square, |||, », \nleftrightarrow, \hookrightarrow_t, \hookrightarrow_e, \hookrightarrow_i\} \quad (3)$$

The Type System of RTPA

A type is a set in which all member data objects share a common logical property or attribute. The maximum range of values that a variable can assume is a type, which is associated with a set of predefined or allowable operations. A type can be classified as *primitive* and *derived* (complex) types. The former is the most elemental types that cannot further divided into simpler ones; the latter is a compound form of multiple primitive types based on given type rules. Most primitive types are provided by programming languages; while most user defined types are derived ones.

Definition 5: *A type system specifies data object modeling and manipulation rules in computing.*

The 17 RTPA primitive types in computing and human cognitive process modeling have been elicited from works in (Cardelli & Wegner, 1985; Martin-Lof, 1975; Mitchell, 1990; Stubbs & Webre, 1985; Wang, 2002, 2003, 2007a), which is summarized in the following lemma.

Lemma 3: *The primary types of computational objects state that the RTPA type system $\mathfrak{T}$ encompasses 17 primitive types elicited from fundamental computing needs, that is:*

$$\mathfrak{T} \triangleq \{\mathsf{N}, \mathsf{Z}, \mathsf{R}, \mathsf{S}, \mathsf{BL}, \mathsf{B}, \mathsf{H}, \mathsf{P}, \mathsf{TI}, \mathsf{D}, \mathsf{DT}, \mathsf{RT}, \mathsf{ST}, @e\mathsf{S}, @t\mathsf{TM}, @int\odot, \circledS s\mathsf{BL}\} \quad (4)$$

where the primitive types stand for natural number, integer, real, string, Boolean, byte, hexadecimal,

Table 2. The process relations and algebraic operations of RTPA

	Process Relation	Notation	Syntax
1	Sequence	$\rightarrow$	$P \rightarrow Q$
2	Jump	$\curvearrowright$	$P \curvearrowright Q$
3	Branch	|	$\Box exp\mathsf{BL} = \mathsf{T} \rightarrow P$ $\mid \Box\sim \rightarrow Q$
4	Switch	| … |	$\Box \exp\mathbb{T} =$ $\mathrm{i} \rightarrow P_i$ $\mid \sim \rightarrow \oslash$ where $\mathbb{T} \in \{\mathsf{N}, \mathsf{Z}, \mathsf{B}, \mathsf{S}\}$
5	While-loop	R^*	$\overset{\mathsf{F}}{\underset{\exp\mathsf{BL}=\mathsf{T}}{R}} P$
6	Repeat-loop	R^+	$P \rightarrow \overset{\mathsf{F}}{\underset{\exp\mathsf{BL}=\mathsf{T}}{R}} P$
7	For-loop	R^i	$\overset{n_N}{\underset{i_N=1}{R}} P(i\mathsf{M})$
8	Recursion	$\circlearrowleft$	$\overset{0}{\underset{i_N=n_N}{R}} P^{i\mathsf{M}} \circlearrowleft P^{i\mathsf{M}-1}$
9	Function call	$\rightarrowtail$	$P \rightarrowtail F$
10	Parallel	||	$P \mid\mid Q$
11	Concurrence	$\Box$	$\mathrm{P} \Box Q$
12	Interleave	|||	$P \mid\mid\mid Q$
13	Pipeline	»	$P \gg Q$
14	Interrupt	$\lightning$	$P \lightning Q$
15	Time-driven dispatch	$\hookrightarrow_t$	$@t_i\mathsf{TM} \hookrightarrow_t P_i$
16	Event-driven dispatch	$\hookrightarrow_e$	$@e_i\mathsf{S} \hookrightarrow_e P_i$
17	Interrupt-driven dispatch	$\hookrightarrow_i$	$@int_j\circledcirc \hookrightarrow_i P_j$

pointer, time, date, date/Time, run-time determinable type, system architectural type, random event, time event, interrupt event, and status.

In Lemma 3, the first 11 primitive types are for mathematical and logical manipulation of data objects in computing, and the remaining six are for system architectural modeling. More rigorous description of RTPA type rules may be referred to (Wang, 2007a).

THE REDUCTION MACHINE OF RTPA

A reduction machine is an abstract machine that defines inference rules for repetitively reducing language constructs until a solid value or behavior is obtained. Reduction machines model the operational semantics of a given language or formal notation system in three components: *specification*, *control*, and *store*. In other words, an operational

semantics describes how the *control* of the machine reduces a given *specification* to values of variables and how the *memory (store)* is changed during the execution of the specification.

In the operational semantics approach, the underlying target machine that operates and implements the semantic rules is modeled by an abstract reduction machine. A program and its behavior space or the semantic environment are realized by the target computer. An abstract model of a generic reduction machine as the system platform for embodying software semantics can be modeled below.

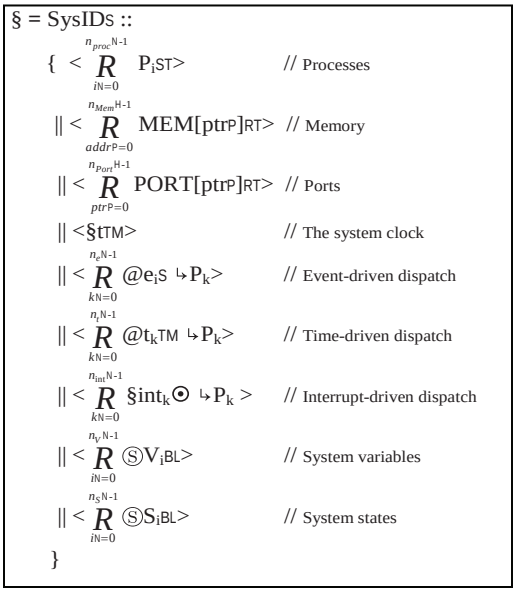
Definition 6: *The reduction machine, §, is an abstract logical model of the executing platform of a target machine denoted by a set of parallel or concurrent computing resources as shown in Figure 1.*

As shown in Figure 1, the reduction machine § for operational semantics is the executing platform that controls all the computing resources of an abstract target machine. The system is logically abstracted as a set of processes and underlying resources, such as the memory, ports, variables, statuses, and the system clock. A process is dispatched and controlled by the system §, which is triggered by various external, system timing, or interrupt events. The reduction machine of RTPA is not only the platform of the computing resources such as processes, memory, ports, and system clocks, but also the implementation of the computing mechanisms such as system dispatches, timing, interrupt handling, and system event captures.

Definition 7. *The semantic environment of the deduction machine, Θ, is the entire set of identifiers and their combinations declared and constrained in the abstract reduction machine, that is:*

$$\Theta \triangleq (I, T, V, A) = I \times T \times V \times A \quad (5)$$

Figure 1. The abstract model of the RTPA deduction machine



where I is a nonempty set of identifies, T is a set of types corresponding to each identifier in I, V is a set of values corresponding to each identifier in I, and A is a set of addresses corresponding to each identifier in I.

The semantic environment Θ can be divided into three subcategories known as the *operational environment* Θ_{op}, *system environment* Θ_{sys}, and *interrupt environment* Θ_{int}, that is:

$$\Theta \triangleq \Theta_{op} \cup \Theta_{sys} \cup \Theta_{int} \tag{6}$$

where only the operational environment Θ_{op} is controllable by users and applications.

Definition 8. *An inference rule in operational semantics, R_{os}, is a formal structure in which a propositional conclusion C is derived based on a set of given true premises P, that is:*

$$R_{OS} \triangleq \frac{\text{Premise(s)}}{\text{Conclusion}} = P \vdash C \tag{7}$$

where P and C are usually Boolean propositions. However, C can be a sequence of behavioral processes.

The semantic environment Θ forms the context of inference rules in operational semantics. Therefore, the following convention is adopted in all notations of inference rules:

$$< P \parallel \Theta > \Rightarrow < \varnothing \parallel \Theta' > \tag{8}$$

where ‖ denotes a parallel relationship between a *process* or an *expression P* and its underlying semantic environment Θ, ⇒ denotes a transition between a pair of semantic items, ∅ denotes an empty operation and/or the completion of a preceding process, and Θ' denotes an updated semantic environment as a result of the operation or effect of a process.

Definition 9. *The evaluations of an identifier id on its type t, value v, and address addr in Θ can be denoted as follows:*

$$\begin{aligned} t_{id} &\triangleq T(id \parallel \Theta) \\ v_{id} &\triangleq V(id \parallel \Theta) \\ addr_{id} &\triangleq A(id \parallel \Theta) \end{aligned} \tag{9.a-c}$$

where $id \in I \sqsubseteq \Theta$, or simply denoted $id \in \Theta$ when there is no confusion.

OPERATIONAL SEMANTICS OF RTPA META-PROCESSES

Using the reduction machine as defined in preceding section, the operational semantics for the 17 meta processes of RTPA can be described as follows.

Definition 10. *The reduction rule for the assignment process of RTPA, y*RT *:= exp*RT*, in operational semantics is shown in Box 1. The assignment rule indicates that an assignment transfers the value*

Box 1.

$$\frac{exp\text{RT}, y\text{RT} \in \Theta, \text{T}(exp\text{RT} \parallel \Theta) = \text{T}(y\text{RT} \parallel \Theta)}{<(y\text{RT} := exp\text{RT}) \parallel \Theta> \Rightarrow (< \text{V}(y\text{RT}) = \text{V}(exp\text{RT})) \parallel \Theta' >} \quad \text{(Rule 1)}$$

*of exp*RT *into y*RT *whenever both variables' types are identical or equivalent.*

Definition 11. *The reduction rule for the Boolean evaluation process of RTPA,* $\diamondsuit exp\mathbf{BL} \rightarrow \mathbf{BL}$*, in operational semantics is shown in Box 2, where | denotes a pair of alternative rules.*

The above rule for Boolean evaluation can be extended to more general cases where numeri-

Box 2.

$$\frac{exp\mathbb{BL}, \mathtt{T}, \mathtt{F} \in \Theta, <\diamondsuit exp\mathbb{BL} \parallel \Theta> \Rightarrow \mathrm{V}(exp\mathbb{BL}) = \mathtt{T} \parallel \Theta'_{sys}>}{<(\diamondsuit exp\mathbb{BL} \rightarrow \mathbb{BL}) \parallel \Theta> | \Rightarrow <exp\mathbb{BL} = \mathtt{T} \mid \Theta'>}$$
$$| \quad \frac{exp\mathbb{BL}, \mathtt{T}, \mathtt{F} \in \Theta, <\diamondsuit exp\mathbb{BL} \parallel \Theta> \Rightarrow \mathrm{V}(exp\mathbb{BL}) = \mathtt{F} \parallel \Theta'_{sys}>}{<(\diamondsuit exp\mathbb{BL} \rightarrow \mathbb{BL}) \parallel \Theta> | \Rightarrow <exp\mathbb{BL} = \mathtt{F} \mid \Theta'>}$$

(Rule 2.a)

Box 3.

$$\frac{exp\mathbb{T}, n\mathbb{T} \in \Theta, \mathbb{T} = \{\mathtt{N}, \mathtt{Z}, \mathtt{R}, \mathtt{B}\} \subset \Theta, <(\diamondsuit exp\mathbb{T} \rightarrow \mathbb{T}) \parallel \Theta> \Rightarrow <\mathrm{V}(exp\mathbb{T}) = n\mathbb{T} \parallel \Theta'_{sys}>}{<(\diamondsuit exp\mathbb{T} \rightarrow \mathbb{T}) \parallel \Theta'> \Rightarrow <exp\mathbb{T} = n\mathbb{T} \parallel \Theta'>}$$

(Rule 2.b)

Box 4.

$$\frac{id\mathbb{S}, ptr\mathbb{P} \in \Theta, <(id\mathbb{S} \Rightarrow ptr\mathbb{P}) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(id\mathbb{S} \Rightarrow ptr\mathbb{P}) \parallel \Theta> \mathrm{A} \Rightarrow <(id\mathbb{S}) = ptr\mathbb{P} \parallel \Theta'>}$$

(Rule 3)

Box 5.

$$\frac{id\mathbb{S}, ptr\mathbb{P}, n\mathbb{N} \in \Theta, <id\mathbb{S} \Leftarrow \mathrm{MEM}[ptr\mathbb{P}]\mathbb{RT}> \Rightarrow <\varnothing \parallel \Theta'>}{<(id\mathbb{S} \Leftarrow \mathrm{MEM}[ptr\mathbb{P}]\mathbb{RT}) \parallel \Theta> \Rightarrow <\mathrm{A}(id\mathbb{S}) = [ptr\mathbb{P}\mathbb{P}ptr + \mathrm{size}(\mathbb{RT})\text{-}1] \parallel \Theta'>}$$

(Rule 4)

cal evaluations are needed. The reduction rule for the *numerical evaluation process,* $\Diamond exp\mathbb{T} \rightarrow \mathbb{T}$, in operational semantics is shown in Box 3,where $\mathbb{T}$ is a numerical type, i.e., $\mathbb{T} = \{N, Z, R, B\} \subset T \sqsubseteq \Theta$.

It is noteworthy that, although the evaluation process does not affect Θ_{op}, but it changes Θ_{sys}.

Definition 12. *The reduction rule for the addressing process of RTPA, id*$S \Rightarrow ptrP$*, in operational semantics is shown in Box 4.*

Definition 13. *The reduction rule for the memory allocations process of RTPA,* $idS \Leftarrow MEM[ptrP]RT$*, in operational semantics is shown in Box 5, where size(RT) is the length of a certain type of variable, RT, in bytes, which is implementation specific.*

Definition 14. *The reduction rule for the memory release process of RTPA,* $idS \Leftarrow MEM[\perp]RT$*, in operational semantics is shown in Box 6, where* $\perp$ *denotes an empty or unassigned value.*

Definition 15. *The reduction rule for the read process of RTPA,* $MEM[ptrP]RT \geqslant xRT$*, in operational semantics isshown in Box 7.*

Definition 16. *The reduction rule for the write process of RTPA,* $MEM[ptrP]RT \leqslant xRT$*, in operational semantics is shown in Box 8.*

Box 6.

$$\frac{id S, ptrP \in \Theta, < idS \Leftarrow MEM[\perp]RT > \Rightarrow < \varnothing \parallel \Theta' >}{< (idS \Leftarrow MEM[\perp]RT) \parallel \Theta > \Rightarrow \{< MEM_{free} \cup MEM[A(idS), A(idS) + size(RT)\text{-}1] \parallel \Theta' > \rightarrow < (A(idS) = \perp) \parallel \Theta'' >\}}$$

(Rule 5)

Box 7.

$$\frac{xRT, ptrP, MEMRT \in \Theta, < (MEM[ptrP]RT \geqslant xRT) \parallel \Theta > \Rightarrow < \varnothing \parallel \Theta' >}{< (MEM[ptrP]RT \geqslant xRT) \parallel \Theta > \Rightarrow \{< (xRT = MEM[ptrP]RT) \parallel \Theta' >}$$

(Rule 6)

Box 8.

$$\frac{xRT, ptrP, MEMRT \in \Theta, < (MEM[ptrP]RT \leqslant xRT) \parallel \Theta > \Rightarrow < \varnothing \parallel \Theta' >}{< (MEM[ptrP]RT \leqslant xRT) \parallel \Theta > \Rightarrow \{< (MEM[ptrP]RT = xRT > \parallel \Theta' >}$$

(Rule 7)

Definition 17. *The reduction rule for the input process of RTPA,* PORT[*ptr*P]RT|≻ xRT, *in operational semantics is shown in Box 9.*

Definition 18. *The reduction rule for the output process of RTPA,* xRT|≺ PORT[*ptr*P]RT, *in operational semantics is shown in Box 10.*

Definition 19. *The reduction rule for the timing process of RTPA, @t*TM $\underline{@}$ §*t*TM, *in operational semantics is shown in Box 11.*

Definition 20. *The reduction rule for the duration process of RTPA,@t*TM$\underline{\Delta}$§*t*TM+Δ*d*Z, *in operational semantics is shown in Box 12.*

Box 9.

$$\frac{x\text{RT}, ptr\text{P}, \text{PORT}\text{RT} \in \Theta, <(\text{PORT}[ptr\text{P}]\text{RT} \mid\!\succ x\text{RT}) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(\text{PORT}[ptr\text{P}]\text{RT} \mid\!\succ x\text{RT}) \parallel \Theta> \Rightarrow \{<(x\text{RT} = \text{PORT}[ptr\text{P}]\text{RT}) \parallel \Theta'>} \quad \text{(Rule 8)}$$

Box 10.

$$\frac{x\text{RT}, ptr\text{P}, \text{PORT}\text{RT} \in \Theta, <(\text{PORT}[\text{ptr}\text{P}]\text{RT} \mid\!\prec \text{xRT}) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(\text{PORT}[\text{ptr}\text{P}]\text{RT} \mid\!\prec \text{xRT}) \parallel \Theta> \Rightarrow \{<(\text{PORT}[ptr\text{P}]\text{RT} = x\text{RT}> \parallel \Theta'>} \quad \text{(Rule 9)}$$

Box 11.

$$\frac{t\text{TM}, \S t\text{TM}, \Delta dz \in \Theta, <(@t\text{TM} = \S t\text{TM}) \parallel \Theta'>, \quad <(@t\text{TM} \underline{@} \S t\text{TM}) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <P \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(@t\text{TM} \underline{@} \S t\text{TM}) \parallel \Theta> \Rightarrow \{<(@t\text{TM}=\S t\text{TM}) \parallel \Theta'> \rightarrow <@t\text{TM} \hookrightarrow P \parallel \Theta">\}} \quad \text{(Rule 10)}$$

Box 12.

$$\frac{t\text{TM}, \S t\text{TM}, \Delta dz \in \Theta, <(t\text{TM} = \S t\text{TM}+\Delta dz) \parallel \Theta'>, \quad <(@\,t\text{TM} \underline{\Delta} \S t\text{TM}+\Delta dz) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <P \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(@t\text{TM} \underline{\Delta} \S t\text{TM}+\Delta dz) \parallel \Theta> \Rightarrow \{<(@t\text{TM}=\S t\text{TM}+\Delta dz) \parallel \Theta'> \rightarrow <@t\text{TM} \hookrightarrow P \parallel \Theta">\}} \quad \text{(Rule 11)}$$

Definition 21. *The reduction rule for the increase process of RTPA,* ↑(xRT)*, in operational semantics is shown in Box 13.*

Definition 22. *The reduction rule for the decrease process of RTPA,* ↓(xRT)*, in operational semantics is shown in Box 14.*

Definition 23. *The reduction rule for the exceptional detection process of RTPA,* !(@eS)*, in operational semantics is shown in Box 15, where CRT*ST *is the standard output device of the reduction machine for displaying system information in the type of strings.*

Definition 24. *The reduction rule for the skip process of RTPA,* ⊗*, in operational semantics is shown is:*

$$\frac{}{< \otimes \parallel \Theta > \Rightarrow \ < \varnothing \parallel \Theta'_{sys} >} \quad \text{(Rule 15)}$$

As defined above, the rule for the skip process of RTPA is an axiom, which does nothing functionally from users' point of view, but jumps to a new point of process by changing the control variables of program execution sequence in Θ'_{sys}.

Definition 25. *The reduction rule for the stop process of RTPA,* □*, in operational semantics is as follows:*

$$\frac{}{< \boxtimes \parallel \Theta > \Rightarrow \ < \varnothing \parallel \Theta' = \varnothing >} \quad \text{(Rule 16)}$$

The rule for the stop process in RTPA is an axiom, which terminates the current system and

Box 13.

$$\frac{xRT \in \Theta, RT \in \{N, B, H, Z, P, TM\} \subset \Theta, <(\uparrow (xRT)) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(\uparrow (xRT)) \parallel \Theta > \Rightarrow \{<(xRT = xRT + 1) \parallel \Theta' >} \quad \text{(Rule 12)}$$

Box 14.

$$\frac{xRT \in \Theta, RT \in \{N, B, H, Z, P, TM\} \subset \Theta, <(\downarrow (xRT)) \parallel \Theta > \Rightarrow <\varnothing \parallel \Theta'>}{<(\downarrow (xRT)) \parallel \Theta > \Rightarrow \{<(xRT = xRT - 1) \parallel \Theta' >} \quad \text{(Rule 13)}$$

Box 15.

$$\frac{eS, ptrP, CRTP \in \Theta, < ptrP = A(CRTST) \parallel \Theta' >, \quad <(!(@eS)) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(!(@eS)) \parallel \Theta > \Rightarrow \{<(PORT[ptrP]S = @eS) \parallel \Theta' >} \quad \text{(Rule 14)}$$

Box 16.

$$\frac{}{<\S(SysID\$) \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta' = (I \cup \{SysID\}),}$$
$$t(SysID) = \$, v(SysID) = \S = (\perp, \perp, \perp, \perp),$$
$$a(SysID) = A(SysID)>$$

(Rule 17)

releases all existing identifiers and their values in the environment.

Definition 26. *The reduction rule for the system process of RTPA, §(SysID$), in operational semantics is shown in Box 16.*

The rule for the system process in RTPA is an axiom, which does nothing functionally from users' point of view, but it creates the system identifier and allocates necessary resources to the newly created system.

OPERATIONAL SEMANTICS OF PROCESS RELATIONS

RTPA process relations are rules of algebraic operations of processes, which describe how the meta processes can be combined to form complex processes. The operational semantics of the 17 process relations of RTPA is elaborated in this section on the platform of the reduction machine.

Definition 27. *The reduction rule for the sequential process of RTPA, $P \rightarrow Q$, in operational semantics is shown in Box 17.*

Definition 28. *The reduction rule for the jump process of RTPA, $P \curvearrowright Q$, in operational semantics is shown in Box 18.*

Definition 29. *The reduction rule for the branch process of RTPA, $\diamond exp\$ \rightarrow P \mid \diamond\sim\rightarrow Q$, in operational semantics is shown in Box 19, where | denotes a pair of alternative sub-rules dependent on given conditions.*

Definition 30. *The reduction rule for the switch process of RTPA, $\diamond exp_i\$ \rightarrow P_i \mid \diamond\sim\rightarrow\otimes$, in operational semantics is shown in Box 20, where*

$$\mathop{\mathrm{R}}_{i=1}^{n}$$

denotes a set of recurrent structures.

The operational semantics of iterations are presented in the following definitions. It is noteworthy that iterations were diversely interpreted in literature (Louden, 1993; McDermid, 1991). Although the decision point may be denoted by branch constructs, most existing operational semantic rules failed to express the key semantics of "while" and the rewinding action of loops. Further, the semantics for more complicated types of iterations, such as the repeat-loop and for-loop, are rarely found in the literature.

Box 17.

$$\frac{<\mathrm{P} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <\mathrm{Q} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(P \rightarrow Q) \parallel \Theta> \Rightarrow \{<\mathrm{P} \parallel \Theta'> \rightarrow <\mathrm{Q} \parallel \Theta">\}} \quad \text{(Rule 18)}$$

Box 18.

$$\frac{<\mathrm{P} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <\mathrm{Q} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<(P \curvearrowright Q) \parallel \Theta> \Rightarrow \{<\mathrm{P} \parallel \Theta'> \rightarrow <\varnothing \parallel \Theta' \cup \Theta'_{sys}> \rightarrow <\mathrm{Q} \parallel \Theta" \cup \Theta'_{sys}>\}} \quad \text{(Rule 19)}$$

Box 19.

$$\begin{array}{l} \dfrac{exp_{\mathrm{BL}} \in \Theta, <exp_{\mathrm{BL}} \parallel \Theta> \Rightarrow exp_{\mathrm{BL}} = \mathrm{T}, <P \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<\diamond exp_{\mathrm{RT}} \rightarrow P \mid \diamond\sim \rightarrow Q> \Rightarrow <P \parallel \Theta'>} \\ \mid \dfrac{exp_{\mathrm{BL}} \in \Theta, <exp_{\mathrm{BL}} \parallel \Theta> \Rightarrow exp_{\mathrm{BL}} = \mathrm{F}, <Q \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<\diamond exp_{\mathrm{RT}} \rightarrow P \mid \diamond\sim \rightarrow Q> \Rightarrow <Q \parallel \Theta'>} \end{array} \quad \text{(Rule 20)}$$

Box 20.

$$\begin{array}{l} i\mathrm{N}, n\mathrm{N} \in \Theta, <exp_{\mathrm{RT}} \parallel \Theta> \Rightarrow exp_{\mathrm{RT}} = i\mathrm{N}, 1 \le i\mathrm{N} \le n\mathrm{N}, \\ <\mathop{\mathrm{R}}_{i\mathrm{N}=1}^{n\mathrm{N}} P \parallel \Theta> \Rightarrow <P_i \parallel \Theta'> \\ <\diamond exp_{\mathrm{i}\,\mathrm{RT}} \rightarrow \mathrm{P_i} \mid \diamond\sim\rightarrow \otimes> \Rightarrow <P_i \parallel \Theta'> \\ i\mathrm{N}, n\mathrm{N} \in \Theta, <exp_{\mathrm{RT}} \parallel \Theta> \Rightarrow exp_{\mathrm{RT}} = i\mathrm{N}, i\mathrm{N} \notin [1, n\mathrm{N}], \\ \mid \dfrac{<\mathop{\mathrm{R}}_{i\mathrm{N}=1}^{n\mathrm{N}} P \parallel \Theta> \Rightarrow <\otimes \parallel \Theta>}{<\diamond exp_{\mathrm{i}\,\mathrm{RT}} \rightarrow \mathrm{P_i} \mid \diamond\sim\rightarrow \otimes> \Rightarrow <\otimes \parallel \Theta'_{sys}>} \end{array} \quad \text{(Rule 21)}$$

Definition 31. *The reduction rule for the while-loop process of RTPA,*

$$\underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P,$$

in operational semantics is shown in Box 21, where the while-loop is defined recursively on Θ.

The above rule indicates that, when a Boolean expression *exp*$\mathbb{BL}$ in the environment Θ is true, the execution of the loop body *P* as a process for an iteration under Θ can be reduced to the same loop under an updated environment Θ', which is resulted by the last execution of *P*; When *exp*$\mathbb{BL} = \mathbb{F}$ in Θ, the loop is reduced to a termination or exit $\otimes$.

Definition 32. *The reduction rule for the repeat-loop process of RTPA,*

$$P \rightarrow \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P,$$

in operational semantics is shown in Box 22.

The above rule indicates that the semantics of a repeat-loop process is semantically equivalent to the sequential composition of *P* and a while-loop.

Box 21.

$$\frac{exp\mathbb{BL} \in \Theta, \langle exp\mathbb{BL} \parallel \Theta \rangle \Rightarrow exp\mathbb{BL} = \mathbb{T}, \langle \mathbb{P} \parallel \Theta \rangle \Rightarrow \langle \varnothing \parallel \Theta' \rangle}{\langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta \rangle \Rightarrow \{\langle \mathbb{P} \parallel \Theta' \rangle \rightarrow \langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta' \rangle\}}$$

$$\mid \frac{exp\mathbb{BL} \in \Theta, \langle exp\mathbb{BL} \parallel \Theta \rangle \Rightarrow exp\mathbb{BL} = \mathbb{F}, \langle \otimes \parallel \Theta'_{sys} \rangle}{\langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta \rangle \Rightarrow \langle \otimes \parallel \Theta'_{sys} \rangle}$$

(Rule 22)

Box 22.

$$\frac{exp\mathbb{BL} \in \Theta, \langle \mathbb{P} \parallel \Theta \rangle \Rightarrow \langle \varnothing \parallel \Theta \rangle}{\langle P \rightarrow \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta \rangle \Rightarrow \{\langle \mathbb{P} \parallel \Theta' \rangle \rightarrow$$

$$(\frac{\langle exp\mathbb{BL} \parallel \Theta' \rangle \Rightarrow exp\mathbb{BL} = \mathbb{T}, \langle \mathbb{P} \parallel \Theta' \rangle \Rightarrow \langle \varnothing \parallel \Theta'' \rangle}{\langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta' \rangle \Rightarrow \{\langle \mathbb{P} \parallel \Theta'' \rangle \rightarrow \langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta'' \rangle\}}$$

$$\mid \frac{\langle exp\mathbb{BL} \parallel \Theta' \rangle \Rightarrow exp\mathbb{BL} = \mathbb{F}, \langle \otimes \parallel \Theta'_{sys} \rangle}{\langle \underset{exp\mathbb{BL}=\mathbb{T}}{\overset{\mathbb{F}}{\mathbb{R}}} P \parallel \Theta' \rangle \Rightarrow \langle \otimes \parallel \Theta'_{sys} \rangle}$$

$$) \}$$

(Rule 23)

Definition 33. *The reduction rule for the for-loop process of RTPA,*

$$\mathop{\mathrm{R}}_{i\mathbb{N}=1}^{n} P(i\mathbb{N}),$$

in operational semantics is shown in Box 23.

The above rule indicates that the semantics of a for-loop process is semantically equivalent to a sequence of *n* serial processes. The semantic rule of for-loop processes may also be defined recursively as that of the while-loop rule as shown in Box 24.

Definition 34. *The reduction rule for the function call process of RTPA, $P \rightarrowtail F$, in operational semantics is shown in Box 25.*

Definition 35. *The reduction rule for the recursion process of RTPA, $P \circlearrowleft P$, in operational semantics is shown in Box 26.*

The semantic rule of recursion processes may also be defined iteratively as that of the for-loop rule as shown in Box 27, where the base process P^0 should be able to be reduced to a constant.

Box 23.

$$\frac{i\mathbb{N},\ n\mathbb{N} \in \Theta, 1 \le i\mathbb{N} \le n\mathbb{N},\ <\mathrm{P}_i \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<\mathop{\mathrm{R}}_{i\mathbb{N}=1}^{n} P(i\mathbb{N}) \parallel \Theta> \Rightarrow \{<P(i\mathbb{N}=1) \parallel \Theta'> \rightarrow <P(i\mathbb{N}=2) \parallel \Theta''> \rightarrow \ldots \rightarrow <P(i\mathbb{N}=n\mathbb{N}) \parallel \Theta^{n'}>\}}$$

(Rule 24)

Box 24.

$$\frac{<\blacklozenge(1 \le i\mathbb{N} \le n)\mathbb{BL} \parallel \Theta> \Rightarrow \mathbb{T}, <\mathrm{P}_i \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<\mathop{\mathrm{R}}_{i\mathbb{N}=1}^{n} P \parallel \Theta> \Rightarrow \{<\mathrm{P}_i \parallel \Theta'> \rightarrow <i\mathbb{N} = i\mathbb{N}+1> \rightarrow <\mathop{\mathrm{R}}_{i\mathbb{N}=1}^{n} P \parallel \Theta'>\}}$$

$$\mid \frac{<\blacklozenge(1 \le i\mathbb{N} \le n)\mathbb{BL} \parallel \Theta> \Rightarrow \mathbb{F}, <\otimes \parallel \Theta'_{sys}>}{<\mathop{\mathrm{R}}_{i\mathbb{N}=1}^{n} P \parallel \Theta> \Rightarrow <\otimes \parallel \Theta'_{sys}>}$$

(Rule 25)

Box 25.

$$\frac{<\mathrm{P} \parallel \Theta> \Rightarrow \{<\mathrm{P}' \parallel \Theta'> \rightarrow <\mathrm{P}'' \parallel \Theta''>\},\ <\mathrm{F} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>}{<P \rightarrowtail F \parallel \Theta> \Rightarrow \{<\mathrm{P}' \parallel \Theta'> \rightarrow <\mathrm{F} \parallel \Theta''> \rightarrow <\mathrm{P}'' \parallel \Theta'''>\}}$$

(Rule 26)

Definition 36. *The reduction rule for the parallel process of RTPA, P || Q, in operational semantics is shown in Box 28, where S_1 and S_2 are two additional synchronization processes introduced by the system.*

The parallel process rule models the process relation with the single-clock multi-processor (SCMP) structure. In other words, it behaves in a synchronized system or a common environment.

Definition 37. *The reduction rule for the concurrent process of RTPA, P ⨌ Q, in operational semantics is shown in Box 29, where C_1 and C_2 are two additional communication processes introduced in two separated system environments Θ_P and Θ_Q .*

The concurrent processes model the process relation with the multi-clock multi-processor (MCMP) structure. In other words, it behaves in an asynchronized system or a separated environments linked by the communication means.

Definition 38. *The reduction rule for the interleave process of RTPA, P ||| Q, in operational semantics is shown in Box 30.*

Box 26.

$$\frac{<P \parallel \Theta> \Rightarrow \{<P' \parallel \Theta' > \rightarrow <P'' \parallel \Theta''>\}, <P \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta' >}{<P \circlearrowleft P \parallel \Theta > \{\Leftrightarrow \ <P' \parallel \Theta' > \rightarrow <P \circlearrowleft P \parallel \Theta''> \rightarrow <P'' \parallel \Theta'''>\}}$$

(Rule 27)

Box 27.

$$\frac{i\mathbb{N}, n\mathbb{N} \in \Theta, 1 \le i\mathbb{N} \le n\mathbb{N}, <P^{i\mathbb{N}} \parallel \Theta> \Rightarrow <P^{i\mathbb{N}-1} \parallel \Theta'>, P^0 = const\mathbb{N}}{<P \circlearrowleft P \parallel \Theta > (\Rightarrow \overset{1}{\underset{i\mathbb{N}=n\mathbb{N}}{\mathrm{R}}} < \ P^{i\mathbb{N}} \rightarrow P^{i\mathbb{N}-1}) \parallel \Theta^i >}$$

(Rule 28)

Box 28.

$$\frac{\begin{array}{c}<P \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <Q \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>, <S_1 \parallel \Theta> \Rightarrow \\ <\varnothing \parallel \Theta>, <S_2 \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta>\end{array}}{\begin{array}{l}<(P \parallel Q) \parallel \Theta> \ \Rightarrow \ \{<S_1 \parallel \Theta_{sys}> \rightarrow \left\langle \begin{array}{l} \rightarrow <P \ \parallel \Theta'> \rightarrow \\ \rightarrow <Q \parallel \Theta''> \rightarrow \end{array} \right\rangle \rightarrow \\ <S_2 \parallel \Theta' \cup \Theta'' \cup \Theta'_{sys}>\}\end{array}}$$

(Rule 29)

Box 29.

$$\frac{\begin{array}{c}<\mathrm{P} \parallel \Theta_P> \Rightarrow <\varnothing \parallel \Theta'_P>, <\mathrm{Q} \parallel \Theta_Q> \Rightarrow <\varnothing \parallel \Theta'_Q>, \\ <\mathrm{C}_1 \parallel \{\Theta_P \parallel \Theta_Q\}> \Rightarrow <\varnothing \parallel \{\Theta_P \parallel \Theta_Q\}>, \\ <\mathrm{C}_2 \parallel \{\Theta_P \parallel \Theta_Q\}> \Rightarrow <\varnothing \parallel \{\Theta_P \parallel \Theta_Q\}>\end{array}}{\begin{array}{l}<(P \oiint Q) \parallel \{\Theta_P \parallel \Theta_Q\}> \Rightarrow \{<\mathrm{C}_1 \parallel \{\Theta_P \parallel \Theta_Q\}> \\ \quad \rightarrow \left\langle \begin{array}{l} \rightarrow <\mathrm{P} \parallel \Theta'_P> \rightarrow \\ \rightarrow <\mathrm{Q} \parallel \Theta'_Q> \rightarrow \end{array} \right\rangle \rightarrow <\mathrm{C}_2 \parallel \{\Theta'_P \parallel \Theta'_Q\}>\}\end{array}}$$

(Rule 30)

Box 30.

$$\frac{\begin{array}{c}\mathrm{iN}, \mathrm{nN} \in \Theta, 1 \le \mathrm{iN} \le \mathrm{nN}, <\mathrm{P} \parallel \Theta> \Rightarrow \mathop{\mathrm{R}}_{\mathrm{iN}=1}^{\mathrm{nN}} <P_i \parallel \Theta_i>, \\ <\mathrm{Q} \parallel \Theta> \Rightarrow \mathop{\mathrm{R}}_{\mathrm{iN}=1}^{\mathrm{nN}} <\mathrm{Q}_i \parallel \Theta_i>\end{array}}{<(P \parallel\!\mid Q) \parallel \Theta> \Rightarrow \mathop{\mathrm{R}}_{\mathrm{iN}=1}^{\mathrm{nN}} (<P_i \parallel \Theta_i> \rightarrow <Q_i \parallel \Theta'_i>)}$$

(Rule 31)

Box 31.

$$\frac{\begin{array}{c}\mathrm{O_P}, \mathrm{I_Q}, \mathrm{iN}, \mathrm{nN} \in \Theta, \mathop{\mathrm{R}}_{\mathrm{iN}=1}^{\mathrm{nN}} \mathrm{o_P} = \mathrm{i_Q}, \mathrm{o_P} \in \mathrm{O_P}, \mathrm{i_Q} \in \mathrm{I_Q}, \#\mathrm{O_P} = \#\mathrm{I_Q}, \\ <\mathrm{P} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>\}, <\mathrm{Q} \parallel \Theta> \Rightarrow <\varnothing \parallel \Theta'>\end{array}}{<P \gg Q \parallel \Theta> \Rightarrow \{<P \parallel \Theta'> \rightarrow <Q \parallel \Theta''>\}}$$

(Rule 32)

The rule of interleave processes models the process relation with the single-clock single-processor (SCSP) structure in a synchronized environment.

Definition 39. *The reduction rule for the pipeline process of RTPA, $P \gg Q$, in operational semantics is shown in Box 31.*

The rule of pipeline processes shows that from the functional point view, a pipeline process rela-

tion is equivalent to a sequential relation as long as the corresponding outputs O_P and inputs I_Q are one-to-one coupled between the two processes.

Definition 40. *The reduction rule for the interrupt process of RTPA, $P \leftarrow Q$, in operational semantics is shown in Box 32, where the interrupt semantic environment Θ_{int} is a subset of Θ as defined in Equation 6.*

The rule of interrupt processes shows that the main environment Θ is protected when an interrupt occurs. However, the interrupt subroutine Q may affect Θ via global or shared variables and data structures after its completion.

Definition 41. *The reduction rule for the time-driven process of RTPA, $@t_k \mathsf{TM} \hookrightarrow P_k$, in operational semantics is shown in Box 33.*

The rule of time-driven processes models a top level system dispatching behavior where the system transfers control to a process P_k after capturing a corresponding timing event $@t_k \mathsf{TM}$;

Box 32.

$$\frac{\begin{array}{c}<\mathrm{P} \parallel \Theta> \Rightarrow \{<\mathrm{P}' \parallel \Theta'> \rightarrow <\mathrm{P}'' \parallel \Theta''>\}, \\ <\mathrm{Q} \parallel \Theta_{int} \subset \Theta> \Rightarrow <\varnothing \parallel \Theta'_{int}>\end{array}}{\begin{array}{l}<(P \leftarrow Q) \parallel \Theta> \Rightarrow \{<\mathrm{P}' \parallel \Theta'> \rightarrow \\ \qquad <\mathrm{Q} \parallel \Theta'_{int}> \rightarrow \\ \qquad <\mathrm{P}'' \parallel \Theta'' \cup \Theta'_{int}>\}\end{array}} \quad \text{(Rule 33)}$$

Box 33.

$$\frac{\begin{array}{c}k\mathsf{N},\ n\mathsf{N}, t_k \mathsf{TM} \in \Theta, 1 \le k\mathsf{N} \le n\mathsf{N}, \\ <\mathrm{P} \parallel \Theta> \Rightarrow \mathop{\mathrm{R}}_{k\mathrm{N}=1}^{n\mathrm{N}} <P_k \parallel \Theta'>, <\S \parallel \Theta> \Rightarrow <\S \parallel \Theta'>\end{array}}{<(@t_k \mathsf{TM} \hookrightarrow \mathrm{P}_k) \parallel \Theta> \Rightarrow \{<\S \parallel \Theta> \rightarrow <P_k \parallel \Theta'> \rightarrow <\S \parallel \Theta''>\}} \quad \text{(Rule 34)}$$

Box 34.

$$\frac{\begin{array}{c}k\mathsf{N},\ n\mathsf{N}, e_k \mathsf{S} \in \Theta, 1 \le k\mathsf{N} \le n\mathsf{N}, \\ <\mathrm{P} \parallel \Theta> \Rightarrow \mathop{\mathrm{R}}_{k\mathrm{N}=1}^{n\mathrm{N}} <P_k \parallel \Theta'>, <\S \parallel \Theta> \Rightarrow <\S \parallel \Theta'>\end{array}}{<(@e_k \mathsf{S} \hookrightarrow \mathrm{P}_k) \parallel \Theta> \Rightarrow \{<\S \parallel \Theta> \rightarrow <P_k \parallel \Theta> \rightarrow <\S \parallel \Theta''>\}} \quad \text{(Rule 35)}$$

Box 35.

$$k \in \mathbb{N},\ n \in \mathbb{N},\ int_k \odot \in \Theta, 1 \le k \le n \in \mathbb{N},$$

$$\frac{<P \parallel \Theta_{int}> \Rightarrow \mathop{\mathrm{R}}_{k=1}^{n} <P_k \parallel \Theta'_{int}>, <\S \parallel \Theta> \Rightarrow <\S \parallel \Theta'>}{<(@int_k \odot \hookrightarrow P_k) \parallel \Theta> \Rightarrow \{<\S \parallel \Theta> \rightarrow <P_k \parallel \Theta'_{int}> \rightarrow <\S \parallel \Theta \cup \Theta'_{int}>\}}$$

(Rule 36)

upon its completion, it returns the control of system resources and the environment to the system.

Definition 42. *The reduction rule for the event-driven process of RTPA, $@e_k \mathbb{M} \hookrightarrow P_k$, in operational semantics is shown in Box 34.*

The rule of event-driven processes models the second type of top level system dispatching behaviors, where the system transfers control to a process P_k after capturing a corresponding event $@e_k \mathbb{S}$; upon its completion, it returns the control of system resources and the environment to the system.

Definition 43. *The reduction rule for the interrupt-driven process of RTPA, $@int_k \odot \hookrightarrow P_k$, in operational semantics is shown in Box 35.*

The rule of interrupt-driven processes models the third type of top level system dispatching behaviors, where the system transfers the control to an interrupt subroutine P_k after capturing a corresponding interrupt event $@int_k \odot$; upon its completion, it returns the control of system resources and the environment to the system.

CONCLUSION

The operational semantics of Real-Time Process Algebra (RTPA) has been developed in this article, which explains how syntactic constructs in RTPA can be reduced to values on an abstract reduction machine. The operational semantics of RTPA has provided a comprehensive paradigm of formal semantics, which extends the conventional express power of operational semantics to an entire set of semantic rules for complicated RTPA process structures and their algebraic operations. Especially, the operational semantics of the parallel, concurrent, and system dispatches have been formally and systematically elaborated.

RTPA has been presented as both a denotational mathematical structure and a system modeling methodology for describing the architectures and behaviors of real-time and nonreal-time software systems. The formal semantics of RTPA has helped to the comprehension and understanding of the RTPA syntactical and semantic rules as well as its expressive power in software engineering, cognitive informatics, and computational intelligence. RTPA has been used not only in software system specifications, but also in human and intelligent system modeling.

ACKNOWLEDGMENT

The authors would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. We would like to thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Aho, A.V., Sethi, R., & Ullman, J.D. (1985). *Compilers: Principles, techniques, and tools.* New York: Addison-Wesley Publication Co.

Baeten, J.C.M. & Bergstra, J.A. (1991). Real time process algebra. *Formal Aspects of Computing, 3,* 142-188.

Boucher, A. & Gerth, R. (1987). A timed model for extended communicating sequential processes. *Proceedings of ICALP'87,* Springer LNCS, 267.

Cardelli, L. & Wegner, P. (1985). On understanding types, data abstraction and polymorphism. *ACM Computing Surveys, 17*(4), 471-522.

Fecher, H. (2001). A real-time process algebra with open intervals and maximal progress. *Nordic Journal of Computing, 8*(3), 346-360.

Higman, B. (1977). *A comparative study of programming languages,* 2nd ed. MacDonald.

Hoare, C.A.R. (1978). Communicating sequential processes. *Communications of the ACM, 21*(8), 666-677.

Hoare, C.A.R. (1985). *Communicating sequential processes.* London: Prentice-Hall International.

Jones, C. B. (2003). Operational semantics: Concepts and their expression. *Information Processing Letters, 88*(1-2), 27 – 32.

Louden K.C. (1993). *Programming languages: Principles and practice*. Boston: PWS-Kent Publishing Co.

Martin-Lof, P. (1975). *An intuitionistic theory of types: Predicative part.* In H. Rose & J. C. Shepherdson (Eds.), Logic Colloquium 1973, NorthHolland.

McDermid, J. (Ed.) (1991). *Software engineer's reference book*. Oxford, UK: Butterworth Heinemann Ltd.

Milner, R. (1980). *A calculus of communicating systems,* LNCS #92. Springer-Verlag.

Milner, R. (1989). *Communication and concurrency.* Englewood Cliffs, NJ: Prentice-Hall

Mitchell, J.C. (1990). Type systems for programming languages. In J. van Leeuwen (Ed.), *Handbook of theoretical computer science* (pp. 365-458). North Holland.

Nicollin, X. & Sifakis, J. (1991). An overview and synthesis on timed process algebras. *Proceedings of the 3rd International Computer Aided Verification Conference,* pp. 376-398.

Plotkin, G. (1981). A structural approach to operational semantics. *Technical Report DAIMI FN-19,* Aarhus University, Denmark.

Schneider, S. (1995). An operational semantics for timed CSP. *Information and Computation, 116*(2), 193-213.

Slonneger, K., & Barry, L.K. (1995). *Formal syntax and semantics of programming languages: A laboratory based approach,* (Chapter 8), Reading, MA: Addison-Wesley Publishing Company.

Stubbs, D.F. & Webre, N.W. (1985). *Data structures with abstract data types and Pascal.* Monterey, CA: Brooks/Cole Publishing Co.

Tan, X. & Wang, Y. (2008). A denotational semantics of real-time process algebra (RTPA). *The International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), 2*(3).

Tan, X., Wang, Y., & Ngolah, C.F. (2006). Design and implementation of an automatic RTPA code

generator. *Proceedings of the 19th Canadian Conference on Electrical and Computer Engineering (CCECE'06),* Ottawa, ON, Canada, May, pp. 1605-1608.

Wang, Y. (2002). The real-time process algebra (RTPA). *Annals of Software Engineering: An International Journal*, *14*, 235-274.

Wang, Y. (2003). Using process algebra to describe human and software system behaviors. *Brain and Mind*, *4*(2), 199–213.

Wang, Y. (2006a). On the informatics laws and deductive semantics of software. *IEEE Transactions on Systems, Man, and Cybernetics (C), 36*(2), 161-171.

Wang, Y. (2006b). Cognitive informatics and contemporary mathematics for knowledge representation and manipulation, invited plenary talk. *Proceedings of the 1st International Conference on Rough Set and Knowledge Technology* (RSKT'06), LNAI #4062, Springer, Chongqing, China, July, pp. 69-78.

Wang, Y. (2007a). *Software engineering foundations: A software science perspective.* New York: Auerbach Publications.

Wang, Y. (2007b). The theoretical framework of cognitive informatics. *International Journal of Cognitive Informatics and Natural Intelligence* (IJCINI), *1*(1), 1-27.

Wang, Y. (2007c). On theoretical foundations of software engineering and denotational mathematics, keynote speech. *Proceedings of the 5th Asian Workshop on Foundations of Software,* BHU Press, Xiamen, China, pp. 99-102.

Wang, Y. (2008a). RTPA: A denotational mathematics for manipulating intelligent and computational behaviors. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), 2*(2), 44-62.

Wang, Y. (2008b). Deductive semantics of RTPA. *International Journal of Cognitive Informatics and Natural Intelligence* (IJCINI), *2*(2), 95-121.

Wang, Y. (2008c). On the Big-R notation for describing iterative and recursive behaviors. *International Journal of Cognitive Informatics and Natural Intelligence* (IJCINI), *2*(1), 17-28.

Wang, Y. & King, G. (2000). Software engineering processes: Principles and applications, CRC series in software engineering, Vol. I. CRC Press.

Wilson, L.B. & Clark, R.G. (1988). *Comparative programming language*. Wokingham, UK: Addison-Wesley Publishing Co.

Winskel, G. (1993). *The formal semantics of programming languages.* MIT Press.

Woodcock, J. & Davies, J. (1996). *Using Z: Specification, refinement, and proof.* London: Prentice Hall International.

This work was previously published in International Journal of Cognitive Informatics and Natural Intelligence, edited by Y. Wang, Volume 2, Issue 3, pp. 71-89, copyright 2008 by IGI Publishing (an imprint of IGI Global).

Chapter 8.11
Validation and Verification of Software Systems Using Virtual Reality and Coloured Petri Nets

Hyggo Oliveira de Almeida
Federal University of Campina Grande, Brazil

Leandro Silva
Federal University of Campina Grande, Brazil

Glauber Ferreira
Federal University of Campina Grande, Brazil

Emerson Loureiro
Federal University of Campina Grande, Brazil

Angelo Perkusich
Federal University of Campina Grande, Brazil

ABSTRACT

Validation and verification techniques have been identified as suitable mechanisms to determine if the software meets the needs of the user and to verify if the software works correctly. However, the existing verification techniques do not support friendly visualization. Also, validation techniques with friendly visualization mechanisms do not allow the verification of the system's correctness. In this chapter, we present a method for the validation and verification of software systems through the integration of formal methods and virtual reality. Furthermore, a software tool associated with such a method is also described along with an embedded system case study.

INTRODUCTION

The complexity of software systems is increasing and, consequently, making it more difficult and necessary to determine if they work correctly. In the context of the currently applied development techniques and processes, tests are commonly used for validating software, but they cannot ensure that the software is in accordance with important behavioral properties, such as robustness or safety requirements.

Verification techniques aim to detect and aid the designer to correct mistakes during the software development, being useful when defining whether the software satisfies its requirements and specifications. Through formal modeling and verification methods, it is possible to determine if the system works correctly while considering all possible behaviors.

On the other hand, validation techniques determine if the software meets the needs of the user (Fuhrman, Djlive, & Palza, 2003). Thus, a graphical and friendly visualization of the system model is very useful in validating software in different domains. However, even though most of the formal modeling techniques have their own graphical representations, such as automata and Petri nets, they do not allow a friendly visualization and, subsequently, validation of the system's behavior. Without such friendly visualization, the final user needs to understand formal modeling concepts, which are not easily understood by nonengineers.

Coloured Petri nets (CPN) (Jensen, 1992) are used to model and verify the correctness of software systems, and virtual reality modeling language (VRML) (International Organization for Standardization, 1998) is applied to validate the software behavior considering the user acceptance.

We will present a software platform for managing the integration of CPN and VRML, easing the application of the proposed method. In order to illustrate the use of the method and the platform, an embedded software case study is presented. Finally, related approaches and concluding remarks are discussed.

Using the proposed method, it is possible to separate the formal verification and the validation activities of the system, allowing the verification of its correctness while still giving a friendly visualization for the final user. Moreover, the proposed tool aids the developers in integrating the CPN formal and the friendly VRML models, hiding the integration complexity.

BACKGROUND

As we have said in the preceding section, the verification and validation phases of the approach we present here are based on coloured Petri nets and the VRML language. Therefore, in order to provide a better understanding of the next sections, it is desirable to present some background information.

Coloured Petri Nets

Petri nets are a formal method with a graphical representation used to model concurrent systems. With Petri nets, it is possible to specify and verify properties like precedence relation and deadlocks, among others. The graphical representation of a Petri net is a bipartite graph composed of places and transitions. Arcs connect places to transitions and transitions to places but never places to places or transitions to transitions. Each place can have zero or several tokens at a given moment. This is called the marking of the place. Therefore, the marking of a Petri net model is the set of markings of all places at a given moment. The transitions represent the actions that can take place.

Different extensions to the Petri nets formalism exist, such as timed Petri nets (Wang, 1998) and coloured Petri nets. In the context of this chap-

ter, the hierarchical coloured Petri nets (HCPN), an extension of the coloured Petri nets, is the formalism we have used in our approach. The HCPN formalism has been used in the verification of many kinds of systems. Analysis of mobile distributed protocols (Zaslavsky, Yeo, Lai, & Mitelman, 1995), verification of multiagent plans (Almeida, Silva, Perkusich, & Costa, 2005) and embedded systems (Silva & Perkusich, 2005), and simulation of network management systems (Christensen & Jepsen, 1991) are some of the practical uses of HCPN.

Hierarchical coloured Petri nets incorporate data types and hierarchy concepts to ease the modeling task. In HCPN, a system is represented by a set of nonhierarchical CPN models, and each of these models is called a CPN page. Hierarchical characteristics are achieved due to the inclusion of two mechanisms: substitution transition and fusion places. The former is a transition that is replaced by a CPN page. The page to which the substitution transition belongs is called a superpage. The page represented by such a transition is called a subpage. Subpages and superpages are associated by means of sockets and ports. Fusion places are places that are graphically distinct but logically the same. Two or more fusion places form a fusion set and all the places in a fusion set always have the same marking.

Indeed, these two additional mechanisms, substitution transition and fusion places, are only graphical, helping in the organization and visualization of a CPN model. They favor the modeling of larger and more complex systems by giving the designer the ability to model by abstraction, specialization, or both. Moreover, the modeling activities in CPNs are supported by a set of computational tools named Design/CPN[1] (Christensen & Mortensen, 1996). These tools provide a graphical environment for the editing and syntax checking, as well as analysis methods like simulation and verification of CPN models.

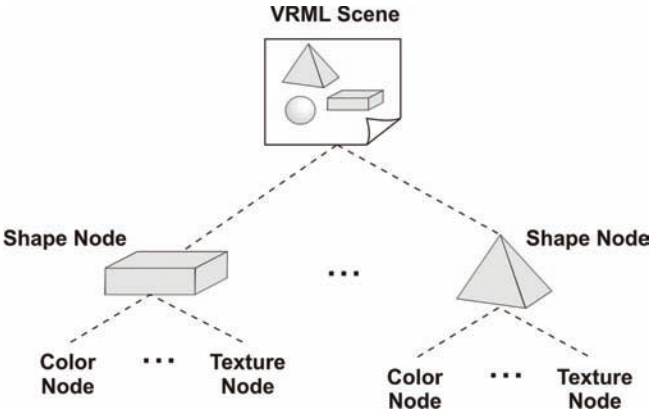
Virtual Reality Modeling Technologies

Different virtual modeling technologies exist. In this section, we outline some of these technologies by exposing their main characteristics.

VRML[2] is a platform-independent language for modeling three-dimensional (3D) worlds. The intent of its creators was to allow 3D scenes to be visualized through the World Wide Web even over low-bandwidth connections. It became the *de facto* approach for providing 3D content on the web. Furthermore, as 3D modeling tools were able to export files to the VRML language, non-VRML experts, like architects and mechanical engineers, were able to build their 3D models and export them to the Web. This characteristic has allowed the usage of VRML in several branches of academy and industry, such as chemical-molecules validation (Casher, Leach, Page, & Rzepa, 1998), virtual buildings modeling (Ferreira, et al., 2004), mathematical functions visualization (Wang & Saunders, 2005), robot simulation (Rohrmeier, 2000), terrains representation (Reddy, Iverson, & Leclerc, 1999), and manufacturing systems (Krishnamurthy, Shewchuk, & McLean, 1998; Silva & Perkusich, 2003).

In a general way, VRML worlds are tree-like structures, as illustrated in Figure 1. Almost everything in VRML is viewed as a node. Cylinders, spheres, lights and even colors, are some examples of VRML nodes. However, these nodes have no behavior associated with them. It means that, 3D scenes composed only of VRML nodes are static. Therefore, in order to embed some dynamics in such scenes, three further mechanisms have been provided: *sensors*, *routes*, and *scripts*. A *sensor*, as its name indicates, senses the changes of a specific feature, such as time and mouse clicking. The information gathered by a *sensor* can be redirected to the other nodes in the 3D world through *routes*. These *routes*, therefore,

Figure 1. The structure of a VRML world



act like event-dispatching mechanisms. Finally, *scripts*, also considered nodes, are the ways by which the developer manipulates the objects of a VRML scene.

Although VRML *scripts* are, most of the time, written in the JavaScript language (Goodman & Morrison, 2004), this is not the only option. It is also possible to implement such *scripts* by using the Java language. This can be performed through a mechanism called *external authoring interface* (EAI). EAI provides a set of Java classes that communicate with a VRML browser in order to manipulate a 3D world. The difference between these approaches is that in the second one the code of the script is separated from the VRML file. As we will show later in this chapter, this feature will be essential for the validation approach we are proposing.

The Java 3D[3] (Selman, 2002) API is part of Sun's JavaMedia suite and is devoted to the construction of Java applications and applets filled with 3D graphics. Like in VRML, Java 3D scenes are structured in a tree-like way. Therefore, each object in a scene is viewed as node of the tree. Through Java 3D, it is possible to control the shape, color, and transparency of objects. Furthermore, it allows the developer to define how such objects move, rotate, shrink, stretch, and morph as time goes by. Environmental manipulations are also possible, by defining background images, lighting modes, and even fog effects.

Internally, Java 3D uses the system native libraries, such as Direct3D and Open GL, to speed up performance. It provides applications with three different rendering modes: immediate, retained, and compiled-retained. The basic difference between these modes is concerned with performance, where the immediate mode is the slowest one and the compiled-retained is the fastest. Each of these modes has its own advantages, and therefore, the choice of which mode to use will depend on the application.

Extensible 3D graphics (X3D)[4] is both a language, based on extensible markup language (XML), and a format for creating and interchanging 3D content. As the X3D is compatible with the VRML specification, X3D files can be modeled using either X3D or VRML syntax.

Different from the previously presented technologies, X3D is embedded with the notion of *profiles*, each one providing a specific set of characteristics. The profiles are based on a componentized architecture, allowing the insertion of new components to support features not provided by the profile. This allows the definition of profiles targeted to specific fields, for example, medical and automobile visualizations.

The structure of a virtual world in X3D is similar to that of Java 3D and VRML, that is, a tree-like model. From the functional perspective, X3D provides many of the functionalities of VRML and Java 3D, such as scripts, lighting, materials, and animations. However, an interesting feature of X3D is the possibility of building virtual worlds by composing objects entirely from other scenes or locations on the Web.

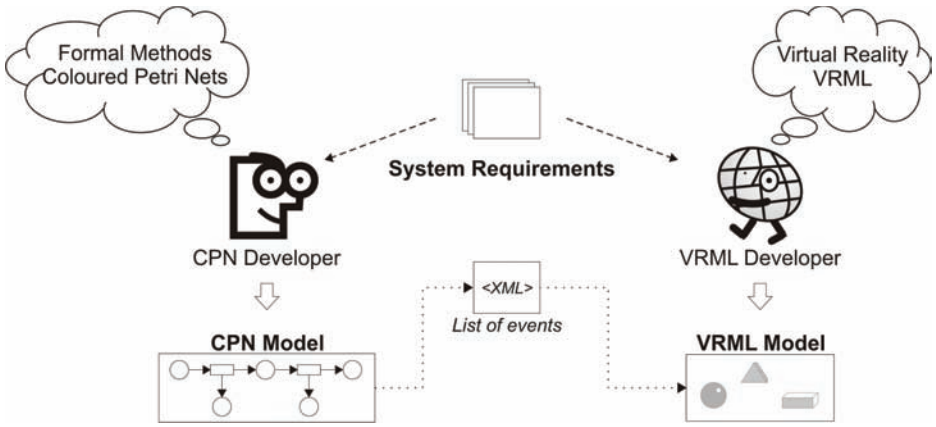
VALIDATION AND VERIFICATION METHOD

In the context of our method for software verification and validation, we have defined three roles: *CPN Developer*, *VRML Developer*, and *Final User*. Based on these roles, some steps for verifying and validating the software have been defined. Figure 2 illustrates such separation of roles in a macro view of our method.

The *CPN Developer* knows how to model and verify the system according to its requirements using coloured Petri nets. The *CPN Developer* aims at proving that the specification is correct. In case of modeling, he or she must identify the global system behavior and model it using arcs, places, and transitions. In general, each transition represents an occurrence of a system execution (e.g., an event, a process, etc.). Therefore, firing a transition or a set of transitions simulates a system behavior. For example, the firing of a transition named "input" may simulate the fact that the user inputs data on the system. When verifying the system, the specification modeling is proved correct for all possible behaviors.

The *VRML Developer* knows how to model and validate the system. However, unlike the *CPN Developer*, the *VRML Developer* aims at final user validation. For that, he or she must model the system as simply as possible, yet still

Figure 2. Separation of roles



in accordance with the specification. The VRML modeling must be performed based on the events defined as transitions by the *CPN Developer*. Thus, the *VRML Developer* is responsible for correlating CPN and VRML events.

The *Final User* knows how the system works. He or she is responsible for specifying the system and validating if the system modeled is the one specified previously. For that, the *Final User* must describe the system requirements and visualize the VRML model in determining if it covers the requirements.

In what follows, we describe a set of steps for validating and verifying systems using CPNs and VRML. Each step is related to a specific role. In Figure 3, the phases of the method are depicted. A simple manufacturing system is used for illustrating the application of each phase (Silva & Perkusich, 2003).

System Specification (Final User)

The specification of the system is the first and main phase. Functional and nonfunctional requirements are described and, based on them, the modeling can be performed. A manufacturing system is a system organized into cells. Each cell has one or more related machines and a transport system. Two

Figure 3. Verification and validation method

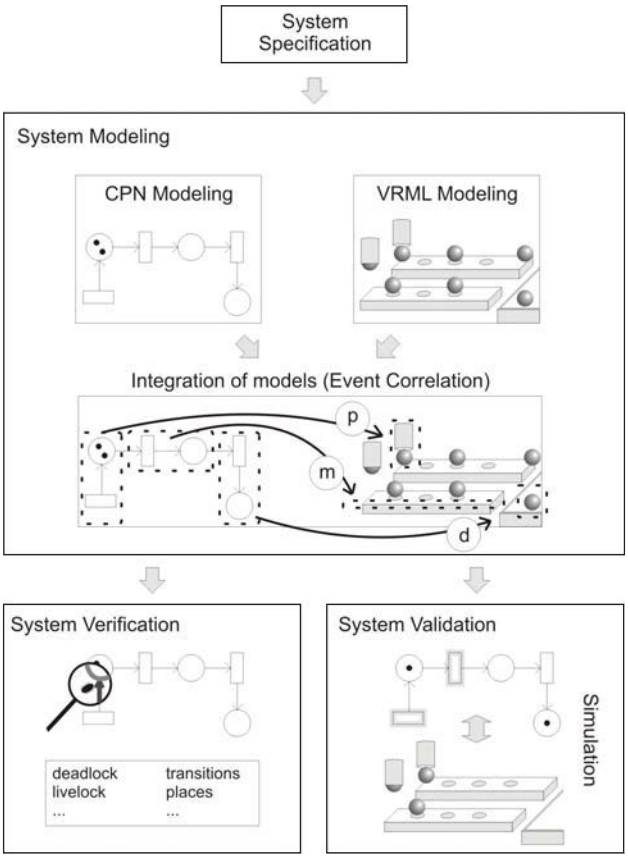
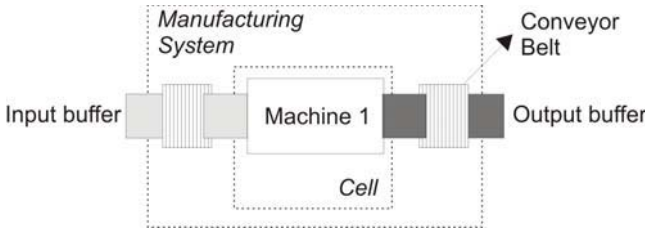


Figure 4. Simple manufacturing system



kinds of transport systems are considered: those inside cells and those outside cells. For simplicity, in this example the focus is on production and transport issues. In Figure 4, the architecture of the manufacturing system is illustrated.

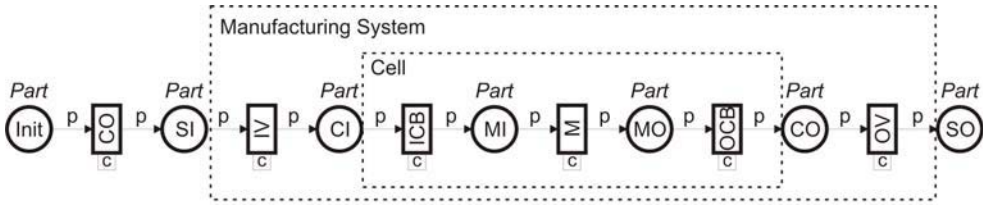
CPN Modeling (CPN Developer)

The system is modeled using hierarchical coloured Petri nets. The Design/CPN tools set should be used in this phase. The result of this activity is a set of CPN models that can be simulated and later verified. Figure 5 illustrates the CPN modeling for the manufacturing system, presenting the places and transitions that are used to model the system specification.

VRML Modeling (VRML Developer)

In the third phase, the system is modeled using VRML. Any VRML modeling tool can be used to perform this activity. This modeling is usually sim-

Figure 5. CPN modeling for the manufacturing system



CO	Connection
SI	System Input
IV	Input Vehicle
CI	Cell Input
ICB	Input Conveyor Belt
MI	Machine Input

M	Machine
MO	Machine Output
OCB	Output Conveyor Belt
CO	Cell Output
OV	Output Vehicle
SO	System Output

pler and friendlier than the CPN modeling, since the objective is to validate the system according to the user needs. Figure 6 illustrates the VRML modeling for the manufacturing system.

Integration of Models (VRML Developer)

The fourth step involves the correlation between the CPN and VRML events. The result of this activity is a map structure that represents the integration of the CPN and VRML models. For example, the event of firing a transition in the CPN model could represent a VRML event, such as a sphere moving. Figure 7 illustrates the correlation of events for the manufacturing systems. The correlated events are named CO, IV, OCB, and OV.

After the *system-modeling* phases, the system can be verified and validated. The generated

Figure 6. VRML modeling for the manufacturing system

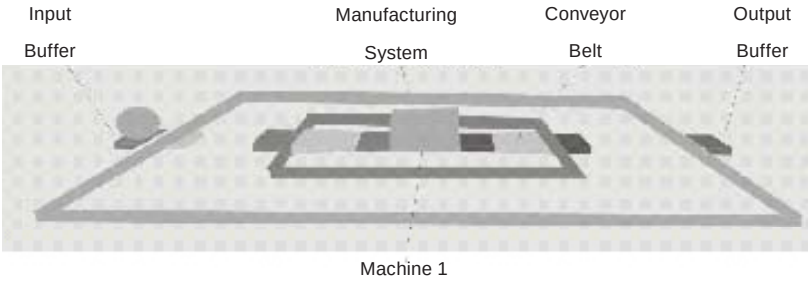
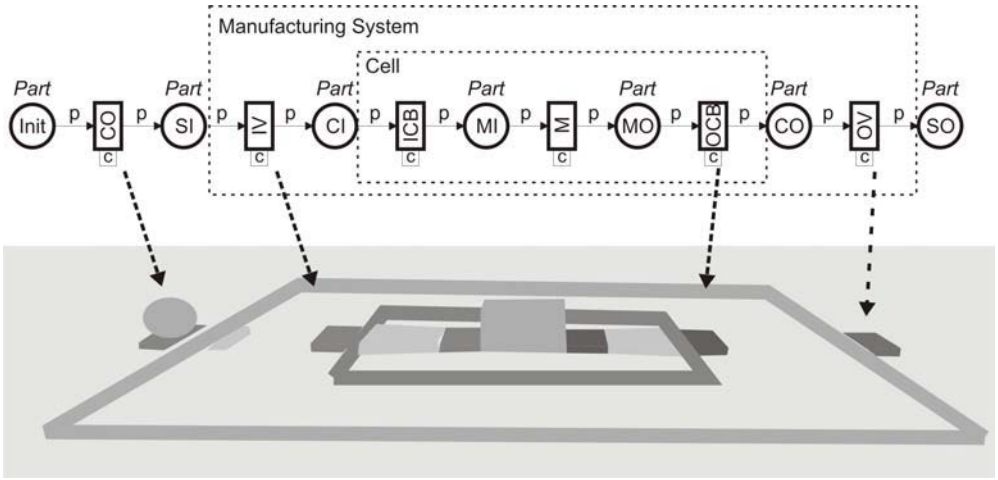


Figure 7. Correlating events for the manufacturing system models



models are used in the *system-verification* and the *system-validation* activities. Such activities are described below.

System Verification (CPN Developer)

In the fifth phase, the system is verified through the CPN models created during the *CPN-modeling* phase. Such models are passed to Design/CPN tools, in order to verify the correctness of the system according to behavioral properties using verification techniques such as model checking. Modeling and verification guidelines for CPNs using the Design/CPN tools and model checking can be found in Almeida et al. (2005).

System Validation (Final User)

In the last step, the system is validated through both the CPN and VRML models. A simulation of the system through the VRML models is performed based on the events generated by the simulation of the CPN models. Each event generated by the CPN model and defined as a "mapped event" in the *integration-of-models* activity can change the state of the VRML world. Thus, it is possible to visualize the behavior of the CPN-model simulation through the friendly VRML graphical animations.

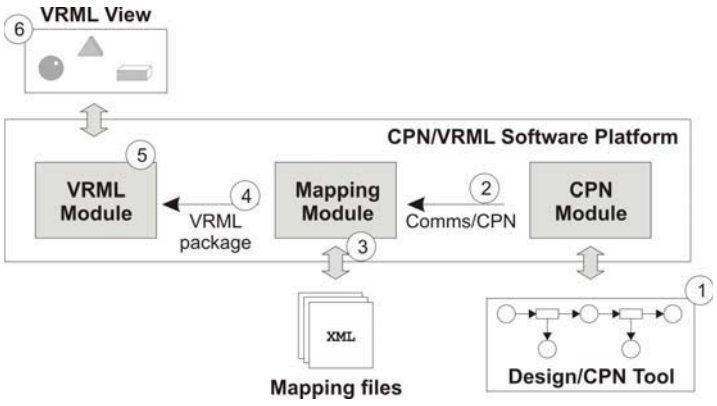
A SOFTWARE PLATFORM FOR INTEGRATING VRML AND CPN

In order to make the application of the steps for the validation and verification of software systems easier, a software platform is described here.

The *CPN/VRML Software Platform* manages the integration of VRML and CPN models. For that, the Design/CPN tool is used for CPN modeling and simulation, and a Java-compliant VRML browser is used for the model visualization. The architecture of the software platform is presented in Figure 8, and the integration steps are described below, considering the event correlation performed in the *integration-of-models* phase.

1. The *CPN Module* is responsible for receiving event notifications from the Design/CPN simulation tools. During the simulation, each event is announced for the *CPN Module*. The communication is performed via sockets.

Figure 8. Software-platform architecture



2. The *CPN Module* extracts the CPN model information using the Comms/CPN library (Gallasch & Kristensen, 2001) and forwards the event notifications to the mapping module.
3. The *Mapping Module* retrieves from the XML mapping files the VRML events that are related to the CPN event.
4. The *Mapping Module* forwards the identification of the VRML events that should be announced through a VRML-Java package.
5. Finally, the *VRML Module* updates the virtual world according to the CPN simulation events, allowing a friendly visualization of the CPN model. The *External Authoring Interface* mechanism, described in the "Background" section, is used to manipulate the VRML view.

CASE STUDY: A COMPONENT-BASED EMBEDDED AUTOMATION AND CONTROL SYSTEM

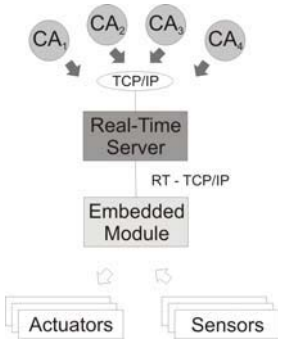
In this section the application of the proposed method and software platform for validating and verifying a component-based embedded automation and control system is described.

System Specification

In Figure 9, the architecture of the embedded system, which is based on the framework structure introduced in Perkusich, Almeida, and Araújo (2003) is illustrated.

The entities shown in this figure are: sensors and actuators, client applications (CAs), the embedded module, and the real-time server. Sensors and actuators are not described since their roles in the architecture are clear. Client applications, in turn, represent application front-ends that access the services of the real-time server, like a Web application. Since Web-based software architectures are well-defined, a more detailed discussion about client applications is omitted from this chapter.

Figure 9. Architecture of the real-time embedded automation and control application



Embedded Module

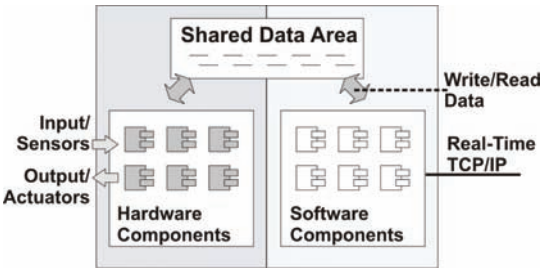
The embedded module is a real-time system that interacts with the physical environment by means of the sensors and actuators, as well as distributed devices connected by a real-time TCP/IP network. The hardware components of the module receive events or alarms from sensors and write information in a shared-data area. Then, software components can read such alarm and event information, convert them to an adequate format, and send them to the real-time server.

On the other hand, the real-time server can request for an actuator through the software components that write the request in the shared-data area. Thus, hardware components can read the requests from a shared area and send them to the specific actuators. The internal architecture for the embedded module is shown in Figure 10.

Since the focus of this chapter is on the validation and verification of software systems, the software components that belong to such infrastructure are depicted in Figure 11 and are detailed in what follows.

- **I/O interpreter:** The I/O interpreter mediates the communication between software components and information in a shared area. It also interprets the information recovered from the shared area and instantiates objects that represent such information. On the other hand, the I/O interpreter receives information from the components as objects, translates them to the shared-data language, and then writes them in the shared area. The type of information present in the shared area may vary according to the sensor and actuators that are being used.
- **Data converter:** The data converter implements data conversion specific algorithms defined by the application. The data are converted from the I/O interpreter language to the specific application language. For example, when the I/O interpreter returns an object with a binary content, the data converter may transform it into an object with a decimal content, if this is the application format. The data conversion also depends on the type of information.
- **Synchronizer:** The synchronizer implements the synchronous communication among the software components of the embedded module and the software components of the real-time server. As said before, a real-time TCP/IP network is used. This component acts as a bridge between the two sides of the distributed application, and all the requests, or the response for the requests, issued between the real-time server and the embedded module occur through this component.
- **Device controller:** The device controller implements the initialization, tests, and liberation of devices connected to the interfaces of the embedded module. Such services are executed based on the requests submitted by the *Device Controller* to the shared area through the interpreter.

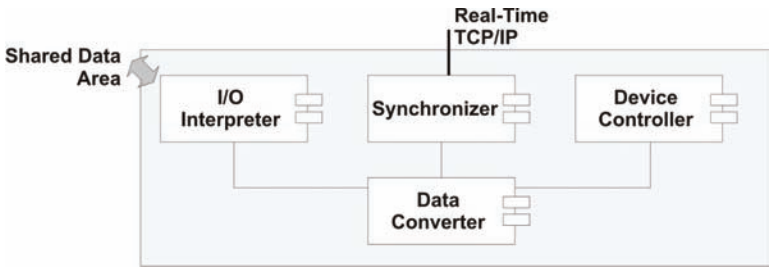
Figure 10. Internal architecture of the embedded module



Real-Time Server

The real-time server can be a personal or industrial computer that manages the network. It runs the

Figure 11. Software components infrastructure for the embedded module



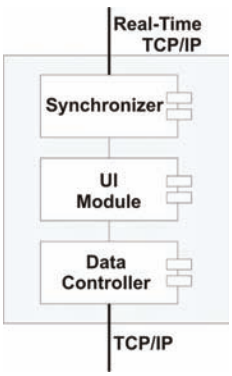
Linux/RT real-time operating system (Abbott, 2003). The internal architecture of the real-time server is shown in Figure 12. The software components that belong to the real-time-server infrastructure are briefly described as follows.

- **Synchronizer:** The synchronizer implements the communication on the real-time-server side. The implementation is similar to the embedded-module synchronizer.
- **Data controller:** The data controller provides data-flow control for the system (i.e., read, write, and parameter forwarding) based on the implementation of the services that are made available by the *UI Module*.
- **UI module:** The user-interface module provides an interface for the services implemented by the software components, so that they can be used by the application connected to the Internet.

CPN Modeling

In Figure 13, the CPN model for the embedded module is shown. The places, arcs, and transitions that model the *I/O-interpreter*, *Data-converter*, *Synchronizer*, and *Device-controller* modules are depicted in this figure. The complete description of the CPN modeling for this case study is presented in Silva and Perkusich (2005).

Figure 12. Software components infrastructure for the real-time server



VRML Modeling

Figure 14 illustrates the VRML modeling of the real-time embedded automation and control application. The software components detailed earlier and the communication among them have been modeled using VRML nodes. Box nodes, with their names modeled with *Text* nodes, represent the software components; whereas *Cylinder* nodes represent the communication among the components.

The "System Validation" section will detail how the dynamic features (i.e., data exchange among the components and data processing) of the system are simulated in VRML.

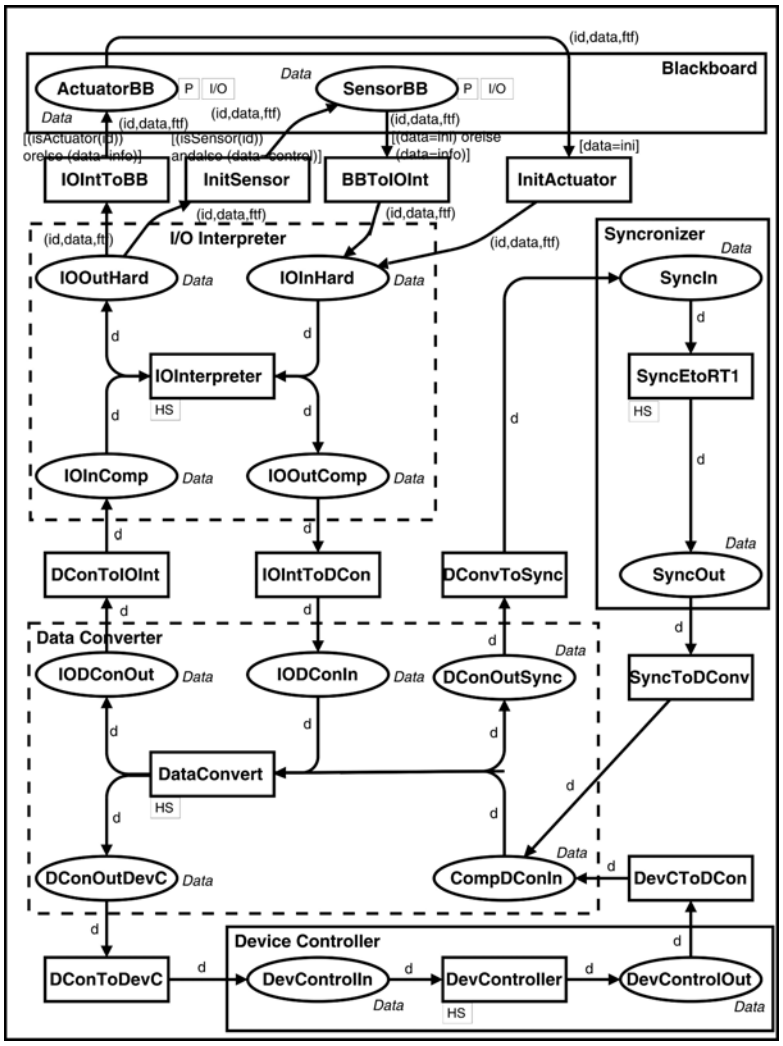
Integration of Models

Some VRML events have been created in order to allow the integration between the CPN and VRML models. Such events are implemented in JavaScript and they are grouped into a *Script* node through VRML input events. In Listing 1, the JavaScript function that implements the VRML-event behavior correlated to the CPN transition *UIModule* is presented. Such an event simulates the data exchange between the *Data Controller* and the *UI Module* of the *Real-time Server.*

The *rtsDataControllerToUIModule* function forwards the received event *value* to the input event *communicateForward* of the *Cylinder* situated between the *Data Controller* and the *UI Module*. Besides, this function also forwards the value *true* to the input event *showDataExchange* of the *UI Module*.

The other VRML events are similarly implemented. The complete mapping between the CPN transitions and the correlated VRML events is presented in Table 1.

Figure 13. CPN modeling for the embedded module

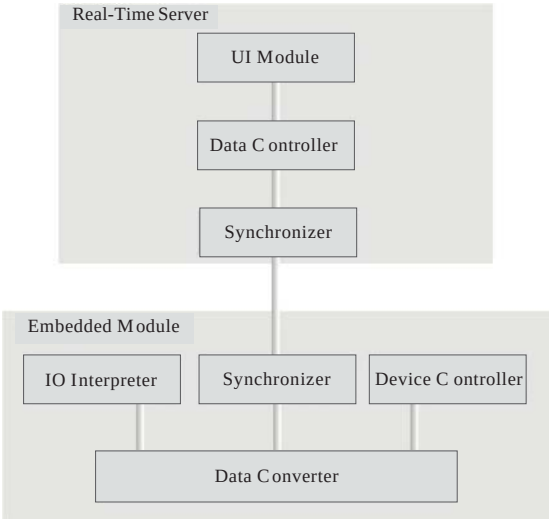


System Verification

Suppose that the sensors send an initial signal when the system is turned on or a new device is plugged in. Also, suppose that the system performs some task when it receives this kind of message and sends an acknowledgment, a calibration, or even an initialization message to the sensor. Such a scenario is illustrated in Figure 15.

When a sensor sends an initialization signal, the *Data Converter* sends it to the *Device Controller* to perform the associated control tasks. Since the simulation captures a single execution sequence or information flow in the model, the expected sequence or flow may have been violated. Thus, we must verify the scenario for all possible situations based on model checking (Clarke, Emerson, & Sistla, 1986) to guarantee that the expected flow has always been satisfied. For that, the desired system properties should be described in a propositional temporal logic such as computation tree logic (CTL) (Clarke et al.).

Figure 14. VRML modeling for the real-time embedded automation and control application



Listing 1. Function that implements the VRML event for the transition UIModule

```
function rtsDataControllerToUIModule(value, timestamp) {
        RTSDataControllerTOUIModuleCylinder.communicateForward = value;
        UIModuleComponent.showDataExchange = true;
}
```

Its semantic is defined with respect to paths in a Kripke structure. A path on such a structure is an infinite sequence of states (s_0; s_1; …) such that s_{i+1} is reached from s_i for all $i \geq 0$. The CTL formula *AG*φ means that for all paths starting from s_0, φ holds at every state along those paths. In other words, φ holds globally. The CTL formula *AF*φ means that for all paths starting from s_0, φ holds at some state along the path. In other words, φ is inevitable.

Considering the scenario shown in Figure 15, we must prove that when some device sends an initialization message, the flow will be through the device controller. To prove this scenario, two atomic propositions *PA* and *PB* are used. The proposition *PA* is true if there is a token in place *IODConIn*, illustrated in Figure 13. The proposition *PB* is true if there is a token in place *DConOutDevC*, also illustrated in Figure 13. The CTL formula to prove this scenario is shown below:

$$AG\ (\ PA \rightarrow AF\ (\ PB\ )\ )$$

Therefore, the formula is true if *PA* is true and *PB* is true in the future. It means that if there is a token in the data converter input, this token is sent to device controller input, which

Table 1. Mapping between the CPN transitions and the correlated VRML events

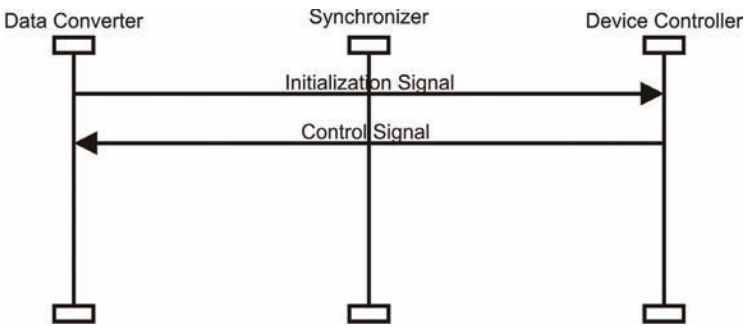
CPN transition	VRML event
ToUiMod	rtsDataControllerToUIModule
UIModule	rtsUIModuleProcessing
BackToDCon	rtsUIModuleToDataController
DataControl	rtsDataControlProcessing
TORTSyncB	rtsDataControllerToSynchronizer
SyncOutRT	rtsSynchronizerProcessing
DConToIOInt	emDataConverterToIOInterpreter
IOIntToDCon	emIOInterpreterToDataConverter
DataConvert	emDataConverterProcessing
DConToDevC	emDataConverterToDeviceController
DevCToDCon	emDeviceControllerToDataConverter
DevController	emDeviceControllerProcessing
SyncToDConv	emSynchronizerToDataConverter
DConvToSync	emDataConverterToSynchronizer
IOInterpreter	emIOInterpreterProcessing

is the component that implements control tasks such as initialization, calibration, and changing devices working parameters. The evaluation of this formula to true means that this part of the model behaves as expected for all possibilities of model execution. We can proceed with the same reasoning to prove that the flow of information back to the device also behaves as expected for all possibilities.

System Validation

The validation of the real-time embedded automation and control application is performed through the CPN and VRML models. The final user visualizes the interaction among the software components through the events generated by the CPN model and through a friendly user interface provided by the VRML model. The

Figure 15. Data converter flow to control signal



system validation becomes easier for the final user, since the VRML model entities are more abstract than the CPN model entities. In what follows, it is described how the dynamic features of the system are modeled in VRML.

The data exchange among the components is simulated through oriented *Cone* nodes (situated inside the *Cylinder* nodes), depending on the direction of the data flow. For example, in Figure 16, the data exchange between the *Data Controller* and the *UI Module* is presented: the *Cone* nodes are directed to the *UI Module* component, representing a data flow from the *Data Controller* to the *UI Module*. Besides, the *Sphere* node at the *UI Module* component represents the data that are sent from the *Data Controller* to the *UI Module*.

Another feature modeled for the system is data processing, which occurs within the software components. For that, a *Text* node with the word "Processing" is displayed at the bottom right of the current processing component and the letter "P" is displayed at the *Sphere* node, which represents the data being processed. In Figure 17, the data

Figure 16. VRML modeling for data exchange among components

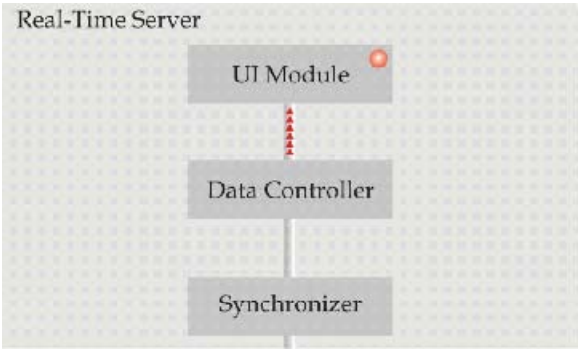
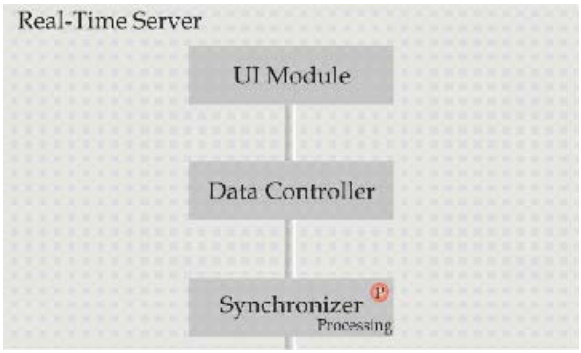


Figure 17. VRML modeling for data processing among components



processing within the *Synchronizer* component of the *Real-time Server* is illustrated.

RELATED APPROACHES

Some related approaches are discussed in this section. A short description showing their features is presented along with some comparisons with our work.

Kindler and Páles (2004) present a prototype tool that allows the 3D visualization of Petri-net models, called PVNis. Such a tool is based on the Petri net kernel (PNK) (Weber & Kindler, 2003) and uses Java 3D for implementing the 3D-visualization. The approach used by this tool is based on the equipment of the Petri net with information on the physical objects used for 3D-visualization. The physical objects are included in the Petri net model. A 3D model representing the shape of the object and an *animation function* representing the behavior of the object must be created for each place of the Petri net that corresponds to a physical object.

There is no method for validation and verification associated with this tool. Its purpose is only to provide the visualization of Petri net models in a 3D way. Therefore, there is also no effort to integrate the validation process through the 3D-visualization, with the verification process, or through the Petri net model. Such activities are realized separately.

Bardohl, Ermel, and Ribeiro (2000) introduce ideas towards visual specification and animation of Petri net-based models. In this work, the animation of algebraic high-level (AHL) nets (Padberg, Ehrig, & Ribeiro, 1995), which are a combination of place-transition nets and algebraic specifications (Ehrig & Mahr, 1985), is described. For that the animation, transformation rules are applied to the behavior rules (semantically equivalent to the transitions of the net) of an AHL net model, generating an animation view from the model. Such a view allows the visualization of the behavior of the system in a domain-specific layout.

In a Petri net Editor,the user defines the transformation rules from the Petri-net view to the animation view. In this way, there is no separation of the roles concerned with the use of this approach. Either the user that knows Petri nets must know the visual-modeling language or the visual modeler must know Petri nets. Finally, it seems the visual-modeling language is not as full-featured as VRML.

CONCLUSION AND FUTURE TRENDS

This chapter describes a method for the verification and validation of software systems using coloured Petri nets (CPN) and VRML. The method is defined over three separated roles: the *CPN Developer*, responsible for CPN issues; the *VRML Developer*, responsible for VRML issues; and the *Final User*, responsible for specifying and validating the system.

According to the proposed method, the system is modeled using CPN and VRML based on its requirements. Next, the CPN and VRML events are correlated in order to allow the CPN model simulation to control the VRML model simulation. For that, we propose a software tool that is used to correlate CPN and VRML events.

We describe the method as a set of phases for modeling and verification. The application of each phase is illustrated using a flexible manufacturing-system example. This makes it easier to understand the implementation of our method. As a real case study, we described the validation and verification of an embedded software system using the proposed method and platform.

The main future trend in the context of software verification and validation related to our work is to provide mechanisms for reducing the gap between validation and verification techniques. In this chapter, we propose a method and a tool that reduce such a gap, promoting the integration between VRML and CPN developers, still obtain-

ing software verification and friendly software validation.

However, for some domains, the integration can be improved. For example, a VRML-component library for a specific domain may be created. Thus, when verifying and validating software for such domain, the integration tool could generate the VRML model based on the name of the component events. This automatic approach would reduce the effort on mapping CPN events to VRML ones.

On the other hand, verification tools could be improved in order to provide friendly mechanisms for visualizing what is being verified. It will be very important to the large-scale use of verification techniques in the context of industrial software engineering.

In this future, we plan to apply our method for validating and verifying software in other domains, focusing on complex interaction-centric ones (e.g., multiagent systems, as in Weiss, 2000, and workflow management systems, as in van der Aalst & van Hee, 2002). The validation of these systems through VRML may be primordial.

REFERENCES

Abbott, D. (2003). *Linux for embedded and real-time applications.* Oxford, UK: Newnes.

Almeida, H., Silva, L., Perkusich, A., & Costa, E. (2005). A formal approach for the modelling and verification of multiagent plans based on model checking and Petri nets. In R. Choren, A. Garcia, C. Lucena, & A. Romanovsky (Eds.), *Software engineering for multi-agent systems III: Research issues and practical applications* (Vol. 3390, pp. 162-179). Berlin, Germany: Springer-Verlag.

Bardohl, R., Ermel, C., & Ribeiro, L. (2000). Towards visual specification and animation of Petri net based models. In *Proceedings of the Workshop on Graph Transformation Systems (GRATRA '00)* (pp. 22-31). Berlin, Germany.

Casher, O., Leach, C., Page, C. S., & Rzepa, H. S. (1998). Virtual reality modelling language (VRML) in chemistry. *Chemistry in Britain*, *34*(9), 26.

Christensen, S., & Jepsen, L. O. (1991, June 17-19). Modelling and simulation of a network management system using hierarchical coloured Petri nets. In E. Moseklide (Ed.), *Modelling and Simulation, 1991: Proceedings of the 1991 European Simulation Multiconference, the Panum Institute Copenhagen, Denmark.* San Diego, CA: Society for Computer Simulation International.

Christensen, S., & Mortensen, K. H. (1996). *Design/CPN ASK-CTL manual.* Denmark: University of Aarhus.

Clarke, E. M., Emerson, E. A., & Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, *2*(8), 244-263.

Ehrig, H., & Mahr, B. (1985). *Fundamentals of algebraic specification 1: Equations and initial semantics.* Berlin, Germany: Springer-Verlag.

Ferreira, G. V., Loureiro, E. C., Nogueira, W. A., Gomes, A. A., Almeida, H. O., & Frery, A. (2004). Uma abordagem baseada em componentes para a construção de edifícios virtuais (in Portuguese). In *Proceedings of VII Symposium on Virtual Reality-SVR 2004* (Vol. 7, pp. 279-290). Porto Alegre, Brazil: Sociedade Brasileira de Computação.

Fuhrman, C., Djlive, F., & Palza, E. (2003). Software verification and validation within the (rational) unified process. In *Proceedings of the Software Engineering Workshop, 2003, 28th Annual NASA Goddard* (pp. 216 - 220). Washington, DC: IEEE Computer Society.

Gallasch, G., & Kristensen, L. M. (2001). Comms/CPN: A communication infrastructure for external communication with Design/CPN. In K. Jensen (Ed.), *3rd Workshop and Tutorial on Practical*

Use of Coloured Petri Nets and the CPN Tools (CPN'01) (pp. 75-90). Denmark: DAIMI PB-554, Aarhus University.

Goodman, D., & Morrison, M. (2004). *JavaScript bible*. Hoboken, NJ: John Wiley & Sons.

International Organization for Standardization. (1998). *ISO/IEC 14772-1:1998: Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding.* Geneva, Switzerland: International Organization for Standardization.

Jensen, K. (1992). *Coloured Petri nets. Basic concepts, analysis methods and practical use* (Vol. 1). Berlin, Germany: Springer-Verlag.

Kindler, E., & Páles, C. (2004, June 21-25). 3D-visualization of Petri net models: Concept and realization. In J. Cortadella & W. Reisig (Eds.), *Applications and Theory of Petri Nets 2004: Proceedings of the 25th International Conference, ICATPN 2004,* Bologna, Italy (Vol. 3099, pp. 464-473). Berlin, Germany: Springer-Verlag.

Krishnamurthy, K., Shewchuk, J., & McLean, C. (1998, May 18-20). Hybrid manufacturing system modeling environment using VRML. In J. J. Mills & F. Kimura (Eds.), *Information infrastructure systems for manufacturing II, IFIP TC5 WG5.3/5.7 Third International Working Conference on the Design of Information Infrastructure Systems for Manufacturing (DIISM '98),* Fort Worth, TX (Vol. 144, pp. 163-174). London: Kluwer.

Padberg, J., Ehrig, H., & Ribeiro, L. (1995). Algebraic high-level net transformation systems. *Mathematical Structures in Computer Science, 5*(2), 217-256.

Perkusich, A., Almeida, H., & Araújo, D. (2003). A software framework for real-time embedded automation and control systems. In *Proceedings of Emerging Technologies and Factory Automation, 2003 (ETFA '03) IEEE Conference* (Vol. 2, pp. 181-184). Washington, DC: IEEE Computer Society.

Reddy, M., Iverson, L., & Leclerc, Y. G. (1999). Enabling geographic support in virtual reality modeling with GeoVRML. *Cartography and Geographic Information Science, 26*(3), 180-182.

Rohrmeier, M. (2000). Web based robot simulation using VRML. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *Proceedings of the 32nd Winter Simulation Conference* (pp. 1525-1528). San Diego, CA: Society for Computer Simulation International.

Selman, D. (2002). *Java 3D programming.* Greenwich, UK: Manning Publications.

Silva, L., & Perkusich, A. (2003). Uso de realidade virtual para validação de modelos de sistemas flexíveis de manufatura (in Portuguese). *Anais do VI Simpósio Brasileiro de Automação Inteligente.* São Paulo, Brazil.

Silva, L., & Perkusich, A. (2005). A model-based approach to formal specification and verification of embedded systems using coloured Petri nets. In C. Atkinson, C. Bunse, H. G. Gross, & C. Peper (Eds.), *Component-based software development for embedded systems* (vol. 3778, pp. 35-58). Berlin, Germany: Springer-Verlag.

van der Aalst, W., & van Hee, K. (2002). *Workflow management: Models, methods, and systems: Cooperative information systems.* Cambridge, MA: MIT Press.

Wang, J. (1998). *Timed Petri nets: Theory and applications.* London: Kluwer Academic.

Wang, Q., & Saunders, B. (2005). Web-based 3D visualization in a digital library of mathematical functions. In *Proceedings of the Tenth International Conference on 3D Web Technology* (pp. 151-157). New York: ACM Press.

Weber, M., & Kindler, E. (2003). The Petri net kernel. In H. Ehrig, W. Reisig, G. Rozenberg, &

H. Weber (Eds.), *Petri net technology for communication-based systems: Advances in Petri nets* (Vol. 2472, pp. 109-124). Berlin, Germany: Springer-Verlag.

Weiss, G. (Ed.). (2000). *Multiagent systems — A modern approach to distributed artificial intelligence*. Cambridge, MA: MIT Press.

Zaslavsky, A., Yeo, L., Lai, S., & Mitelman, B. (1995). Petri nets analysis of transaction and submitter management protocols in mobile distributed computing environment. In *Proceedings of the 4th International Conference on Computer Communications and Networks (ICCCN '95)* (pp. 292-299). Washington, DC: IEEE Computer Society.

ENDNOTES

1. Design/CPN can be downloaded at http://www.daimi.au.dk/designCPN/
2. VRML resources such as editors and browsers are available at http://www.vrmlsite.com/
3. The official site of Java 3D is available at https://java3d.dev.java.net
4. The official site of X3D is available at http://www.web3d.org

This work was previously published in Verification, Validation and Testing in Software Engineering, edited by A. Dasso, pp. 28-54, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 8.12
Software Component Survivability in Information Warfare

Joon S. Park
Syracuse University, USA

Joseph Giordano
Air Force Research Laboratory, USA

ABSTRACT

The need for software component survivability is pressing for mission-critical systems in information warfare. In this chapter, we describe how mission-critical distributed systems can survive component failures or compromises with malicious codes in information warfare. We define our definition of survivability, discuss the survivability challenges in a large mission-critical system in information warfare, and identify static, dynamic, and hybrid survivability models. Furthermore, we discuss the trade offs of each model. Technical details and implementation of the models are not described in this chapter because of space limitations.

INTRODUCTION

As information systems became ever more complex and the interdependence of these systems increased, the survivability picture became more and more complicated. The need for survivability is most pressing for mission-critical systems in information warfare. When components are exported from a remote system to a local system under different administration and deployed in different environments, we cannot guarantee the proper execution of those remote components in the current run-time environment. Therefore, in the run time, we should consider component failures (in particular, remote components) that may occur due to poor implementation, during integration with other components in the system,

or because of cyber attacks. Although advanced technologies and system architectures improve the capability of today's systems, we cannot completely avoid threats to them. This becomes more serious when the systems are integrated with commercial off-the-shelf (COTS) products and services, which typically have both known and unknown vulnerabilities that may cause unexpected problems and that can be exploited by attackers trying to disrupt mission-critical services (Kapfhammer, Michael, Haddox, & Colyer, 2000). Organizations, including the Department of Defense (DoD), use COTS systems and services to provide office productivity, Internet services, and database services, and they tailor these systems and services to satisfy their specific requirements. Using COTS systems and services as much as possible is a cost-effective strategy, but such systems—even when tailored to the specific needs of the implementing organization—also inherit flaws and weaknesses from specific COTS products and services that are used. Therefore, we need reliable approaches to ensure survivability in mission-critical systems that must rely on commercial services and products in a distributed computing environment.

Definitions of survivability were introduced by previous researchers (Knight & Sullivan, 2000; Lipson & Fisher, 1999). We define survivability as the capability of an entity to continue its mission even in the presence of damage to the entity (Park, Chandramohan, Devarajan, & Giordano, 2005). An entity ranges from a single software component (object), with its mission in a distributed computing environment, to an information system that consists of many components to support the overall mission. An entity may support multiple missions.

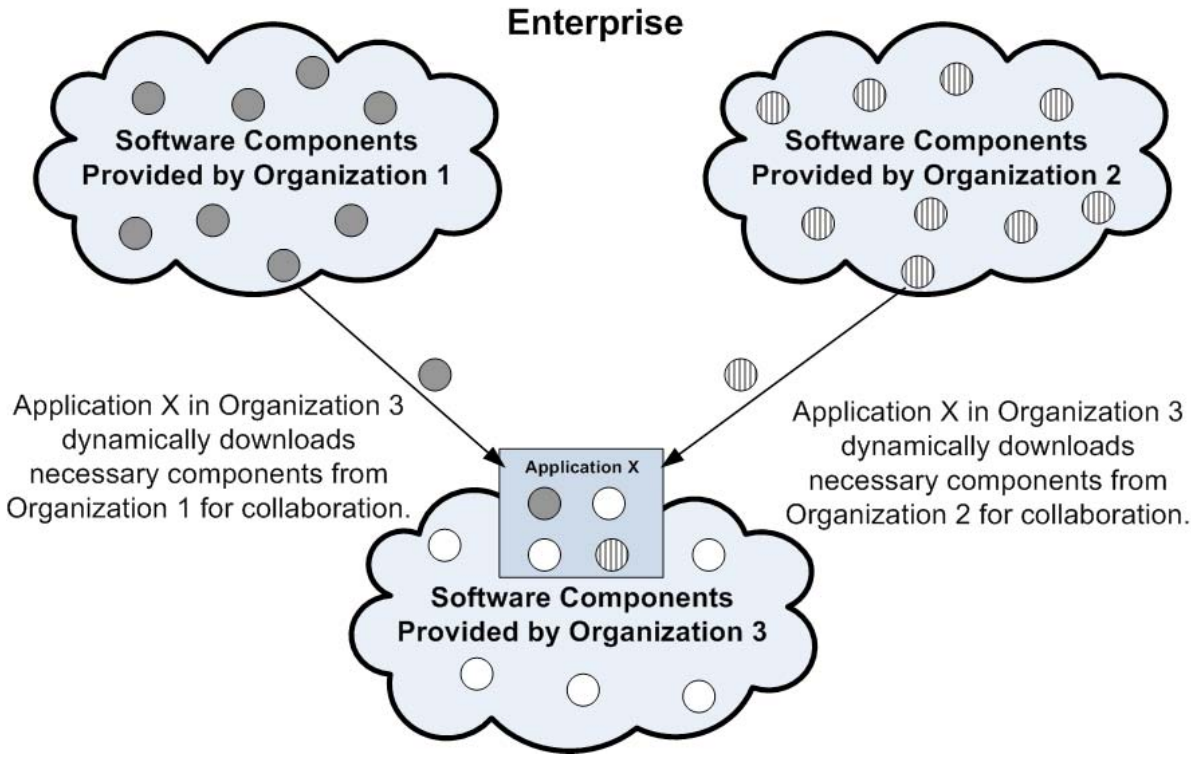
The damage caused by cyber attacks, system failures, or accidents, and whether a system can recover from this damage (Jajodia, McCollum, & Ammann, 1999; Knight, Elder, & Du, 1998; Liu, Ammann, & Jajodia, 2000), will determine the survivability characteristics of a system. A survivability strategy can be set up in three steps: protection, detection and response, and recovery (Park & Froscher, 2002). To make a system survivable, it is the mission of the system rather than the components of the system. This implies that the designer or assessor should define a set of critical services the system must provide in order to fulfill the mission. In other words, they must understand what services should be survivable by the mission and what functions of which components in the system should continue to support the system's mission.

In this article, we focus on the survivability of mission-critical software components downloaded on the Internet. We assume that all software components are susceptible to malicious cyber attacks or internal failures. Cyber attacks may involve tampering with existing source code to include undesired functionality (e.g., Trojan horses), or replacing a genuine component with a malicious one. When using such components, particularly in mission-critical applications in information warfare, we must check to see if the component was developed by a trusted source, and whether the code has been modified in an unauthorized manner since it was created. Furthermore, we should check to see if the component is functioning in an expected way. If all these conditions are satisfied, we call it "trusted component sharing."

CHALLENGES TO SOFTWARE SURVIVABILITY IN A MISSION-CRITICAL SYSTEM

Typically, an application running at an enterprise level may span more than one organization. Figure 1 shows an example of a distributed application that spans multiple organizations. The figure depicts three organizations interconnected to form a large enterprise-computing environment. In the real world, there may be more than two or three organizations connected to form a large

Figure 1. A distributed application spanning multiple organizations



enterprise, and some of the organizations in the enterprise may provide specialized services that other organizations do not provide (e.g., Department of Homeland Security). In the figure, for example, components in Organizations 1 and 3 are involved in application X. In this example the application running in Organization 3 downloads necessary components for some special features that it lacks. These components are dynamically downloaded from remotely administered hosts (in Organization 1 in the example) and run locally. This situation becomes complex when one must administer components downloaded from disparate administrations. For instance, Figure 1 shows that a user under Organization 3's administration can dynamically download software components under Organization 1's administration. From this point forward, the software running in Organization 3 should cooperate with the downloaded components that originated from different administrations. To employ autonomous administration, the local administrator must perform the extra job of dealing with interoperability problems and failures of or attacks via external components.

Based on the typical scenario described above, we identify the following generic challenges for software survivability in large distributed mission-critical systems.

Challenge 1. An autonomous mechanism to support component survivability between dif-

ferent organizations or systems is needed due to the fact that no single administrator can control every aspect (e.g., software component testing and implementation) of the various systems used in an enterprise. This is an inherent challenge because many systems, including those from different organizations, are integrated within current distributed computing environments. This implies that a remote component may have failures or malicious code that could affect a local computing environment. Unfortunately, a remote component cannot be tested in a local environment until runtime.

Challenge 2. Testing software components before deployment cannot detect or anticipate all of the possible failures or attacks that may manifest themselves during run time, especially when external components are integrated. Some failures are detected only when the components are deployed and integrated with other components in the operational environments. Existing faults in one component can be triggered by other components during runtime. Furthermore, since we cannot simply assume that all the participating organizations followed proper testing procedures for their software components, we need a new component-test mechanism that can test the component in the actual run-time environment—especially for components downloaded from different environments. The test criteria can vary, based on current applications or run-time environments, even for the same component. The remote component may cause an interoperability problem in a different run-time environment, although it passed its original test.

Challenge 3. In a distributed mission-critical system, we must check whether a remote component has been altered in an unauthorized manner, especially if it contains malicious codes, such as Trojan Horses, viruses, or spyware, before malicious codes are activated in the run-time environment. For instance, in Figure 1, when different organizations collaborate for a common enterprise but are competitors in the market, each organization should check the components from other organizations before they are used in the local environment. Furthermore, if a component includes any malicious codes, but the functionality of the original code is still needed for the system, we cannot simply reject the entire component. Instead, we should safely retrieve only the original code, enervating the malicious code.

Challenge 4, Currently available redundancy-based static approaches cannot solve the problem completely. If one component has failed because of reason R1, then the rest of the redundant components will fail for the same reason. It is only a matter of time before every redundant component is compromised for the same reason, especially when those components are identical. Furthermore, the strength of the redundancy-based approaches depends on the prepared redundancy, which brings up the question of "how many" redundant components we need to provide. Technically, one could maintain as many redundant components as necessary for a critical service. However, if the initially selected component is running in its normal state—meaning there is no need to use other redundant components as the component is not defective or compromised—the cost for running the redundant components has been wasted. In this situation, the resource efficiency is low, and the maintenance cost is high. Therefore, for mission-critical systems in information warfare, a dynamic technique is needed to detect and analyze the possible faults and attacks in the components and fix/immunize these components on-the-fly.

Challenge 5. Even if we know the reasons for and the locations of the software failures or attacks, in most currently available recovery approaches in distributed computing environments, changing the component's capability (e.g., for immunization) in run time is difficult, especially when the source code is not available (which is not an uncommon situation). When dealing with component failures, we are concerned with the problem of how to fix these failed components. One possibility is to modify the source code ac-

cording to the identified failures; however, this approach is possible only if the source code for that component is available. In the case of COTS components and other components downloaded from externally administered systems, the source code is often unavailable. Although some source codes are available to the public, if they are poorly documented, it will hardly be possible to modify the source code effectively. Furthermore, many mission-critical systems do not tolerate the suspension of operation for code debugging and recompilation. One must, therefore, employ other techniques to achieve the goal of fixing failed components on-the-fly—without access to the source code—in order for the mission of the component to continue.

SUPPORT MECHANISMS FOR SOFTWARE SURVIVABILITY

Component Recovery

Barga, Lomet, Shegalov, and Weikum (2004) introduced a framework for an application-independent infrastructure that provides recovery of data, messages, and states in Web-based applications. The framework requires an interaction contract between two components that specifies the joint behavior of those two interacting components. An application component can be replayed from an earlier installed state and arrive at the same state as in its prefailure incarnation. It masks from users the failures of clients, application servers, or data servers only if the replay could recover the component. Unfortunately, however, there are many component failures that cannot be fixed simply by replays. With this approach, a component may go back to the state before the failure; however, it cannot proceed further from that failure point again, unless the reason for the failure is fixed. Therefore, the component is not able to continue its mission. Furthermore, this approach does not consider malicious codes that are already included in the components. Similarly, the state-based recovery approaches for client-server systems were introduced by Freytag Cristian, and KŁhler (1987) and Barga, Lomet, Baby, and Agrawal (2000), and for databases by Liu et al. 2000) and Jajodia et al. (1999).

Ring, Esler, and Cole (2004) introduced self-healing mechanisms for kernel system compromises in run time, which analyze the system call table and enable the compromised addresses to return to their original values, terminate hidden processes, remove hidden files, and block hidden connections. As with state-based recovery, these can recover a compromised system to its state before the failure or attack and prevent the situation from getting worse. However, like the state-based approach, this approach does not fix the fundamental reason for the problem or let the system go to further states. Also, the scope of the work is within a kernel module in Linux.

Dowling and Cahill (2004) introduced the idea of K-components for self-adaptive decentralized systems. By using a component interface definition language called K-IDL, the definitions of component states and adaptation actions are used by decision-making programs to reason about and adapt component operation. Programmers can specify adaptation contracts for their local environments. However, this limits the overall robustness of the adaptive systems because, typically, programmers do not know all the possible adaptive options in various computing environments in which their components will be running. When the components are used in a large distributed application, which is the scope of our work, this approach is especially not scalable.

Helsinger, Kleinmann, and Brinn (2004) introduced a multitiered control framework between high-level observable metrics and low-level control actions for distributed multiagent systems that are composed of distributed autonomous agents interacting on a peer-to-peer basis. The framework imposes intermediate measurers of performance (MOPs) that measure the contri-

butions of different components and actions to higher-level functions. They observe the behaviors at one level and seek to manage those behaviors by taking control actions at a lower level, such as by restarting dead agents or load balancing by moving agents between hosts. Assuming that each agent will be developed correctly and trusted, this framework can improve the availability of agents and may optimize load balancing. However, the framework may not comply properly when the components have internal failures or are compromised by malicious codes, which is not unusual in a real mission-critical distributed system in information warfare.

Component Test

Existing technologies for identifying faulty components are more or less static in nature. One of these approaches employs black-box testing of the components. In this technique, behavioral specifications are provided for the component to be tested in the target system. This technique treats the target component as a black box and can be used to determine how the component behaves anomalously. Traditionally, black-box testing is done without knowledge of the internal workings of the component tested. Normally, black-box testing involves only input and output details of the component, while information on how the output is arrived at is not needed. The main disadvantage of this technique is that the specifications should cover all the details of the visible behavior of the components, which is impractical in many situations. Another approach employs a source-code analysis, which depends on the availability of the source code of the components. Software testability analysis employs a white-box testing technique that determines the locations in the component where a failure is likely to occur. Unlike black-box testing, white-box testing allows the tester to see the inner details of the component, which later helps him to create appropriate test data. Yet another approach is software component dependability assessment, a modification or testability analysis that thoroughly tests each component. These techniques are possible only when the source code of the components is available.

In the past, Kapfhammer et al. (2000) have employed a simple behavioral specification utilizing execution-based evaluation. This approach combines software fault injection (Avresky, Arlat, Laprie, & Crouzet, 1996; Hsueh, Tsai, and Iyer, 1997; Madeira, Costa, and Vieira, 2000) at component interfaces and machine learning techniques to: (1) identify problematic COTS components and (2) to understand these components anomalous behavior. They isolated problematic COTS components, created wrappers, and introduced them into the system under different analysis stages to uniquely identify the failed components and to gather information on the circumstances that surrounded the anomalous component behavior. Finally, they preprocessed the collected data and applied selective machine learning algorithms to generate a finite state machine to better understand and to increase the robustness of faulty components. In other research (Chen, Kiciman, Fratkin, Fox, & Brewer, 2002; Voas & McGraw, 1998), the authors developed a dynamic problem determination framework for a large J2EE platform, employing a fault-detection approach based on data-clustering mechanisms to identify faulty components.

COMPONENT SURVIVABILITY MODELS

In this section, we identify static, dynamic, and hybrid models for software survivability and discuss their trade offs. Technical details and implementation of these models are available in our previous publications (Park & Chandramohan, 2004; Park et al., 2005). We describe our approaches using client and server components in a typical distributed environment.

Static Survivability Model

The survivability of this model is based on redundant components, prepared before the operation, to support critical services continuously in a distributed client-server environment. Redundant servers can be located in the same machine, in different machines in the same domain or even in different domains. Existing approaches, such as dynamic reconfiguration, can be associated with this static model. Although the term "dynamic" is used in the terminology, it belongs to the static model, according to our definition, as long as the available components are generated before the operation starts. The same service can be provided by identical components (e.g., copies of the original servers) or by different components that are implemented in various ways. Isolated redundancy (in different machines or domains) usually provides higher survivability because the replaced component can be running in an uninfected area. For instance, if the redundant components are distributed in different network places, then the services provided by those components can be recovered in the event of network failures, in different environments. However, if there is a failure within a component, replacing that component with an identical copy is not effective, because identical components are vulnerable to the same failure.

Dynamic Survivability Model

In this model, unlike the static model, there is no redundant component. Components with failures or malicious codes are replaced by dynamically generated components on-the-fly and deployed in run time when required. Furthermore, this model allows replacement of the infected components with immunized components, if possible, which enables it to provide more robust services than the static model.

Basically, when failures or malicious codes are detected in a component, the corresponding factory generates an immunized component and deploys it into a safe environment, while the monitor issues a command to shut down the old component. The concept of "factory" was originally introduced as one of the commonly used design patterns in the object-oriented design. If we do not know the exact reason for the failures or types of malicious codes, or if it is hard to immunize components against known failures or malicious codes, we can simply replace the infected component with a new one in a safe environment. We call this a generic immunization strategy, which is effective against cyber attacks. If a component (a machine or a whole domain) is under attack, the generic immunization strategy suggests generating a new copy of the component and deploying it in a new environment that is safe from attack. Although this approach supports service availability continuously, the new component might still be susceptible to the same failures or attacks. Therefore, an immunized component, if possible, would provide a more robust survivability.

The Hybrid Model

The hybrid survivability model combines the features of the above two models. The dynamic model has an inherent disadvantage in terms of service downtime. The recovery process can range from seconds to a few minutes. This downtime drawback will cause major problems in mission-critical systems in information warfare because there will be no service available for clients during the recovery period. On the other hand, the static model has inherent disadvantages in terms of resource efficiency, adaptation, and robustness.

To compensate for the weaknesses in the two models, we incorporated the idea of a hybrid model. At the beginning of the operation, an array of n redundant components is initiated as described in the static model approach. These redundant servers will be used as a buffer while a more robust server is generated and deployed by the dynamic model approach. When a server fails

because of an attack or internal failure, the buffer servers will take over the service for a brief period until the new immunized server is initialized. Since all of the buffer servers are susceptible to the same failures or attacks, it is a matter of time before the redundant servers are also infected for the same reason, especially when those servers are identical. Therefore, if the transition period is long, multiple buffer servers may be used before the immunized server is ready. The hybrid model ensures the availability of service to the client and provides more robust services in the end. However, this model needs more complex implementations than the two models previously discussed.

Trade Offs

In the static survivability model there are *n* numbers of redundant servers running in parallel that are deployed even before the operation starts. If one server fails, the broker delegates incoming requests to another server in the remaining server pool. Since the redundant components are ready to be used (unlike those in the dynamic model) during the failure or attack, the service downtime is relatively shorter than in the dynamic model. The implementation of the static survivability model is comparatively simpler than the dynamic or the hybrid model. In the static survivability model, it is unnecessary to maintain a server component factory, whose main job is to replace faulty server components with immunized ones. On the other hand, the static model has inherent disadvantages in terms of resource efficiency, adaptation, and robustness.

The hybrid model combines the logic of both the static and dynamic models. Here, if the monitor finds a fault in one of the server components, it informs the factory (as in the dynamic model) to replace the faulty component with an immunized one, and, in the meantime, it also sends a message to the broker to temporarily deploy a redundant component (as in the static model) until the new immunized component is built and deployed. Since the dynamic and hybrid models are able to build immunized components, they are more robust than the static model. The static model uses more memory resources than the dynamic model to maintain the redundant server components, making it less resource-efficient than the dynamic model. But this resource efficiency is accompanied by higher downtime in the dynamic model. The hybrid model is a balance between the static and dynamic models in terms of its resource efficiency.

CONCLUSION

In this chapter, we have described how mission-critical distributed systems can survive component failures or compromises with malicious codes in information warfare. We defined our definition of survivability, discussed the survivability challenges in a large mission-critical system in information warfare, and identified static, dynamic, and hybrid survivability models. Furthermore, we discussed the trade offs of each model. Technical details and implementation of the models are not described here because of space limitations.

Components can be immunized based on the generic or specific strategies provided by the monitors. If the monitor figures out the reasons for the failures or types of attacks on the components, it can provide specific strategies to the corresponding component factories. The level of immunization depends on the monitor's capability. The more powerful monitoring mechanisms and their communication channels between other components can provide more resistance to failures and attacks. Analyzing specific reasons for unexpected failures in run time is another challenge. Monitors can cooperate with other monitors, and different monitors are in charge of different interests. To detect and analyze types of attacks, the monitors can be associated with existing intrusion detection systems, which is also part of our future work.

REFERENCES

Avresky, D. R., Arlat, J., Laprie, J.-C., & Crouzet, Y. (1996). Fault injection for formal testing of fault tolerance. *IEEE Transactions on Reliability, 45*(3), 443-455.

Barga, R. S., Lomet, D. B., Baby, T., & Agrawal, S. (2000). Persistent client-server database sessions. In *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology* (pp. 462-477). Konstanz, Germany.

Barga, R., Lomet, D., Shegalov, G., & Weikum, G. (2004). Recovery guarantees for Internet applications. *ACM Transactions on Internet Technology (TOIT), 4*(3), 289-328.

Chen, M. Y., Kiciman, E., Fratkin, E., Fox, A., & Brewer, E. (2002). Pinpoint: Problem determination in large, dynamic Internet services. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)* (pp. 595-604). Washington, DC.

Dowling, J., & Cahill, V. (2004). Self-managed decentralized systems using k-components and collaborative reinforcement learning. In *Proceedings of the 1st ACM SIGSOFT workshop on self-managed systems (WOSS)* (pp. 39-43). New York.

Freytag, J. C., Cristian, F., & KŁhler, B. (1987). Making system crashes in database application programs. In *Proceedings of the 13th International Conference on Very Large Data Bases* (pp. 407-416). Brighton, UK.

Helsinger, A., Kleinmann, K., & Brinn, M. (2004). A framework to control emergent survivability of multi agent systems. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 28-35). Washington, DC.

Hsueh, M.-C., Tsai, T. K., & Iyer, R. K. (1997). Fault injection techniques and tools. *Computer, 30*(4), 75-82.

Jajodia, S., McCollum, C., & Ammann, P. (1999). Trusted recovery. *Communications of the ACM, 42*(7), 71-75.

Kapfhammer, G., Michael, C., Haddox, J., & Colyer, R. (2000). An approach to identifying and understanding problematic cots components. In *Proceedings of the Software Risk Management Conference (ISACC).* Reston, VA.

Knight, J., Elder, M., & Du, X. (1998). Error recovery in critical infrastructure systems. In *Proceedings of the Computer Security, Dependability, and Assurance (CSDA) Workshop.* Williamsburg, VA.

Knight, J., & Sullivan, K. (2000). Towards a definition of survivability. In *Proceedings of the 3rd Information Survivability Workshop (ISW).* Boston.

Lipson, H., & Fisher, D. (1999). Survivability: A new technical and business perspective on security. *Proceedings of the New Security Paradigms Workshop (NSPW),* Caledon Hills, Ontario, Canada.

Liu, P., Ammann, P., & Jajodia, S. (2000). Rewriting histories: Recovering from malicious transactions. *Distributed and Parallel Databases, 8*(1), 7-40.

Madeira, H., Costa, D., & Vieira, M. (2000). On the emulation of software faults by software fault injection. In *Proceedings of the International Conference on Dependable Systems and Networks (DNS)* (pp. 417-426). Washington, DC.

Park, J. S., & Chandramohan, P. (2004). Component recovery approaches for survivable distributed systems. In *Proceedings of the 37th Hawaii International Conference on Systems Sciences (HICSS-37).* Big Island, HI.

Park, J. S., Chandramohan, P., Devarajan, G., & Giordano, J. (2005). Trusted component sharing by runtime test and immunization for survivable distributed systems. In *Proceedings of the 20th IFIP International Conference on Information Security (IFIP/SEC 2005).* Chiba, Japan.

Park, J. S., & Froscher, J. N. (2002). A strategy for information survivability. In *Proceedings of the 4th Information Survivability Workshop (ISW).* Vancouver, Canada.

Ring, S., Esler, D., & Cole, E. (2004). Self-healing mechanisms for kernel system compromises. In *Proceedings of the 1st ACM SIGSOFT workshop on self-managed systems (WOSS)* (pp. 100-104). New York.

Voas, J. M., & McGraw, G. (1998). *Software fault injection: Inoculating programs against errors.* Wiley Computer Publishing.

KEY TERMS

Commercial Off-the-Shelf (COTS): These software systems are developed to interoperate with existing systems without the need for customization and available for sale to public.

Software Component: This reusable software element provides predefined services and can communicate with other components with relatively little effort.

Spyware: This software program monitors a user's computer activity surreptitiously or collects sensitive information about users or organizations without their knowledge.

Survivability: This refers to the capability of an entity to continue its mission even in the presence of damage to the entity.

Trojan Horse: A rogue software program installed on the victim's machine that can run secretly with the user's privileges.

Trusted Component: This software component has not been modified by an unauthorized manner, since it was created by a trusted source and functions in an expected way.

Virus: This software program attaches itself to other programs, propagating itself in this way, and does something, usually malicious and unexpected.

This work was previously published in Cyber Warfare and Cyber Terrorism, edited by L. Janczewski & A. Colarik, pp. 403-411, copyright 2008 by Information Science Reference (an imprint of IGI Global).

Chapter 8.13
A Novel Application of the P2P Technology for Intrusion Detection

Zoltán Czirkos
Budapest University of Technology and Economics, Hungary

Gábor Hosszú
Budapest University of Technology and Economics, Hungary

INTRODUCTION

The importance of the network security problems come into prominence by the growth of the Internet. This article presents a new kind of software that uses the network itself to protect the hosts and increase their security. The hosts running this software create an application level network (ALN) over the Internet (Hosszú, 2005). Nodes connected to this ALN check their operating systems' log files to detect intrusion attempts. Information collected this way is then shared over the ALN to increase the security of all peers, which can then make the necessary protection steps, for example, blocking network traffic by their own *firewall*.

Different kinds of security software utilizing the network were also written previously (Snort, 2006). The novelty of Komondor is that its client software entities running in each host create a *peer-to-peer* (P2P) *overlay network* (Czirkos, 2006). Organization is automatic; it requires no user interaction. This network model ensures stability, which is important for quick and reliable communication between nodes. By this build-up, the system remains useful over the unstable network.

THE IMPORTANCE OF THE P2P COMMUNICATIONS

The Internet-based communication technology enabled people to share information with anybody in seconds. This has brought benefits to people spanning many spheres from social services to education (Frasz, 2005). Probably the best example of such extended network of content sharing is

the P2P that allows users to download media files off other computers free of charge. Once content enters the Internet, it can be downloaded by an unlimited number of people.

One of the latest steps in the steady advances in P2P technologies is the release of new P2P technologies in 2005 that enable a user community to filter out mislabeled or corrupt files (Goth, 2005). One approach to build a more trustworthy P2P overlay is the application credence (Sirer & Walsh, 2005). It rates a certain network object instead of a given peer node for trustworthiness. The reason is that nodes can be inhabited by various people over time, but the data in the object itself does not change. This system uses a secure and anonymous voting mechanism. Over time, users with similar votes or the legitimacy of a file will dynamically form a kind of community enabling enough correlation of trust. Similarly, a user that systematically answers contrarily will get an equally significant negative weighting; however, an inconsistent voter will have less statistical weight. In such a way, the more users who join credence overlay, the more accurate an overall rating each file will receive.

The trend of the P2P systems is building more resilient services. Centralized solutions are fragile, since a single link breakage in the network can cut access to the whole service. P2P enables higher ability to construct overlays that self-organizes and recovers from failures.

Another interesting and important feature of the development process of the P2P technology is that the most successful projects are open sources such as LimeWire, which is a Gnutella client with rapidly growing popularity (Bildson, 2005). Its business model has two sides. One version is free, however, advertising-supported, and the other is ad-free, but the users must pay for it. LimeWire guarantees no bundled software with downloads. The open source property of the LimeWire encourages its users to monitor its development. The largest competitor of LimeWire is BitTorrent, which is very efficient in sharing large files (BitTorrent, 2006). Its users upload portions of required documents to a requester instead of forcing one client to upload the whole file many times.

THE PROBLEM OF THE INTRUSION

Computers connected to networks are to be protected by different means (Kemmerer & Vigna, 2002). Information stored on a computer can be personal or business character, private or confidential. An unauthorized can person can therefore steal it; its possible cases are shown in Table 1.

We have to protect not only our data, but also our resources. Resources are not necessarily hardware only. Typical types of attack are to gain access to a computer to initiate other attacks from it. This is to make the identification of the attacker more difficult because this way the next intruded host in this chain sees the IP address of the previous one as its attacker.

Stored data can not only be stolen, but changed. Information modified on a host is extremely useful to cause economic damage to a company. The attacker can alter or obstruct its functioning properly and cause damage.

Intrusion attempts, based on their purpose, can be of different methods. But these methods share things in common, scanning networks ports or subnetworks for services, and making several

Table 1. The types of the information stealth

• An unauthorized person gains access to a host.
• Monitoring or intercepting network traffic by someone.
• An authorized but abusive user.

attempts in a short time. This can be used to detect these attempts and to prepare for protection.

Simple, low strength passwords are also a means of security holes. These are used by the so-called dictionary method, trying to log into the system with common names or proper names as somebody's password. They are of a relatively small number and easily guessable.

The attacker can also be trying to find resources through security holes. With this type of action, whole ranges of network addresses are scanned for a particular service having a bug or just being badly configured. The port number is fixed here. An example for this is scanning for an open e-mail (SMTP) relay to send junk mail anonymously.

A common feature of the attack methods described above is that the attacker makes *several attempts* against a host. The Komondor software developed by us uses this as a base. As one host running the Komondor detects an intrusion attempt and shares the address of the attacker on the overlay network, the other ones can prepare and await the *same attacker* in safety, who will usually arrive sooner or later.

Traditionally, organizations have relied on their firewall to enforce their corporate policies. To stop the use of P2P file sharing, organizations may add a rule that denies outbound ports not required for business (Sorensen & Richards, 2004). Unfortunately, many of the P2P applications today use a "port-hopping" method of communication to circumvent firewall rule sets that limit outbound connections to specifically allowed ports. If the firewall restricts the ports permitted to establish outbound connections to only the essential ports, such as port 80 (HTTP) and port 25 (SMTP), the P2P application modifies the port that it uses to communicate with other P2P nodes to use these ports allowed through the firewall.

To effectively detect this type of P2P application traffic in an environment requires the use of a device that can examine the contents of the packets allowed through the firewall. *Intrusion detection systems* (IDS) were designed to satisfy this need. These systems are designed to monitor network traffic to look for known signature attack patterns and/or deviations from protocol specifications that represent malicious intent. When potentially malicious traffic is observed, they generate an alert. More importantly, these "detection" technologies lack the capabilities to effectively prevent this traffic, leaving the burden with the administrator to manually investigate and respond.

WAYS OF PROTECTION

The P2P based file sharing software can cause various security problems. There are file sharing programs, which install further software known as spyware. Spyware monitors the browsing habits of the user and then sends the collected information to third parties. Typically, the user gets advertisements based on the data collected by the spyware. This kind of software can be hard to sense and to clean from the system. Before installing any file sharing program, the user should buy antivirus software that can detect the already downloaded spyware on the user's hard drive and prevent the operating system from downloading further spyware programs.

The active network vulnerability scanners such as the NMAP (2006), Nessus (2005), and ISS Scanner (ISS Scanner, 2005) send packets or communicate in some manner with the systems they are auditing. Naturally, they are bound by the physical limitations of the networks and systems they are auditing to send and receive these packets. In other words, scanning can take a long time, especially for large networks (Deraison, Gula, & Hayton, 2005).

In some rare cases, the act of probing may cause instability in the audited system. Network devices such as routers and switches may also be affected by large numbers of port scans, host enumeration, and vulnerability testing. Also,

networks change all too often. After the scan has been finished, it slowly becomes out of date as the network changes.

Active network vulnerability scanners may also have a political stigma within large organizations. For a variety of reasons, a system administrator may feel that there is no need to have a third party scan their systems. To compensate for this, a *passive vulnerability scanner* can be deployed in a manner to watch these "off limits" networks and report on their vulnerabilities.

Passive vulnerability scanning is the process of monitoring network traffic at the packet layer to determine topology, services, and vulnerabilities (Deraison et al., 2005). The passive scanners, such as NeVO, monitor for client and server vulnerabilities of a specific network through direct analysis of the packet stream. As NeVO observes network packets, it builds a model of the active hosts on a network and their services. For example, observing a TCP port 25 SYN-ACK packet from one of the monitored hosts will cause NeVO to reevaluate the network model.

NeVO uses different methods to determine if a host is alive and what the host is. It reconstructs both sides of a network conversation and then analyzes the data for evidence of specific client or server vulnerabilities. Unique client and servers for protocols such as HTTP, SMTP, and FTP have unique strings which identify the version of the service. NeVO also makes use of passive operating system identification by monitoring SYN packets which may be emitted by the system during network usage. Each operating system, such as Windows or Linux, builds its SYN packets in its own unique way, and this can be used to distinguish between some specific types.

Passive vulnerability scanning is not a replacement for active vulnerability scanning, but it is a very efficient technology. When it is deployed on its own, the passive vulnerability scanner will produce very interesting information about the security profile of a monitored network. However, this is by no means a full view of network security. When deployed together, both methods will efficiently detect vulnerabilities and changes to the monitored networks (Deraison et al., 2005).

The ability to correctly identify the attacks on the Internet is very important. The exchange of alert information between organizations can greatly supplement the knowledge obtained from local monitors (Locasto, Parekh, Keromytis, & Stolfo, 2005). The IDS usually operates within one administrative domain. Therefore, information about the global state of network attack patterns is usually unexamined. However, global information exchange can help organizations in ranking and addressing threats, which they would not have otherwise identified.

In order to construct an efficient IDS, a lot of problems must be solved before intrusion alert data can be safely distributed among cooperating organizations. For example, distributing all alert information in a P2P overlay quadratically increases bandwidth requirements. Also, information exchange between cooperating organizations must be carefully managed, since confidential network information can be distributed with the alert information by chance.

There are developments to construct a P2P system that can detect and identify attacks in a distributed manner. Krugel, Toth, and Kerer (2001) proved that only a small number of information exchanges need to be exchanged to determine an attack in progress. Their results confirm that a P2P solution is feasible and usable for creating decentralized IDS. They supposed that the attack signatures are known by the systems before the intrusion takes place. The DOMINO overlay system (Yegneswaren, Barford, & Jha, 2004) applies the Chord document routing protocol (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001) to disseminate alert data based on a hash of the source IP address. The DShield project uses a centralized repository to receive intrusion alerts from a lot of distributed sources (Ullrich, 2005). The DShield supports the exchange of information among administrative domains, too.

ALGORITHMS USED IN THE DEVELOPED SYSTEM

This section introduces a novel method, which utilizes the results of the existing IDS methods and efficiently combines the individual security solutions of the operating systems with the P2P technology to construct an overlay of the developed client programs running on individual hosts. The most important algorithms making up the proposed system will also be presented.

Table 2. The design goals of the proposed system

• Creating a stable overlay network to share information. • Reports of intrusions should spread as fast as possible over the network. • Decentralized system, redundant peers. • Masking the security holes of each peer based on the reports.

Figure 1. Architecture of the Komondor system

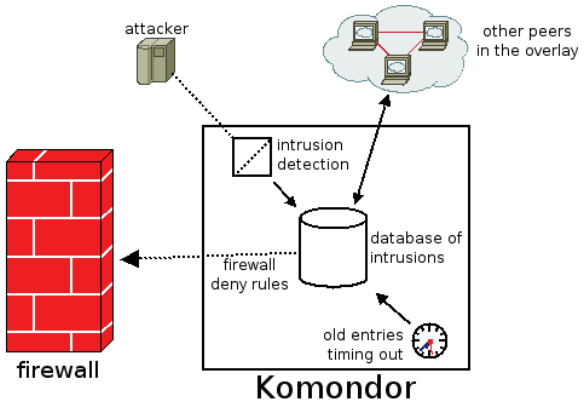
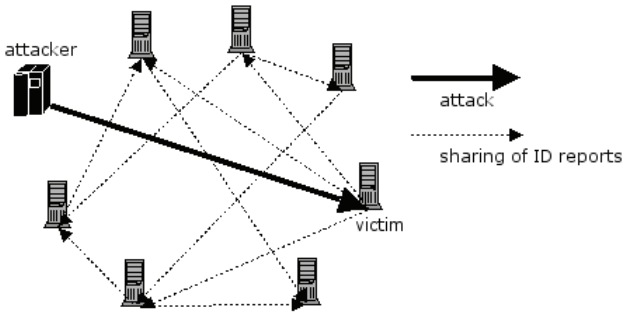


Figure 2. Attack against a Komondor node



The use of the peer-to-peer network model for this purpose is new in principle. Test results proved its usefulness, with its aid were not only simulated, but also real intrusion attempts blocked. The design goal of the system is listed in Table 2.

The proposed method is implemented in software called *Komondor*, which got its name from the famous Hungarian guard dog. The Komondor software is intended to mask the security holes of services provided by the host, not to repair them. For this, it does not need to know about the security hole in detail. It can provide some protection in advance, but only if somewhere on the network an intrusion was already detected. It does not fix the security hole, but keeps the particular attacker from further activity. If the given security hole is already known, it is worth rather fixing that itself.

The inner architecture of Komondor is presented in Figure 1. Different hosts run the uniform copies of this program, monitoring the occurring network intrusion attempts. If one of the peers detects an attempt on a system supervised, it takes two actions:

1. Strengthens the protection locally, by configuring the firewall to block the offending network address.
2. Informs the other peers about the attempt.

Komondor nodes protect each other this way. If an intrusion attempt was recorded by a node, the other ones can prepare for the attack in advance. This is shown in Figure 2.

Information about intrusion attempts is collected by two means: intrusion detected by the node in question or intrusion detected by another one. The first working version of Komondor monitors system log files to collect information. These log files can contain various error messages, which may refer to an intrusion attempt. Possible examples are log-in attempt with an inexistent user name or several attempts to download an inexistent file through a HTTP server (Czirkos, 2005). The actual topology and related data of the Komondor overlay is continuously displayed on the Komondor project page (Czirkos, 2006).

CONCLUSION

The article reviewed some security-related problems of the P2P virtual networks and presented many useful and applicable approaches, which address the problem of the intrusion detection. A novel application of P2P theory was also introduced that helps the users of this system to increase security of their hosts.

The developed system is easy to use; the nodes organize the P2P overlay automatically and do not need any user interaction. It proves the reasonability of a security application utilizing local protection and P2P theory together. The dependability of a P2P overlay ensures that intrusion alerts will reach all hosts which need protection. The self-organizing property of such an overlay also ensures that it will remain functional in an environment with a high number of network link failures.

REFERENCES

Bildson, G. (2005). *LimeWire home page*. Retrieved February 26, 2008, from http://www.limewire.com/english/content/home.shtml/

BitTorrent. (2006). *Bittorrent home page*. Retrieved February 26, 2008, from http://www.bittorrent.com

Czirkos, Z. (2005, November 11). Development of P2P based security software [in Hungarian]. In *Proceedings of the Conference of Scientific Circle of Students* (Second Award), Budapest, Hungary.

Czirkos, Z. (2006). *The Komondor project page with dynamic monitoring of the Komondor overlay.* Budapest University of Technology & Economics, Department of Electron Devices. Retrieved February 26, 2008, from the http://jutas.eet.bme.hu

Deraison, R., Gula, R., & Hayton, T. (2005). *Passive vulnerability scanning: Introduction to NeVO.* Tenable Network Security. Retrieved February 26, 2008, from http://www.tenablesecurity.com

Frasz, A. (2005). Download this article. *The Platform, 4*(2). AEA Consulting. Retrieved February 26, 2008, from http://www.aeacondulting.com

Goth, G. (2005). A (P2)Perfect Storm. *IEEE Distributed Systems Online, 6*(5).

Hosszú, G. (2005). Mediacommunication based on application-layer multicast. In S. Dasgupta (Ed.), *Encyclopedia of virtual communities and technologies* (pp. 302-307). Hershey, PA: Idea Group Reference.

ISS Scanner. (2005). *The ISS Scanner home page.* Retrieved February 26, 2008, from http://www.iss.net

Kemmerer, R. A., & Vigna, G. (2002). Intrusion detection: A brief history and overview. Security & Privacy-2002., *Supplement to Computer Magazine.* IEEE Computer Society, pp. 27-30.

Krugel, C., Toth, T., & Kerer, C. (2001, December). *Decentralized event correlation for intrusion detection.* Paper presented at the International Conference on Information Security and Cryptology (ICISC).

Locasto, M.E., Parekh, J.J., Keromytis, A.D., & Stolfo, S.J. (2005, June 15-17). Towards collaborative security and P2P intrusion detection. In *Proceedings of the 2005 IEEE Workshop on Information Assurance Security*, United States Military Academy, West Point, NY (pp. 30-36).

Nessus. (2005). *The Nessus home page.* Retrieved February 26, 2008, from http://www.nessus.org

NMAP. (2006). *NMAP free security scanner, tools & hacking resources.* Retrieved February 26, 2008, from http://www.insecure.org/

Sirer, E.G., & Walsh, K. (2005). *Credence home page.* Cornell University. Retrieved February 26, 2008, from http://www.cs.cornell.edu/People/egs/credence/

Snort. (2006). *Snort: The de facto standard for intrusion detection/prevention.* Retrieved February 26, 2008, from http://www.snort.org

Sorensen, S., & Richards, S. (2004). *Preventing peer-to-peer file sharing activity: Mitigating the risks with Juniper Networks NetScreen-IDP product line.* Juniper Networks, Inc. Retrieved February 26, 2008, from http://www.juniper.net

Stoica, I., Morris, R., Karger, D., Kaashoek, F., & Balakrishnan, H. (2001, August). Chord: A scalable peer-to-peer lookup service for Internet application. In *Proceedings of ACM SIGCOMM.*

Ullrich, J. (2005). *Dshield home page.* Retrieved February 26, 2008, from http://www.dshield.org/

Yegneswaran, V., Barford, P., & Jha, S. (2004, February). *Global intrusion detection in the DOMINO overlay system.* Paper presented in the ISOC Symposium on Network and Distributed Systems Security.

KEY TERMS

Active Network Vulnerability Scanner: Such systems send packets and communicate in some manner with the systems they are auditing.

Application Level Network (ALN): The applications, which are running in the hosts,

can create a virtual network from their logical connections. This is also called ***overlay network***. The operations of such software entities are not able to understand without knowing their logical relations. ALN software entities usually use the ***P2P model***, not the ***client/server*** model, for the communication.

Client/Server Model: A communicating way, where one host has more functionality than the other. It differs from the ***P2P model***.

Firewall: This is a host or router which provides a strict gateway to the Internet for a subnetwork, checking traffic and maybe dropping some network packets.

Intrusion Detection System (IDS): Examines the contents of the packets allowed through the firewall. It monitors network traffic to look for known signature attack patterns. When the malicious traffic is observed, the IDS generates an alert.

Overlay Network: The applications, which create an ***ALN***, work together and usually follow the ***P2P communication model***.

Passive Network Vulnerability Scanner: Monitors network traffic at the packet layer to determine topology services. They also try to identify the vulnerabilities of the client and the server in a specific network through direct analysis of the packet stream.

Peer-to-Peer (P2P) Model: A communication way where each node has the same authority and communication capability. They create a virtual network, overlaid on the Internet. Its members organize themselves into a topology for data transmission.

This work was previously published in Encyclopedia of Information Communication Technology, edited by A. Cartelli & M. Palma, pp. 616-621, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.14
Reducing the Complexity of Modeling Large Software Systems

Jules White
Vanderbilt University, USA

Douglas C. Schmidt
Vanderbilt University, USA

Andrey Nechypurenko
Siemens AG, Germany

Egon Wuchner
Siemens AG, Germany

ABSTRACT

Model-driven development is one approach to combating the complexity of designing software intensive systems. A model-driven approach allows designers to use domain notations to specify solutions and domain constraints to ensure that the proposed solutions meet the required objectives. Many domains, however, require models that are either so large or intricately constrained that it is extremely difficult to manually specify a correct solution. This chapter presents an approach to provide that leverages a constraint solver to provide modeling guidance to a domain expert. The chapter presents both a practical framework for transforming models into constraint satisfaction problems and shows how the Command Pattern can be used to integrate a constraint solver into a modeling tool.

INTRODUCTION

Model-driven development (MDD) (Ledeczi, 2001a; Kent, 2002; Kleppe, Bast, & Warmer, 2003; Selic, 2003) is a promising paradigm for

software development that combines high-level visual abstractions—specific to a domain—with constraint checking and code-generation to simplify the development of a large class of systems (Sztipanovits & Karsai, 1997). MDD tools and techniques help improve software quality by automating constraint checking (Sztipanovits & Karsai, 1997). For example, in developing a software system for an automobile, automated constraint checking can be performed by the MDD tool to ensure that components connected by the developer, such as the antilock braking system and wheel RPM sensors, send messages to each other using the correct periodicity. An advantage of model-based constraint checking is that it expands the range of development errors that can be caught at design time rather than during testing.

Compilers for third-generation languages (e.g., Java, C++, or C#) can be viewed as a form of model-driven development (Atkinson & Kuhne, 2003). A compiler takes the third-generation programming language instructions (model), checks the code for errors (e.g., syntactic or semantic mistakes), and then produces implementation artifacts (e.g., assembly, byte, or other executable codes). A compiler helps catch mistakes during the development phrase and automates the translation of the code into an executable form.

Domain-specific Modeling Languages (DSML) (Ledeczi, 2001a) are one approach to MDD that use a language custom designed for the domain to model solutions. A metamodel is developed that describes the semantic type system of the DSML. Model interpreters traverse instances of models that conform to the metamodel and perform simulation, analysis, or code generation. Modelers can use a DSML to more precisely describe a domain solution, because the modeling language is custom designed for the domain.

MDD tools for DSMLs accrue the same advantages as compilers for third-generation languages. Rather than specifying the solution in terms of third-generation programming languages or other implementation-focused terminology, however, MDD allows developers to use notations specific to the domain. With a third-generation programming language approach (such as specifying the solution in C++), high-level information (such as messaging periodicity or memory consumption) is lost. Because a C++ compiler does not understand messaging periodicity (i.e., it is not part of the "domain" of C++ programs) it cannot check that two objects communicate at the correct rate.

With an MDD-based approach, in contrast, DSML developers determine the granularity of the information captured in the model. High-level information like messaging periodicity can be maintained in the solution model and used for error checking. By raising the level of abstraction for expressing design intent, more complex requirements can be checked automatically by the MDD tool and assured at design time rather than testing time (Sztipanovits & Karsai, 1997), as seen in Figure 1. In general, errors caught during the design cycle are much less time consuming to identify and correct than those found during testing (Fagan, 1999).

As model-based tools and methodologies have developed, however, it has become clear that there are domains where the models are so large and the domain constraints so intricate that it is extremely hard for modelers to handcraft correct or high quality models. In these domains, MDD tools that provide only solution-correctness checking via constraints provide few real benefits over the third-generation programming language approach. Even though higher-level requirements can be captured and enforced, developers must still find ways of manually constructing a model that adheres to these requirements.

Distributed real-time and embedded (DRE) systems are software intensive systems that require guaranteed execution properties (e.g., deadlines), communication across a network, or must operate with extremely limited resources. Examples of DRE systems include automobile

safety and aircraft autopilot systems. Inherent complexities in DRE systems is their large model sizes and the combinatorial nature of their constraints—not code construction per se. Specifying the deployment of software components to Electronic Control Units (ECUs, which are the automotive equivalent of a CPU) in a car, while observing configuration and resource constraints, can easily generate solution spaces with millions or more possible deployments. For these large modeling problems, it is impractical (if not impossible) to create a complete and valid model manually.

To illustrate the complexities of scale, consider a group of 10 components that must be deployed to one of 10 ECUs within a car. There are $9^{\wedge 9}$ = *387,420,489 unique deployments* that could be tried. Part of the complexity of a DRE system model is how quickly the solution space grows as the number of model elements increases. Figure 2 depicts the speed at which the solution space grows for our automotive example.

Clearly, any approach to finding a deployment that observes the constraints must be efficient and employ a form of pruning to reduce the time taken to search the solution space. A manual approach may work for a model with five or so elements. As shown in Figure 2, however, the solution space can increase rapidly as the number of elements grows, which render manual solutions infeasible for nontrivial systems.

Each component in an automobile typically has multiple constraints governing its placement. For example, an Antilock Braking System (ABS) must be hosted by a controller at least a certain distance from the perimeter of the car to enhance survivability in a crash. Moreover, the ABS will have requirements governing the CPU, memory, and bus bandwidth available on its host. When these constraints are considered for all the components, it becomes hard for modelers to handcraft correct solutions. The example in Figure 2 only has nine components and nine control units. Real automotive models typically contain 80 or more

Figure 1. Complexity of identifiable errors increases with level of abstraction

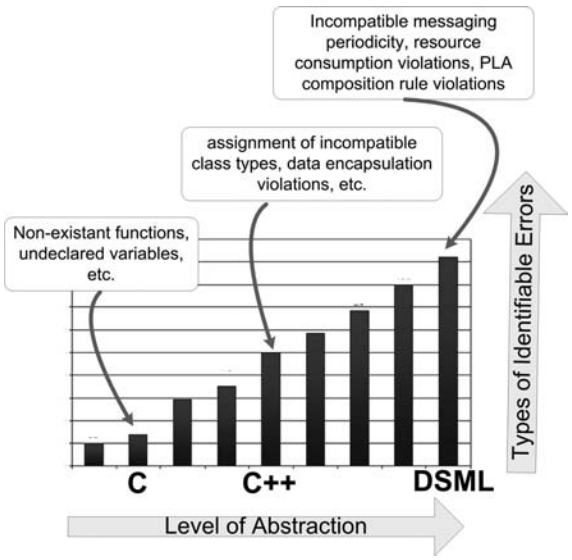
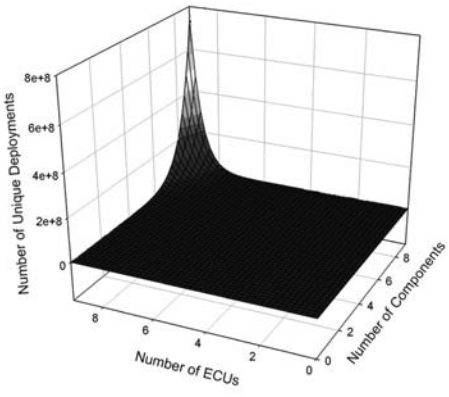


Figure 2. Number of unique deployments vs. model size



control units and hundreds of components. In models of this scale, manual approaches simply cannot handle the large numbers of possibilities and the complexity of the constraints.

The remainder of this chapter is organized as follows. The *Background* section illustrates the specific challenges of using MDD tools for these types of complex domains; The *Modeling Guidance* section presents techniques based on integrating constraint solvers into modeling environments that can be used to address these challenges; The *Future Research Directions* section describes future trends in modeling software intensive systems; and the final section presents concluding remarks.

BACKGROUND

Current Modeling Languages and Tool Infrastructure

There are a plethora of technologies and standards available for building MDD tools. This section explores some of the main frameworks, tools, and specifications that are available to develop model-driven processes for software systems.

Domain-Independent Modeling Languages

On one end of the MDD tool spectrum are Unified Modeling Language (UML) (Fowler & Scott, 2000) based tools, such as IBM's Rational Rose (Quatrani, 2003), that focus on building UML and UML-profile (Fowler & Scott, 2000) based models. When using UML, all models and languages must be specializations of the UML language. UML provides a single generic language to describe all domains. The advantage of the domain-independent approach of UML-based tools is the increased interoperability between modeling platforms that can be obtained by describing models using a single modeling language and the wide acceptance of the language. New languages can be constructed on top of UML by defining profiles, which are language extensions. UML is based on the MOF metamodel specified by the OMG.

UML is well established in software development. More recently, numerous extensions to the language based on profiles have been developed. *SecureUML* (Lodderstedt, Basin, & Doser, 2002) provides security-related modeling capabilities to UML. *Embedded UML* (Martin, Lavagno, & Louis-Guerin, 2001) is another profile available for UML that provides DRE-specific extensions, such as timing properties of components. The UML extension approach allows developers to customize the language to meet the application domain, while still maintaining (some degree of) compatibility between tools.

Domain-Specific Modeling Languages

On the other end of the MDD tool spectrum are domain-specific modeling language (DSML) (Ledeczi, 2001a) tools. In contrast to UML, DSML tools do not necessarily share a common metamodel or language format. This freedom allows DSMLs to have greater expressivity and handle domains (such as warehouse management, automotive design, and product line configuration), that contain concepts (such as spatial attributes) that are not easily expressed and visualized using UML-based tools. The drawback of DSMLs, however, is that choosing a language generally ties a development process not only to a specific way of representing the model but also generally to a specific tool. Although the loss of interoperability can be problematic, transformations can be written to convert between model formats and still achieve tool interoperability. In many cases, the greater expressivity gained by using a DSML can greatly improve the usability of the MDD tool.

Tools for Building DSMLs

To build a DSML, a metamodeling language must be used to define the syntax of the language. A metamodel describes the rules that determine the correctness of a model instance and specifies the types that can be created in the language. The OMG's current standard is the Meta-Object Facility (MOF) (Object Management Group, 2007) language. MOF provides a metamodel language, similar to UML, that can be used to describe other new languages. MOF itself is recursively defined using MOF. MOF is a specification and therefore is not wedded to a particular tool infrastructure or language technology. Many DSMLs can be described using MOF.

Another popular metamodeling language is the Eclipse Modeling Framework's (EMF) (Moore, 2004) Ecore language. Ecore has nearly identical language constructs to MOF but is a concrete implementation rather than a standard specification. Developers can describe DSMLs using Ecore (Moore, 2004) and then leverage EMF to automatically generate Java data structures to implement the DSML. EMF also possesses the capability to generate basic tree-based graphical editing facilities for Eclipse that operate on the Java data structures produced by EMF.

Complex diagram-like visualizations of EMF-based modeling languages can be developed using the Graphical Editor Framework (GEF) for Eclipse (Moore, 2004). GEF provides the fundamental patterns and abstractions for visualizing and interacting with a model. Editors can be developed using GEF that allow modelers to draw connections to create associations, nest elements to develop containment relationships, and edit element attributes. GEF editors are based on the Model, View, Controller (MVC) pattern (Gamma, Helm, Johnson, & Vlissides, 1995). GEF, however, requires complex graphical coding.

The Graphical Modeling Framework (GMF) (Graphical Modeling Framework, 2007), is higher level framework, built on top of GEF, that simplifies the development of graphical editors. GMF automates the construction of the controller portion of GEF editors and provides a set of reusable view classes. MVC controllers are developed using GMF by creating complex XML files that map elements and their attributes to views in the model. GMF takes the XML mappings of elements to views and generates controllers that developers can use to synchronize the model and view of the MDD tool automatically.

Even with the powerful development frameworks presented thus far, developing a visual MDD tool requires significant effort. *Metaprogrammable* modeling environments (Ledeczi, 2001a) help alleviate this effort by allowing developers to specify the metamodel for a DSML visually. After the visual specification for the language is complete, the metaprogrammable modeling environment can automatically generate the appropriate code and configure itself to provide graphical editing capabilities for the modeling language.

Metaprogrammable modeling environments also provide complex remoting, model traversal, library, and other capabilities that are hard to develop from scratch. Two examples of these environments are the Generic Modeling Environment (GME) (Ledeczi, 2001b), which a windows-based metaprogrammable MDD tool, and the Generic Eclipse Modeling System (GEMS) (Generic Eclipse Modeling System, 2007), a part of the Eclipse Generative Modeling Technologies (GMT) project. The main tradeoff in using metaprogrammable modeling environments is that they tend to provide less flexibility in the visualization of the model.

Constraint Checking with OCL

Many modeling techniques rely on a constraint specification language to provide correctness checking rules that are hard to concisely describe using a graphical language. Certain types of constraints that specify conditions over multiple

types of modeling elements, not necessarily related through an interface or inheritance, are more naturally expressed using a textual constraint specification language. The constraint language rules are run against instances of the UML, EMF, or other models to ensure that domain constraints are met. Constraint failures are returned to the modeler through the use of popup windows or other visual mechanisms.

The OMG Object Constraint Language (OCL) (Warmer & Kleppe, 1998) is a standard constraint specification for modeling technologies. OCL allows developers to specify invariants, preconditions, and postconditions on types in the modeling language. For example, the OCL constraint:

```
context ECU
inv: self.hostedComponents->collect(x
        | x.requiredRAM)->sum() < self.RAM
```

can be used to check that the sum of the RAM demands of the components hosted by an ECU do not exceed the available RAM on the ECU. The first line of the OCL rule defines the context or the type to which the OCL rule should be applied. The second part of the rule, beginning with "inv," defines the invariant condition for the rule. When there is a change to a property of a modeling element of the context type, the invariant conditions for the rules applicable to the element must be checked. Invariants that do not hold after the modification are flagged as errors in the MDD tool.

OCL works well for localized constraints that check the correctness of the properties of a single modeling element. As described earlier, however, the rule can only be used to check the correctness of the state of a modeling element and not to *derive valid states for a modeling element*, which is a process called ***backward chaining*** (Ginsberg, 1989). In a modeling context, backward chaining is a process whereby the MDD tool deduces correct modeling actions based on the domain constraints. For example, if it were possible to use the above OCL rule to backward chain, a MDD tool could not only determine whether or not an ECU was in a correct state but also, given the current state of an ECU, produce a list of components that could be hosted by the ECU without violating the rule.

For software systems with global constraints and large models, the inability of traditional modeling and constraint checking approaches, such as OCL, to not only flag errors but deduce solutions limits the utility of model-based development approaches. Backward chaining (providing modeling guidance) becomes more important as domains become more complex, and where it is thus harder to handcraft solutions.

Emerging Modeling Challenges

Deriving Solutions that Meet a Global Constraint

The increasing proliferation of DRE systems is leading to the discovery of further hard modeling problems. These domains all tend to exhibit problems, such as scheduling with resource constraints (Yuan & Nahrstedt, 2003), that are exponential in complexity because they are different types of NP problems. A key challenge in developing effective and scalable DSMLs and models for these domains is deriving the overall organization and architecture of MDD tools and software platforms that can simultaneously meet stringent resource, timing, or cost constraints.

Mobile devices are a domain that have become widely popular and typically exhibit tight resource constraints that must be considered when designing software (Forman & Zahorjan, 1994). Software design decisions, such as the CPU demand of the application, often have physical impacts on the device as well. For example, the scheduling of and workload placed on the CPU can affect the power consumed by the device. Poor scheduling or resource allocation decisions can therefore limit battery life (Yuan & Nahrstedt, 2003).

Determining the appropriate scheduling policies and application design decisions to handle the resource constraints of mobile devices is critical. Without the proper decisions, devices can have limited battery life and usability. Scheduling with resource constraints, however, is an NP problem (Cormen, Rivest, Leiserson, & Stein, 1990) and thus cannot be solved manually for nontrivial problems.

Adhering to Nonfunctional Requirements

Another challenge of DRE systems is that they often exhibit numerous types of nonfunctional QoS requirements that are hard to handle manually. For example, in automotive development, an application may have communication timing constraints on the real-time components (e.g., antilock braking control), resource constraints on components (e.g., infotainment systems), and feature requirements (e.g., parking assistance) (Weber & Weisbrod, 2002). In environments with this range of QoS requirements, a correct design must solve numerous complex problems and solve them in a layered manner so the solutions are compatible.

For example, the placement of two components on particular ECUs may satisfy a timing constraint but cause a resource constraint failure for another component, such as the infotainment system. Not only must modelers be able to solve numerous types of individually challenging problems, therefore, but they must be able to find solutions that meet all of the requirements.

Another area where complex constraints are common is in configuration management, which is key in emerging software development paradigms, such as product-lines (Jaaksi, 2002) and feature modeling (Antkiewicz & Czarnecki, 2006). In these domains, applications are built from reusable software components that interact through a common set of interfaces or framework. Applications are assembled using existing software assets for specific requirement sets. For example, in mission critical avionics product lines, such as Boeing Bold Stroke (Schmidt, 2002), the correct software component to update the HUD display is selected based on the timing, memory, and other requirements of the particular airframe to which the software is being deployed. Configuration-driven domains exhibit the same characteristics of computationally complex constraints that drive overall system organization as other complex domains.

The remainder of this chapter presents an approach to using a constraint solver integrated into a modeling environment to address these challenges. First, the chapter introduces the types of modeling assistance that can be provided to help alleviate these challenges. Second, the chapter illustrates how a constraint solver can be used to provide these types of modeling assistance. Finally, an architecture for integrating Prolog into a modeling environment as a constraint solver is described.

MODELING GUIDANCE

This section illuminates the challenges of modeling software intensive systems and then presents an approach to providing modelers with modeling guidance from a constraint solver. Specific emphasis is placed on how modeling guidance can be used to reduce the complexity of modeling software intensive systems. Finally, the chapter illustrates how a constraint solver can be integrated into a graphical modeling tool.

Measuring Domain Complexity

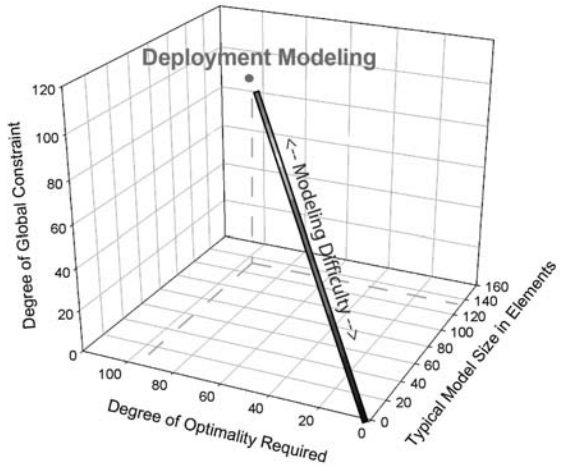
The complexity of modeling an arbitrary domain can be measured along the following three axes:

- **Typical model size in elements:** Large models are harder to work with using a manual approach. Clearly, modelers are more apt to

make mistakes managing—and much more likely to have trouble visualizing—a domain with hundreds of model elements than one with dozens of model elements.

- **Degree of global constraint:** Global constraints, such as resource constraints, that are dependent on multiple modeling steps or the order of modeling steps make a domain much harder to work with. For example, a constraint requiring the deployment of an ABS component to a single ECU at a certain distance from the perimeter of the car is relatively easy to solve. It is much harder to solve constraints of an ABS component requiring its deployment to two ECUs, both a minimum distance from the outside of the car and a minimum distance from each other (for fault tolerance guarantees).
- **Degree of Optimality Required:** Optimality is hard to achieve with a manual modeling approach. In many domains, such as manufacturing, a small increase in the cost of a solution can lead to a dramatic increase in the overall cost of manufacturing when the millions of units affected by the change are considered. Many solutions must therefore be tried to find the best one. Domains that require optimal or good answers are much more challenging to model.

Figure 3. Axes of measuring modeling complexity



The three axes described above can be used to categorize and evaluate different modeling domains. The difficulty of modeling a domain can be viewed as the distance of the domain from the origin when plotted according to its degree of global constraint, degree of required model optimality, and typical model size, as shown in Figure 3.

Key Challenges of Complex Domains

The key reasons that manual modeling approaches do not scale as a modeling domain moves further and further along the axes, shown in Figure 3, away from the origin is:

1. When there are thousands, millions, billions, or more possible ways that a model can be constructed and few correct ones, finding a valid solution is hard.
2. A valid solution may not be a good solution in these domains. Often, a modeler may find a solution that is valid but is far from the optimal solution. Automation and numerical methods, such as the Simplex method (Nelder & Mead, 1965), are needed to efficiently search the solution space and find good candidates. A human modeler cannot effectively search a solution space manually once it grows past a certain magnitude.
3. For large models, manual construction methods, such as pointing and clicking to intricately connect hundreds or more components, are tedious and error prone.
4. Often, global constraints rely on so much information that not all of the relevant bits of

information can be seen at once. When not all of the information can be seen, modelers cannot make an informed decision.

Another difficulty of highly combinatorial domains is that although modelers may create a model that satisfies the domain constraints, the model may be considered poor in quality. For example, a modeler creating a deployment of components to ECUs could easily select a scheme that utilized far more ECUs than the true minimum number required to host the set of components. For domains, such as automotive manufacturing, each modeling decision can have significant cost consequences for the final solution. For example, if a model can be constructed that uses three fewer control units to host the car's components and consequently saves $100 in manufacturing costs, millions of dollars in overall cost reduction for all cars of this make that are manufactured can be achieved. In these cases, it is crucial to not only find a correct solution but to find a cost effective one.

The difficulty of finding a good solution is that with large models and complex global constraints, modelers are lucky to find *any* valid solution. Because finding a single solution is incredibly challenging, it becomes infeasible or cost prohibitive to produce scores of valid solutions and search for an optimal one. Even if the set of valid solutions is large, there are numerous numerical methods to search for a solution with a given percentage of optimality. These methods, however, all rely on the ability to generate large numbers of valid solutions and are not possible without automation.

In domains with large models and intricate constraints, modelers must be able to see hundreds of modeling moves into the future to satisfy a global constraint or optimize a cost. The more localized a modelers decisions are and the less distant they peer into the future, the less chance there is that a correct or good solution will be found. Good local decisions, also known as "greedy decisions," do not necessarily produce a globally good decision.

For example, consider a simple model that determines the minimum number of ECUs needed to host a set of components. Assume that there are two types of ECUs, one that costs $10 and can host 2 components and another that costs $100 and can host 42 components. If modelers are deploying using a myopic view and not peering into the future, they will select many $10 ECUs and create a solution that costs $210, rather than looking ahead and choosing two $100 controllers for a final cost of $200. Making a series of locally good decisions may not produce the overall best decision (Cormen et al., 1990).

Solution: Integrating Constraint Solvers and MDD Tools

An MDD tool provides a visual language for a developer to build a solution specification. An instance of a visual model contains modeling entities or elements, similar to OO classes, and different visual queues (e.g., connections, containment) specifying relationships between the elements. For example, a connection between a component and an ECU specifies deployment in the automotive modeling example from the Introduction section.

The key objective of a modeler is to add the right model entities and relationships between the entities so that they create a solution that meets the application requirements. Modelers express relationships between entities by drawing connections between them, placing entities within each other for containment, or other visual means. For each relationship that a modeler creates between entities, such as deployment, the modeler must find the right source and target for the relationship so that the relationship satisfies any constraints placed on it. In the example of deploying components to ECUs, the modeler must only draw

a connection from a component to an ECU that has the OS and resource capabilities to support the component.

As has been shown in the *Introduction*, *Background*, and *Measuring Domain Complexity* sections, the large size of DRE models and their complex constraints can make manually finding the right endpoints for these relationships, such as deployment, infeasible. To address the scalability challenges of manual modeling approaches presented in the aforementioned sections, this section outlines how a constraint solver can be integrated with an MDD tool to help automate the selection of endpoints for relationships between model entities.

In the context of modeling, a constraint solver is a tool that takes as input one or more model elements, a goal that the user is attempting to achieve, and a set of constraints that must be adhered to while modifying the elements to reach the goal. As output, the constraint solver produces a new set of states for the model elements that achieves the desired goal while adhering to the specified constraints. For example, a set of components can be provided to a constraint solver along with the deployment requirements (constraints) of the components. The goal can then be set to "all components connected to an ECU." The constraint solver will in turn produce a mapping of components to ECUs that satisfies the deployment constraints.

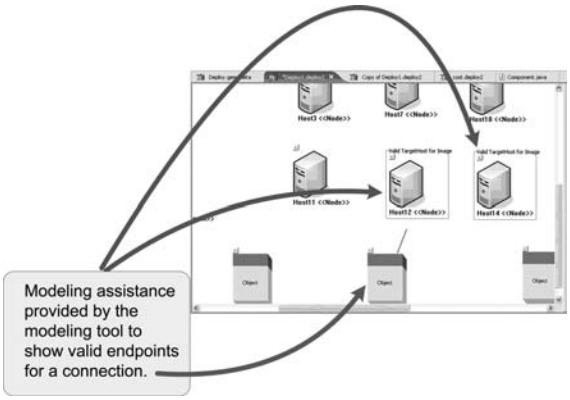
The remainder of this section first outlines the different type of modeling assistance that an MDD tool and integrated constraint solver can provide to a user. Next, the section discusses how a user's actions in an MDD tool can be translated into constraint satisfaction problems (CSPs) so that a constraint solver can be used to automatically derive the correct endpoints for the relationships the user wishes to create. Finally, the section illustrates an architecture for integrating Prolog as a constraint solver into an MDD tool.

Modeling Assistance

There are two types of constraint solver guidance that can be used to help modelers produce solutions in challenging domains: *local guidance* and *batch processes.* Local guidance is a mechanism whereby the constraint solver is given a relationship and one endpoint of the relationship and provides a list of valid model entities that could serve as the other endpoint for the relationship. One example is that a constraint solver could be provided a deployment relationship and a component and return the valid ECUs that could be attached to the other end of the connection. This type of local guidance for deploying components is shown in Figure 4.

The second type of modeling guidance is for deriving endpoints for a group of relationships so that the group as a whole satisfies a global constraint. An example of a batch process would be to connect each component to an ECU in a manner such that the no ECU hosts more components than its resources can support. A batch process takes an overall goal that the modeler is trying to achieve, such as all components connected to an ECU, and creates a series of relationships on

Figure 4. Local modeling guidance



behalf of the user to accomplish that goal. By offering both local guidance and batch processes, a MDD tool can help users to accomplish both small incremental refinements to a model and large goals covering multiple modeling steps.

Local Guidance

Local guidance helps modelers correctly complete a single modeling step. A single modeling step is defined as the creation of one relationship between two modeling elements. Local guidance can be implemented as a visual queue that shows the modeler the valid endpoints for a relationship that he or she is creating. For example, when a modeler creates a connection from a component to an ECU to specify where a component is deployed, the modeler must first click on the component modeling element to initiate the connection. When the connection is initiated, the constraint solver can be used to solve for the valid deployment locations for the component and the model elements corresponding to these deployment locations can be highlighted in the model.

Challenges 3 & 4 from the section *Key Challenges of Complex Domains* can be addressed with local guidance. By identifying the model elements that are valid target endpoints of the modeling action a user is performing, a modeling tool can use visual queues (e.g., highlighting, filtering, etc.) to show the user only the information relevant to the action. Furthermore, the modeling tool can use the list of valid targets to both help the modeler identify valid solutions (helping address challenge 1 of *Key Challenges of Complex Domains*) and to prevent the user from applying an action to an invalid target endpoint (addressing challenge 3 of *Key Challenges of Complex Domains*). With a traditional MDD approach, the correctness of a user's action is checked after completion and thus the user may have to do and undo an action multiple times before the correct target endpoint is found. By finding valid solutions before a modeler completes a modeling action, the tool can preemptively constrain (e.g., veto modeling actions) what modeling elements the action can be applied to and prevent tedious and error-prone manual solution searching.

Local guidance can not only provide suggestions of correct endpoints of a relationship but can provide rankings of the local optimality of each of the endpoints. For example, deployment locations could be ranked by the resource slack available on them so that modelers are led to choose deployment targets with sufficient free resources. This manner of local guidance provides a greedy strategy to modeling guidance. At each step, modelers are led towards a solution that provides the greatest immediate benefit to the model's correctness.

Correct solutions to modeling transactions of a single modeling step can be found using local guidance. In some cases, only considering single step transactions will not produce a solution that satisfies global constraints. For example, if modelers can add ECUs as needed to deploy components to, local guidance can produce a solution that is correct with respect to the constraints, although not necessarily optimal. If, however, ECUs cannot be added to the model and the local strategy guides the modeler to a solution where no ECU has free resources and several components are undeployed, the global constraints cannot be met.

Although a greedy strategy may not produce optimal results for certain types of CSPs, such as bin-packing, in many cases these localized strategies can provide a lower bound on the optimality of the final solution. With bin-packing, a First Fit Decreasing (FFD) (Coffman, Galambos, Martello & Vigo, 1998) packing strategy that sorts items to be placed into bins by their size and nondeterministically selects the first bin that can hold the item will guarantee that the solution never uses more than ~1.87 times as many bins as the optimal solution. Providing a lower bound on the quality of the solution that a modeler can produce can

be extremely important in some domains, such as automotive manufacturing, where you want to minimize risk or cost. Although not guaranteed, a localized strategy may in fact arrive at an optimal or nearly optimal solution. Moreover, local guidance is substantially less computationally complex than providing a global maximum and can be implemented easily with a number of the approaches discussed later in this section.

Batch Processes

Global constraints require the correct completion of numerous modeling steps and are typically not amenable to user intervention. For global strategies, therefore, ***batch processes*** guided by constraint solvers can be used to create multiple relationships to bring the model into a correct state. The key differentiator between local guidance and a batch process is that local guidance deals with modeling transactions involving a single relationship while batch processes operate on modeling transactions containing two or more relationships. The larger the number of relationships in the transaction, generally the more complicated it is to complete.

One possible batch process for the component-to-ECU deployment tool could take each component in the model and create a connection to an ECU in the model to specify a deployment location. Local guidance would produce a single deployment connection for a single component. By increasing the size of the modeling transaction to consider the deployment locations of multiple components, the batch process can use the constraint solver to guarantee that if a possible solution is found, it utilizes only the ECUs currently in the model. By expanding the transaction size that the solver operates on, the batch process allows it to make model modifications that are not locally optimal, but lead to a globally optimal or globally correct solution.

Batch processes help address challenges 1, 2, & 3 of the section *Key Challenges of Complex Domains*. First, a batch process can correctly complete large numbers of modeling actions on behalf of the user, eliminating tedious and error-prone manual modeling (addressing challenge 3). Second, a constraint solver can create both a correct and an optimal solution that can be enacted by a batch process on behalf of the modeler (addressing challenge 1). By tuning the parameters used by the constraint solver, as is discussed in the section *Transforming Non-functional Requirements into Constraint Satisfaction Problems*, the modeler can guarantee both optimality and correctness (addressing challenge 2).

Transforming Nonfunctional Requirements into Constraint Satisfaction Problems

To integrate local and batch process guidance from a constraint solver, a model and the actions that modelers can perform on the model must be transformed into a series of constraint satisfaction problems (CSPs). This transformation allows the MDD tool to translate the actions of users into queries for a constraint solver. Valid satisfactions of the CSPs correspond to correct ways of completing a modeling action, such as creating a connection.

A CSP is a set of variables and constraints over the values assigned to the variables. For example, $X < Y < 6$ is a CSP with integer variables *X* and *Y.* Solving a CSP is finding a set of values (a labeling) for the variables such that the constraints hold true. The labeling X = 3, Y=4, is a correct labeling of X < Y < 6. A constraint solver takes a CSP as input and produces a labeling (if one exists) of the variables. Solvers may also produce labelings that attempt to maximize or minimize variables. For example, X = 4, Y =5, is a labeling that maximizes the value of X.

For the deployment example, a deployment of a set of components to a set of ECUs can be viewed as a binary matrix where the cell at row *i* and column *j* is 1 if and only if the *i*th component

is deployed to the *j*th ECU (and 0 otherwise). Each cell can be represented as an independent variable in a CSP. Thus, each variable *Dij* determines if the ith component is deployed to the jth ECU. Finding a correct labeling of the values for the *D* variables creates a deployment matrix that can be used to determine where components should be placed.

Assume that the *ABS* (antilock braking system) component and the *WheelRPMs* components must be deployed to the same ECU. Also assume that the ABS component must be placed on an ECU at least 3 feet from the perimeter of the car. This series of deployment constraints can be translated in a CSP model. Let the ABS component be the 0^{th} component and the WheelRPMs component be the 1^{st} component. First, the constraint that the ABS component be deployed to the same ECU as the WheelRPMs component is encoded as $(D_{0j} = 1) \rightarrow (D_{1j} = 1)$. Next, for each ECU, a constant $Dist_j$ can be created to store the distance of the jth ECU from the perimeter of the car. Using these constants, the constraint on the placement of the ABS component relative to the perimeter of the car can be encoded as $(D_{0j} = 1) \rightarrow (Dist_j \geq 3)$. If this CSP is input into a constraint solver, the solver will label the variables and produce a deployment matrix that is guaranteed to be correct with respect to the deployment constraints.

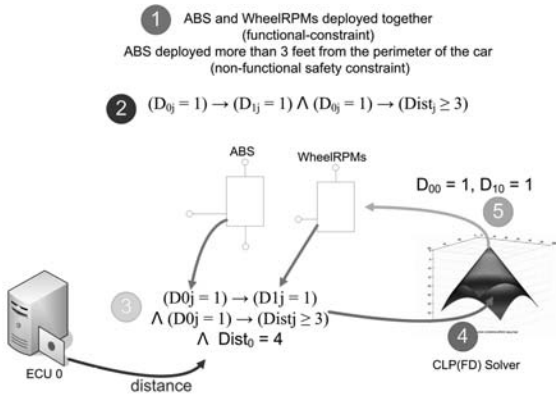
A constraint solver can also be used to derive a solution with a certain degree of optimality. Assume that *N* components need to be deployed to one or more of *M* ECUs using as few ECUs as possible. A new variable *UsedECUs* can be introduced to store the total number of ECUs used by a solution. The constraint $UsedECUs = \sum D_{ij}$ for all i from 0..N and all j from 0..M. The solver can then be asked to produce a labeling of the variables D_{ij} that minimizes the variable UsedECUs. The solver will in turn produce a valid deployment of the components to ECUs that also minimizes the total number of ECUs used.

Constraint solvers typically offer a number of solution optimization options. The options range from maximizing or minimizing a function to using a fast approximation algorithm that guarantees a specific worst-case percentage of optimality. Depending on the constraint solver settings used, a modeler can guarantee the optimality of a model or trade a certain percentage of model optimality for significantly reduced solving time. In contrast, a manual modeling approach provides no way to guarantee correctness, optimality, a percentage of optimality, or a tradeoff between optimality and solution time. For software intensive systems where optimality is important, allowing modelers to tune these parameters is a key advantage of using a constraint solver-integrated modeling approach.

One goal of using a constraint solver is to produce better solutions than a human modeler can create manually and to produce good solutions more reliably. When a solver uses either optimal or approximation algorithms, the solver's solution has a known and guaranteed worst case solution quality. In contrast, there is no guarantee on the solution quality with a manual approach.

As shown in Figure 5, the nonfunctional requirements for the software system must first be

Figure 5. Transforming a model into a constraint satisfaction problem



collected and documented (step 1). Each nonfunctional requirement must then be translated into a CSP, such as a system of linear equations (step 2). At this point, the data from the model, such as ECU distances to the car perimeter, are collected and bound to variables in the CSP produced in the previous step (step 3). Next, the CSP with some bound variables (such as resource demands) and some unbound variables (such as the D_{ij} variables in Figure 5) are input into the constraint solver (step 4). The constraint solver then produces bindings for the unbound variables and maps them back to changes in the model (step 5).

A crucial element for creating the right translation from nonfunctional requirements to a set of CSPs is the abstraction used to decompose the model into the variables and facts (i.e., bound variables) that the CSPs operate on. For example, should ECU and component be present in the formulation of the CSP to represent the bin-packing of the model's resources? The metamodel of a language, as described in *Background* section, provides the terminology and syntactic rules for a modeling language. Because the metamodel contains a precise definition of the relevant types in a modeling language it is ideal for identifying the key concepts that the CSPs should use. The metamodel of a modeling language can be viewed as a set of model entities and the role-based relationships between them. By using this abstraction based on entities and role-based relationships, a model can be conveniently decomposed for processing by a constraint solver. The idea of relationships between elements is the same as the widely used Resource Description Framework's predicate/argument format.

The role-based relationships of an entity represent both its properties (such as available CPU) and its associations (such as hosted components). Each entity can be decomposed into a unique ID and a set of role-based relationships associated with the ID. A requirement, such as "a component is only deployed to an ECU with the correct OS" can be translated into a CSP involving the *Deployment*, and *OS* relationships of a component and ECU. The variables of the CSP for this requirement would be the component and ECU that are being associated through the *Deployment* relationship. The constraint would be that the *OS* relationship of the component and the ECU had the same value (i.e., the same OS).

Associating Modeling Actions with the Constraint Solver

An important integration question is how/when to invoke the constraint solver and what CSPs and variable bindings should be passed to it. The goal is to use the constraint solver to provide local guidance and batch processes to bind the endpoints of relationships in the model. A constraint solver requires a CSP, a set of unbound variables (e.g., unbound endpoints), and a set of bound variables to produce a list of endpoints for relationships. Thus, users' actions and model state must be interpreted to find the correct CSPs, model entities, and unbound endpoints to pass to the solver. By defining the right formal model of the process by which users' actions are interpreted and translated into input data for the constraint solver, the integration process can be more cleanly defined. This section presents a formal abstraction for a user's interaction with a modeling tool and shows the point in the formal specification at which the constraint solver can be integrated and used to automate relationship endpoint binding decisions.

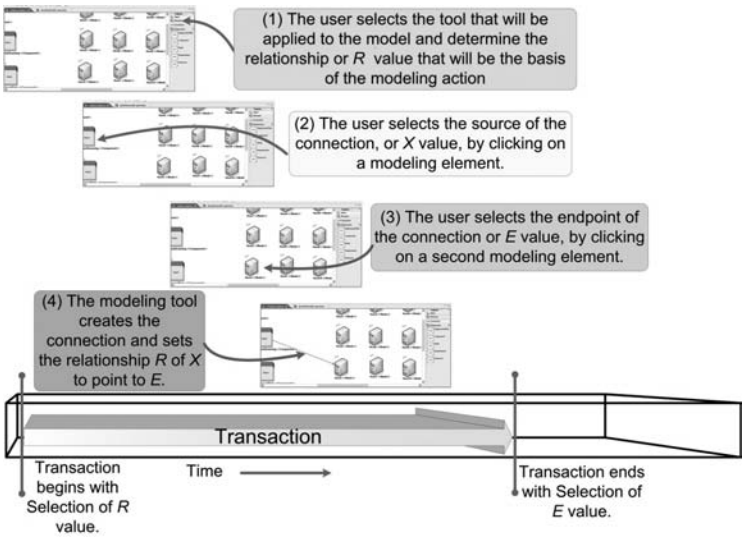
Modeling actions are transactions that take one or more elements of the model and modify the endpoints of the selected elements' role-based relationships. Creating a deployment connection takes a component (the source of the connection) and sets the endpoint of its *TargetECU* relationship. In the *Local Guidance* and *Batch Processes* sections, a modeling action was defined as a transaction by the user that takes a relationship and sets its source and target entities. More formally, a modeling action is a function, *action(X,*

R, E), that takes a model element *X*, a relationship of the element, *R*, and produces an endpoint for that relationship *E,* as shown in Figure 6.

The goal of a traditional MDD tool is to take the input produced by the user, such as mouse clicks, and translate them into the values for *X*, *R*, and *E* to update the model. With a traditional MDD tool, the values for *E* are explicitly bound by modelers. A MDD tool integrated with a constraint solver not only provides this traditional explicit binding capability but also provides a constraint solver binding process, in which the constraint solver deduces the proper endpoints for relationships on behalf of the modeler.

The GEF and EMF frameworks can be used to illustrate how X, R, and E are actually implemented in a modeling framework. GEF provides an MVC framework for displaying and editing EMF models. In GEF, each possible user action, such as connecting two elements with a line in the graphical model, is represented with a *Command* object. The command object is a part of the *Command Pattern* (Gamma et al., 1995), which encapsulates actions that can affect a model in an object. When the user clicks on an element and then presses the delete key, GEF constructs a *DeleteCommand*, sets the command's argument to be the element that was click on, and then calls the command's *execute()* method, which deletes the element from the EMF model. When the user wishes to create a connection, the user selects the connection tool from a tool palette. Selecting the connection tool causes GEF to construct a *ConnectionCommand*. When the user clicks on the first element for the connection, GEF passes the element to the ConnectionCommand as the source argument. When the user clicks on the endpoint for the connection, GEF passes the command the endpoint as the target argument and calls the command's execute() method, which creates the connection between the two elements. Tool implementers create Command objects to specify how each possible user action is translated into changes of the underlying EMF model.

Figure 6. Diagram of a modeling transaction

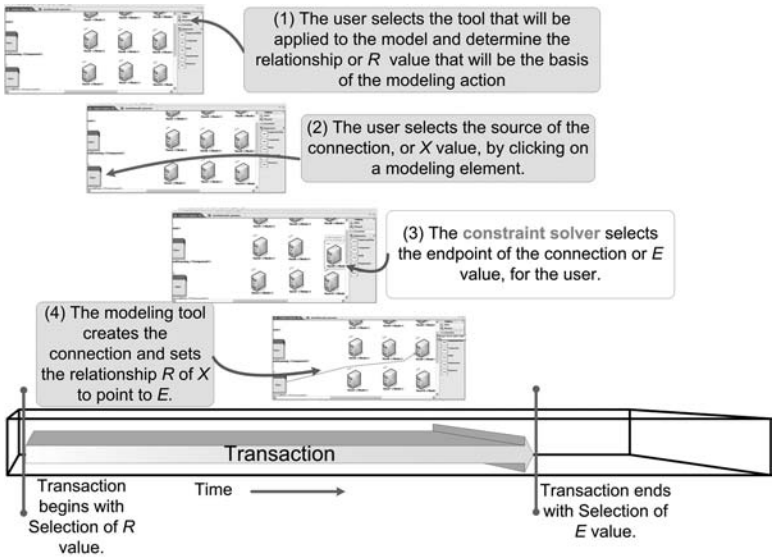


With GEF's command pattern, R is determined by the type of Command object that GEF instantiates. In the deployment example, when the user selects the *DeploymentConnection* tool, GEF creates a corresponding *DeploymentConnectionCommand* object. The Command knows (because it is coded into the command object's execute method) that it is modifying the *TargetECU* relationship of its source argument. The command also knows that its source argument is the X variable in the action(X,R,E) function. Finally, the command knows that its target endpoint represents the E variable. Each Command object is used to translate a graphical user action (e.g., adding a connection) into values for X, R, and E. The Command is also responsible for modifying the R relationship between X and E in its execute method. The execute() method of a DeploymentConnectionCommand is shown in the Java code below:

```
public class DeploymentConnectionCommand
extends Command{
 ....
 //apply action(X,R,E)
 public void execute() {
  Component source = (Component)this.get-
Source(); //the X
 ECU target = (ECU)this.getTarget(); //the E

 //the R relationship (targetECU) between X and
E is set here
 source.setTargetECU(target);
 }
}
```

In the modified binding process for *E*, each relationship *R* is associated with a CSP specifying what is considered a correct value for *E*. For example, a component could specify that a correct value for its *TargetECU's E* value requires that the chosen *E* value and the component both have the same OS type. When a user input is translated

Figure 7. A diagram of a modeling transaction with a constraint solver



into values for *X* and *R*, a constraint solver integrated MDD tool uses the CSP associated with *R* to automatically derive values for *E* on behalf of the user. In Figure 5, the CSP was found in step 2, the values for *X* and *R* were produced in step 4 and the bindings for *E* were delivered by the constraint solver in step 5. The modified modeling transaction process can be seen in Figure 7.

In the first step, the user selects a tool or action that will be applied to the model. The tool determines the *R* value or relationship that will be modified by the user's actions. In the second step, the user clicks on a modeling element to initiate a connection and hence modify a relationship in the underlying model. The element that the user clicks on becomes the *X* value that will be passed to the constraint solver. In the third step, the modeling environment looks up the correct CSP that must be satisfied by the endpoints of the relationship specified by the *R* value. The modeling environment then passes this CSP, the *X*, and *R* values to the solver. The solver finds the endpoints that satisfy the CSP and returns these endpoints as possible *E* values. Finally, the *E* values are presented graphically to the user.

The GEF *DeploymentConnectionCommand* can be modified to incorporate this new process by which the constraint solver chooses the value for *E*. The Command creation and initial argument setting remains unchanged. However, after the source of the connection has been set, the constraint solver can be invoked to solve for a value for E. If a value is returned, the execute() method can be called immediately. The new *DeploymentConnectionCommand* is:

```
public class DeploymentConnectionCommand
extends Command{
 ....
 public void setSource(Object obj) {
  this.source = obj;

  //the X
  Component source = (Component)obj;

  //call the solver to find valid values for E
 List endpoints = this.solver.findEndpoints(source.
getId(),
                    "targetECU");

  //if there is only one possible value, go ahead
and execute
  if(endpoints.size() == 1){
    setTarget(endpoints.get(0));
    execute();
  }
  else if(endpoints.size() > 0) {
    //otherwise, show the user valid E values by
    //modifying their background color
    for(Object obj : endpoints)
        ((ECU)obj).setBackgroundColor(Color.yel-
low);
  }
else {
 //notify the user that there are no
  //possible deployment locations for the Com-
ponent
 source.setBackgroundColor(Color.red);
}
 }

 //apply action(X,R,E)
 public void execute() {
  Component source = (Component)this.get-
Source(); //the X
 ECU target = (ECU)this.getTarget(); //the E

 //the R relationship (targetECU) between X and
E is set here
 source.setTargetECU(target);
 }
}
```

In the modified DeploymentConnectionCommand, immediately after GEF sets the source of the connection, the command invokes the constraint solver to find valid endpoints. If exactly one endpoint is found, the setTarget method is called with that endpoint and the Command is executed. If more than one valid endpoint is found, each valid target has its background color changed to yellow (a visual queue). If there is no possible deployment location for the Component, its background color is changed to red.

In a traditional process, the user would be required to click first on the source element, decide on a valid deployment location for the source, and then click on the deployment location. With the modified Command object, the object itself attempts to determine the valid targets (E) using the constraint solver. The Command can then automatically complete the action on the user's behalf, if there is exactly one possible endpoint. If there is more than one possible endpoint, the Command can highlight those endpoints for the user. If no endpoints are found, the Command can notify the user by changing the Component's background color to red.

In many situations, the user will wish to find a valid endpoint for a specified R relationship for every member of a set of modeling elements. For example, the user may wish to select some or all of the Components and have the solver find a valid target ECU for every Component such that no global deployment constraint, such as resource consumption, is violated. Using the GEF framework, a new *BatchDeploymentCommand* can be created.

Just as with other GEF commands, the BatchDeploymentCommand can have a tool palette entry associated with it that the user can select. When the user selects the corresponding tool entry, the BatchDeploymentCommand is created. The batch command takes a group of modeling elements, which the user specifies through a group selection, and creates a connection for each member of the group to a valid ECU. The Java code for the BatchDeploymentCommand is:

```
public class BatchDeploymentCommand extends
Command{
 ....

 public void execute() {
 //the set of Xs
  Component[] sources = (Component[])this.
getSources();
 //the solver deduces an E for each X
  Object[] targets = this.solver.findValidTargets(
sources,
                    "targetECU");

 if(targets != null){
  for(int i = 0; i < targets.length; i++) {
   sources[i].setTargetECU((ECU)targets[i]);
  }
 }
 }
}
```

Constraint Solver and MDD Tool Integration Frameworks

There are a large number of optimization, constraint solver, and inference engines available for use with local guidance and batch processes. As noted in Van Hentenryck and Saraswat (1996), however, automating the formulation of real problems in a suitable form for efficient algorithmic processing is hard. Transforming an arbitrary graphical model, a modeling action, and a set of modeling constraints into a CSP for a constraint solver is tedious and error-prone. Integrating the results of the solver back into a MDD tool and providing interactive capabilities is also hard. Each of the five steps from the section *Transforming Non-functional Requirements into Constraint Satisfaction Problems* may require substantial effort. By choosing the right approach and architecture, however, the difficulty of leveraging a constraint solver in a modeling environment can be reduced substantially.

The following are five important properties of an architecture for integrating a constraint solver with a MDD tool:

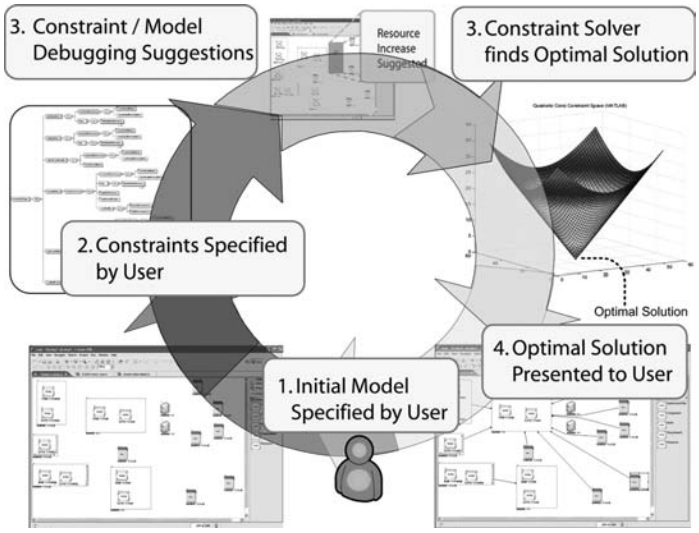
1. **Solver frameworks must respect domain-specific concepts from the MDD tool** and provide a flexible mechanism for translating nonfunctional requirements into CSPs using domain notations. MDD tool users should

be able to specify constraints in a language or notation that mirrors the domain rather than a system of linear equations and makes mapping requirements to a CSP easier.

2. **The local guidance and batch processes should lead modelers toward solutions that are considered optimal** or good based on quality metrics from the domain. Whenever possible, solvers should be used to iterate through multiple valid solutions and suggest only those considered most optimal. Modelers should be able to plug-in custom formulas for measuring optimality in the target domain and the tool should be able to present multiple suggestions based on various types of optimization.
3. **The constraint solver integration should automate tedious and complex modeling tasks**, such as solving for and assigning values for global constraints, performing repetitive localized decisions, or providing feedback to modelers to suggest valid modeling decisions.
4. **The solver framework must accommodate long-running analyses** for problem instances that cannot be solved online. For large optimization problems, such as finding the lowest cost assignment of components to ECUs, the constraint solver may need several hours or days to find a solution. In cost-critical situations, such as manufacturing, allowing the solver the extra time to find the best solution can be critical.

With a constraint-solver integrated modeling environment, a user goes through an iterative process of specifying portions of a model, adding or refining nonfunctional requirements as constraints, and using the constraint solver to automate model construction and optimization. Figure 8 illustrates the modeling processing with an integrated constraints solver.

Figure 8. A modeling cycle with constraint solver integration



In the first step, a user specifies the initial model entities in the solution. In the second step, the user adds constraints for the requirements of the solution into the MDD tool. During the third phase, the user invokes the constraint solver, using local-guidance or a batch process, to find valid endpoints for various relationships in the model. Finally, in the fourth step, the valid endpoints found by the constraint solver are shown to the modeler using visual queues, such as highlighting valid entities.

Figure 9. Execution of a ROCs batch process shown in GEMS

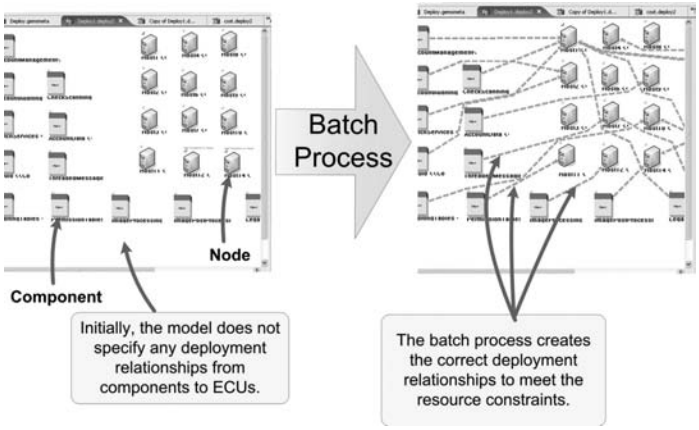
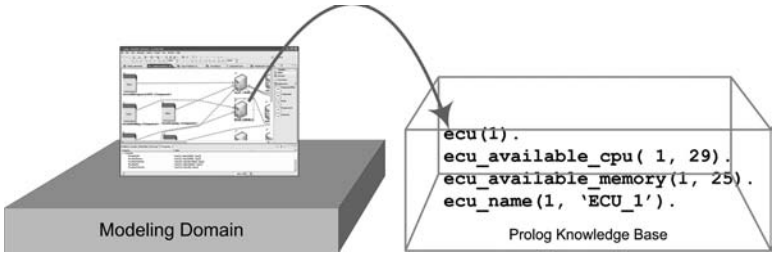


Figure 10. Transforming model elements into prolog facts



A Prolog-Based Approach to Constraint Solver Integration

Choosing a constraint solver is one of the driving forces in the process of transforming a set of nonfunctional requirements into a CSP. Each solver will generally have a unique representation of the problem in its native format. The choice of solver therefore affects how the transformation from nonfunctional requirements to a concrete representation of a CSP is performed. Many types of solvers are available and implemented in a number of languages. The remainder of this section presents an approach we have developed, called Role-based Object Constraints (White, Nechypuren, Wuchner, & Schmidt, 2006), to providing local guidance and batch processes based on Prolog (Bratko, 1986).

Using ROCs, we have implemented constraint-solver integrated modeling tools for automated product line variant selection (White, 2007a), component to ECU deployment in automobiles (White et al., 2006), and aspect weaving (Nechypurenko, Wuchner, White & Schmidt, 2007). Our implementation of ROCs is integrated with the Generic Eclipse Modeling System (GEMS) (White, 2005), a part of the Eclipse Generative Modeling Tools (GMT) project. A screenshot of a batch process executing in our GEMS- based deployment modeling tool is shown in Figure 10.

Prolog is a declarative programming language that allows programmers to define a Knowledge Base (KB) (also known as a fact set) and a group of rules that implement a set of CSPs (see Sidebar 1). Prolog can then evaluate these rules and determine if they can be satisfied by the known facts. Prolog uses a predicate syntax, where rules can be defined as predicates that resolve to the satisfaction of a conjunction of other predicates. Rules are akin to methods that check if a constraint over a set of variables holds true.

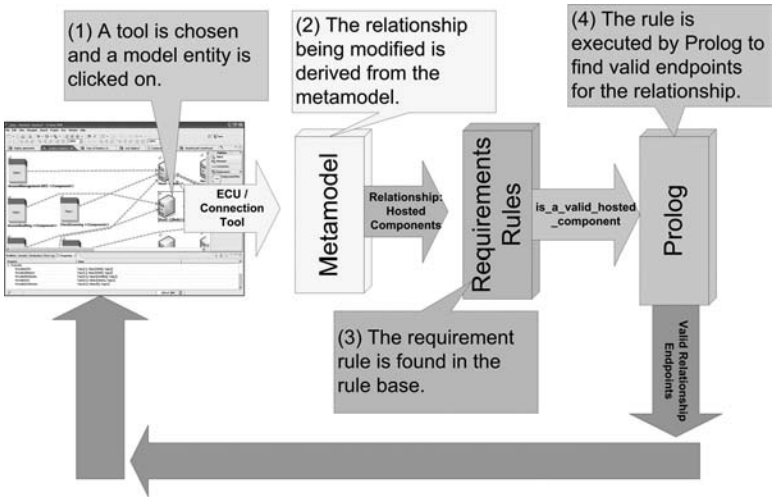
Predicate rules can be used to check constraints, by invoking the rule with all variables bound, in which case Prolog replies with whether or not the rule or CSP evaluates to true. If variables are left unbound when the rule is invoked, however, Prolog uses backward chaining to produce bindings from the KB of the unbound variables that will satisfy the CSP. Prolog therefore provides a key degree of flexibility because it can be used both to check constraints (similar to OCL described in the section *Constraint Checking with OCL*) or to derive solutions to the CSPs.

The remainder of this section presents an approach to integrating Prolog into a modeling tool. Prolog was chosen because it has a readable textual syntax as opposed to the linear-equation-based syntax of other possible solvers. Using Prolog, however, does trade some speed for readability and ease of use. Prolog also is a widely used and supported programming language for constraint solving and numerous existing solvers and libraries are available in Prolog.

Sidebar 1. Prolog

Prolog is a logic programming language that allows developers to specify a set of facts or Knowledge Base and then create rules specifying logical assertions or constraints on the facts. Prolog rules take one or more input variables, denoted by variable names with capital letters, and specify a series of logical assertions on these variables, other facts, or rules in the KB. When a Prolog rule is invoked with only bound variables, meaning all variables have values assigned to them, Prolog returns whether or not the logical assertions contained within the rule hold true. An important capability of Prolog is that if a rule is invoked with some unbound variables, Prolog will attempt to find bindings of those variables from the facts in the KB that satisfy the logical assertions in the rule. When constraints are implemented as Prolog rules, Prolog can deduce valid bindings for the variables that the constraints restrict.

Figure 11. Invoking requirement rules to find relationship endpoints



Transforming Models into Prolog Knowledge Bases

Integrating Prolog as a constraint solving engine involves capturing the state of the model and translating it into a Prolog KB, as seen in Figure 11. For the deployment of components to ECUs example from the *Introduction* section, the components, ECUs, and their resources must be translated into predicate facts in Prolog. Generally, predicates are created that relate a unique key or ID of each model element to various properties that the model element possesses. This concept is similar to the use of pointers and allows the flattening of an object-oriented model into a predicate KB.

Developers must select the format of the predicates used to translate the model into a Prolog KB. One approach is to use a consistent set of Prolog predicates across modeling languages and customize them by adding domain-specific information into the variables the predicates operate on. For example:

```
self_type(1, ecu).
self_attribute(1, available_cpu, 29).
self_attribute(1, available_memory, 25).
self_attribute(1, name, 'ECU_1').
```

describes a set of Prolog facts that provide a general predicate format applicable to a range of model types. This set of facts asserts that the element with ID "1" is of type "ecu." The facts also assert that the element has three attributes: *available_cpu*, *available_memory*, and *name*, with values: "29," "25," and "ECU_1," respectively. Different modeling languages can be accommodated by changing the second argument of the predicates, the attribute name, which is being defined. The tradeoff of using a general format, however, is it violates the first design criterion described in the section *Constraint Solver and MDD Tool Integration Frameworks*, that is, offering a domain-specific interface. The predicates do not vary across domains, which makes it harder for a domain expert to understand how they relate the concepts from his or her domain.

Rather than using terminology specific to the deployment of components to ECUs (e.g., ecu, available_memory, etc.), the predicates are based on describing the attributes and types.

A more domain-specific approach is to create custom predicates for each modeling language to mirror the notation from the domain. For example, the same set of facts can be rewritten as:

```
ecu(1).
ecu_available_cpu( 1, 29).
ecu_available_memory(1, 25).
ecu_name(1, 'ECU_1').
```

which provides a more domain-specific interface. The main drawback of this format, however, is that introspection is not possible, that is, rules cannot query for all of the properties of an arbitrary element. When translating nonfunctional requirements into Prolog rules, domain-specific predicates are generally more advantageous because they allow the production of more compact and readable rules. Introspection is also typically not needed for writing CSPs in Prolog.

To identify the domain-specific predicates to use for the KB, the metamodel for a modeling language can be viewed as a set of model entities and the role-based relationships between them. For each entity, a unique id and a predicate statement specifying the type associated with the entity. For example, each ECU in the model is transformed into the predicate statement ecu(id), where id is the unique id for the ECU. For each instance of a role-based relationship in the model, a predicate statement is generated that takes the id of the first participating entity and the id of the entity to which the first entity is being related.

For example, if a component, with id 23, has a *TargetECU* relationship with an ECU, with id 25, the predicate statement targetECU(23,25) is generated. This predicate statement specifies that the entity with id 25 is a *TargetECU* of the entity with id 23. Each KB provides a domain-specific set of predicate statements. As a model is manipulated in its graphical editor, the Prolog KB is updated using assert/1 and retract/1 statements, which add and remove facts from the Prolog KB, respectively.

Mapping Nonfunctional Requirements to Prolog Rules

Using a domain-specific knowledge base, modelers can specify nonfunctional requirements in the form of Prolog rules for each type of metamodel relationship. These constraints semantically enrich the model to indicate the nonfunctional requirements of a correct model. They are used by constraint solvers to deduce solutions to local guidance and batch process problems. For example, consider the following constraint to check whether an ECU is a valid ECU of a component:

```
is_a_valid_component_targetECU(Component,
ECU) :-
  component_requiredOS(Component, OS),
  ecu_providedOS(ECU, OS).
```

This constraint, which checks to ensure that the OS required by the component matches the OS provided by the ECU, can be used to check a component-ECU combination, that is:

```
is_a_valid_component_targetECU(component_
23, ecu_25).
```

by assigning the *Component* variable the value "component_23" and the *ECU* variable the value "ecu_25." The rule can also be used to find valid ECUs that can play the *TargetECU* role for a particular component using Prolog's ability to deduce the correct solution

```
is_a_valid_component_targetECU(component_
23, ECU)
```

by leaving the *ECU* variable unbound (unbound variables are begin with capital letters). In this example, the *ECU* variable will be bound to the

ID's of the ECUs in the KB that have the same OS as the component. This example shows how the nonfunctional requirement rules can be used both to check and to deduce solutions.

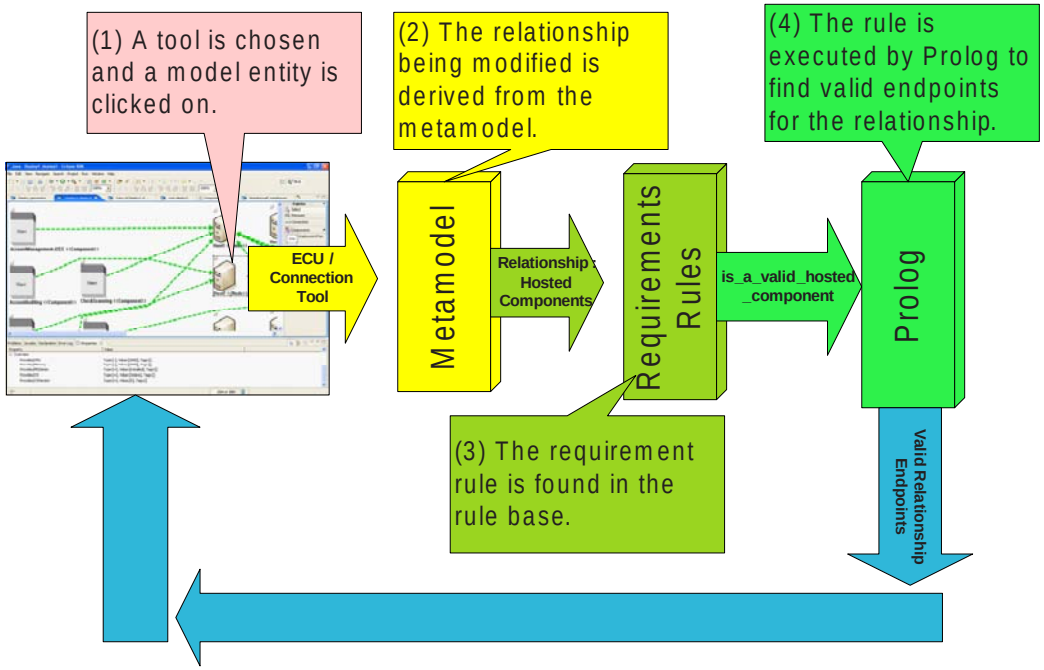
The role-based relationships present in the metamodel not only produce domain-specific predicates but also serve as the glue between graphical modeling actions, such as creating connections, and the constraint solver. The nonfunctional requirement rules that developers create can be associated with role-based relationships in the metamodel as seen in Figure 12.

When a model element is clicked on to initiate a change (step 1), the metamodel is consulted to determine the role-based relationship (step 2) affected by the change. The corresponding nonfunctional requirement rules can then be obtained (step 3) and executed by Prolog (step 4) to check the validity of the change.

To provide local guidance, the nonfunctional requirement rules associated with metamodel relationships can be executed with unbound variables to deduce endpoints for the relationship. For example, if a user begins creating a deployment connection originating from a component, the MDD tool can deduce that the deployment connection will set the *TargetECU* relationship of the component and execute the is_a_valid_component_targetECU rule with only the originating component bound as an argument. Prolog will then return bindings for the ECUs variable of the rule that are valid deployment targets for the component.

In a graphical modeling environment that does not include a constraint solver, graphical actions, such as mouse clicking and movement, are translated into changes in the underlying ob-

Figure 12. Invoking requirement rules to find relationship endpoints



ject graph of the model. By adding a constraint solver, graphical actions are translated into proposed model modification transactions and then the proposed transactions are turned into CSPs and solved. Users can therefore modify the model directly as in a traditional approach and use graphical actions to initiate constraint solvers that modify the model on their behalf.

In the connection creation example from the *Local Guidance* section, a modeler's mouse-click on the source component is translated into the initiation of a connection from the component. This connection initiation proposal is then used to query the metamodel to determine the relationship that is being modified on the component. Next, the CSPs or nonfunctional requirement rules that are bound to the relationship are obtained and finally they are executed in the constraint solver to find valid bindings for the unbound variables. The bindings produced represent the valid completions of the modeling transaction. These valid bindings can be returned to the user as graphical proposals, such as highlighting model elements, or committed as changes to the underlying model.

CONCLUSION

For large-scale DRE systems, traditional modeling approaches allow developers to raise the level of abstraction used for solution specification and illuminate design flaws earlier in the development cycle. Many DRE systems, however, have exponential design constraints, extremely large model sizes, or other complexities that make it hard to handcraft a model of a solution. For these types of challenging domains, automated design guidance based on the design constraints is needed.

A constraint solver can be integrated into a modeling environment to provide design guidance for complex domains. As shown in section *Modeling Guidance*, using the concepts of *local guidance* and *batch processes* a constraint solver can help modelers perform both single and multistep modeling activities.

The lessons learned from our ROCs approach, described in the section *Modeling Guidance*, to integrating a Prolog constraint solver with the GEMs modeling environment are:

- **Constraints can be used for reasoning by a constraint solver**. A constraint solver improves solution quality not by checking manually produced solutions, but by actively guiding a user toward a correct solution. The solver helps ensure that users do not produce an incorrect solution, rather than just notifying them if their solutions are invalid.
- **User actions can be abstractly modeled as functions**. User interactions with MDD tools can be viewed as a function that takes a set of modeling elements and maps the endpoints of a specific relationship of the elements to an endpoint explicitly provided by users. A constraint solver can be integrated into a modeling environment by dynamically choosing the endpoints for the relationships with the constraint solver rather than requiring endpoints to be explicitly enumerated by the modeler. Local guidance and batch processes can be used to produce the endpoints for relationships, as described in the section *Local Guidance*.
- **Constraint solvers should be reused**. Writing a constraint solver is hard. Developers of model-driven processes should therefore focus on integrating existing constraint solvers or constraint solving languages. Prolog is a good choice for a general purpose constraint solving language that can be integrated, as shown in the section *A Prolog-based Approach to Constraint Solver Integration*.
- **Debugging constraint conflicts or over constrained systems is hard.** When no valid solution to a CSP can be found, deriving why a solution can't be found can be complex. With global constraints, the cause of the failure can be the overall organization of the

solution, and thus it is difficult to provide a meaningful explanation to the modeler.

- **Constraint solvers typically perform well in practice.** Although many optimization and constraint satisfaction problems are combinatorialy complex, constraint solvers typically can solve them in a reasonable time frame. Constraint solvers can use approximation algorithms to quickly produce solutions that are good but not optimal.
- **The complexity of a constraint satisfaction problem is dependent on each problem instance.** Certain instances of a type of constraint satisfaction problem will be easier to solve than others. Predicting which instances are challenging is hard. Although it is hard to predict which instances are challenging, constraint solvers often work well on all instances in domains that humans manually produce solutions for.

As MDD tools continue to develop and capture more useful design decisions for larger and more complex applications, constraint solving and other design automation techniques will become more important. Design automation should not only improve design quality but should also help to allow model-driven processes to scale to handle next-generation models with significantly more complexity.

The tools and code presented in this chapter are a part of the Generic Eclipse Modeling System (GEMS). GEMS is an opensource project available from http://www.sf.net/projects/gems.

FUTURE RESEARCH DIRECTIONS

This section describes the emerging trends in the development of software intensive systems, how these trends will affect software development methodologies, and what future research will be needed to address future development problems. The future trends are presented in the context of MDD. Particular emphasis is placed on how these trends will impact the use of constraint solvers in software development.

Capturing Design Rules

Model-driven technologies are raising the level of abstraction for software development by enabling developers to express higher-level, more domain-specific intentions in the solution specifications they produce. These intentions have traditionally been captured through documentation, such as text files or MS Word documents. With MDD technologies, developers can formally specify the design goals and rules that traditionally could only be expressed in documentation, in the solution specification.

In the past, conventional tools could not document the rules that led a developer to make design decisions in a rigorous manner that could be used for automated design assistance. For example, implementation-based software development methodologies, such as coding a solution in C++, could not capture communication rate information, memory consumption, minimum distance from the car's perimeter, or other constraint information in a form that could drive application organization. With an MDD approach, however, a designer can specify that a connection needs to provide a guaranteed messaging rate rather than the type of connection that should be used, that is, designers can specify *why* one connection type should be preferred over another connection type.

Various MDD tools are developing that can utilize this design information. The Component Utilization and Test Suite (CUTS) (Hill & Gokhale, 2007) is an MDD tool that allows developers to empirically evaluate system designs before they are implemented. Other tools, such as J2EEML (White, 2007b), provide the ability to perform analysis on adaptive applications and anticipate conflicting design goals. Finally, feature

modeling tools (Antkiewicz et al., 2007) provide the capability to capture software component commonality and variability requirements and enforce them during system design.

Utilizing Design Information to Provide Automated Design Assistance

Automation can be applied to help guide designers to better solutions by formally capturing the goals of the application or *why* designers chose particular design decisions. MDD tools, and specifically domain-specific modeling languages, have allowed developers to tailor the solution specification to include information pertinent to their domain. As MDD technologies increase the breadth of information that can be distilled from designers into a solution specification, a wealth of new design guidelines or constraints will become available for constraint solving.

An emerging area of research therefore involves the integration of automated reasoning systems with MDD tools. In particular, the reuse of existing constraint solver, decision assistance, and other guidance mechanisms across will be an important software development goal. These solver and decision assistance mechanisms are costly to produce and thus will benefit from greater portability between MDD tool infrastructures, such as the Eclipse Modeling Framework, Microsoft DSL tools (Microsoft, 2007), and the Generic Modeling Environment.

One challenge of leveraging constraint solvers in a modeling environment is mapping the domain requirements to CSPs that can be used for automated solving. These mappings are often complex and tightly coupled to individual solver and metamodel formats, despite the fact that requirement types, such as resource requirements, occur across multiple domains. Current solutions for transforming requirements into CSPs tightly couple the translation to a specific solver or metamodel and require costly reinvention and rediscovery of existing mappings. Additional work is needed ensure that the complex mappings from requirements to CSPs can be templatized and reused across applications.

Model transformation techniques are developing rapidly and may provide a mechanism for future decoupling of CSPs from solvers and provide portability through translation. The Atlas Transformation Language (ATL) (Jouault & Kurtev, 2005) provides powerful transformation capabilities as well as compilation to bytecode. Other emerging approaches include Open Architecture Ware (oAW) (Open Architecture Ware, 2007). Finally, templatization approaches are also viable, such as those proposed by Willans, Sammut, Maskeri, and Evans (2002).

Constraint Solver Guided Software Reuse

Significant advances in the area of software reuse will drive the need for integrating constraint solvers and MDD tools. These advances are evident in the increased use of commercial-off-the-shelf (COTS) components (Schmidt, 2002) rather than customized proprietary solutions. In particular, using COTS components in the DRE systems domain requires constraint solvers because applications in this domain often have exponential constraints that must be met by selecting and assembling COTS components together with proprietary components into a final composite application.

The selection of components is an example of a CSP. Developers must take the requirements of an application in a DRE system, the capabilities of the COTS and proprietary components, and find a set of compatible components, possibly from numerous vendors, that will satisfy both the functional and complex nonfunctional constraints. Moreover, components may have complex

configuration needs, such as setting messaging policies, to enable them to function properly. Solving these challenging CSPs will require the use of constraint solvers to select components based on high-level design criteria captured in models.

COTS components will require further development and standardization in how metadata, such as messaging periodicity, is captured and disseminated to tools. Standardization will allow for greater interoperability between tools. A standard metadata format will also provide component developers with a consistent methodology for documenting component requirements, dependencies, and capabilities.

Services have seen the most standardization in metadata descriptions. The Resource Description Framework (RDF) (Lassila & Swick, 1999) and the Web Services Description Language (WSDL) are emerging as promising standards for describing services. Other approaches, such as those presented by O'Sullivan, Edmond, and Ter Hofstede (2002), focus on capturing the non-functional aspects of services. Formal methods for describing components are also emerging, such as those proposed by Poizat, Royer, and Salaun (2004).

REFERENCES

Antkiewicz, M., & Czarnecki, K. (2006, October). Framework-specific modeling languages with round-trip engineering. In *Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, Genoa, Italy.

Atkinson, C., & Kuhne, T. (2003). Model-driven development: A Metamodeling Foundation. *IEEE Software*, 20(5), 36-41.

Bratko, I. (1986). *Prolog programming for artificial intelligence*. Reading, MA: Addison-Wesley.

Coffman, E., Galambos, G., Martello, S., & Vigo, D. (1998). Bin packing approximation algorithms: Combinatorial analysis. *Handbook of combinatorial optimization*. Norwell, MA: Kluwer Academic.

Cormen, T.H., Rivest, R.L. Leiserson, C.E., & Stein, C. (1990). *Introduction to algorithms*. Cambridge, MA: MIT Press.

Fagan, M. (1999). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, *38*(2/3), 258-287.

Forman, G., & Zahorjan, J. (1994). The challenges of mobile computing. *IEEE Computer*, *27*(4), 38-47.

Fowler, M., & Scott, K. (2000). *UML distilled*. Reading, MA: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Ginsberg, M. (1989). A circumscriptive theorem prover. *Artificial Intelligence, 32*(2), 209-230.

Graphical Modeling Framework. (2007). Retreived March 23, 2008, from http://www.eclipse.org/gmf

Hill, J. H., & Gokhale, A. (2007, to appear). Model-driven engineering for development-time QoS validation of component-based software systems. In *Proceeding of International Conference on Engineering of Component Based Systems*, Tuscon, AZ.

Jaaksi, A. (2002). Developing mobile browsers in a product line. *IEEE Software, 19*(4), 73-80.

Jouault, F., & Kurtev, I. (2005, October). Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS*, Montego Bay, Jamaica.

Kent, S. (2002, May). Model driven engineering. In

Proceedings of Integrated Formal Methods: Third International Conference, Turku, Finland.

Kleppe, A., Bast, W., & Warmer. B. (2003). *The model driven architecture: Practice and promise*. New York: Addison-Wesley.

Lassila, O., & Swick, R. (1999). *Resource description framework (RDF) model and syntax*. World Wide Web Consortium.

Ledeczi, A. (2001a). The generic modeling environment. In *Proceedings of the Workshop on Intelligent Signal Processing*, Budapest, Hungary.

Ledeczi, A., Bakay, A., Maroti M., Volgysei P., Nordstrom, G., Sprinkle, J., & Karsai, G. (2001b). Composing domain-specific design environments. *IEEE Computer, 34*(11), 44-51.

Lodderstedt, T., Basin, D., & Doser, J. (2002). SecureUML: A UML-based modeling language for model-driven security. *UML*, *2460*, 426-441.

Martin, G., Lavagno, L., & Louis-Guerin, J. (2001). Embedded UML: A merger of real-time UML and co-design. In *Proceedings of the 9th International Symposium on Hardware/Software Codesign*, Copenhagen, Denmark.

Microsoft Domain-Specific Language Tools. (2007). Retrieved March 23, 2008, from http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx

Moore, B. (2004). *Eclipse development using the graphical editing framework and the eclipse modeling framework*. Boca Raton, Florida: IBM, International Technical Support Organization.

Nechypurenko, A., Wuchner, E., White, J., & Schmidt, D.C. (2007). Application of aspect-based modeling and weaving for complexity reduction in the development of automotive distributed real-time embedded systems. In *Proceedings of the Sixth International Conference on Aspect-oriented Software Development*. Vancouver, British Columbia.

Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal, 7*(4), 308-313.

Object Management Group. (2007). *Meta object facility (MOF), version 1.4*. Retrieved March 23, 2008, from http://www.omg.org/docs/formal/02-04-03.pdf

Open Architecture Ware. (2007). Retrieved March 23, 2008, from www.openarchitectureware.org

O'Sullivan, J., Edmond D., & Ter Hofstede, A. (2002). What's in a service? Towards accurate description of non-functional service properties. *Distributed and Parallel Databases, 12*(2), 117-133.

Poizat, P., Royer, J., & Salaun, G. (2004, June). Formal methods for component description, coordination and adaptation. In *Proceedings of the 1st International Workshop on Coordination and Adaptation Techniques for Software Entities*, Oslo, Norway.

Quatrani, T. (2003). *Visual modeling with rational rose and UML*. Reading, MA: Addison-Wesley.

Schmidt, D.C. (2002). Middleware for real-time and embedded systems. *Communications of the ACM*, *45*(6), 43-48.

Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, *20*(5), 19-25.

Sztipanovits, J., & Karsai, G. (1997). Model-integrated computing. *IEEE Computer, 30*(4), 110-111.

Van Hentenryck, P., & Saraswat, V. (1996). Strategic directions in constraint programming. *ACM Computing Surveys*, *28*(4), 701-726.

Warmer, J., & Kleppe, A. (1998). *The object constraint language: Precise modeling with UML*. Boston, MA: Addison-Wesley.

Weber, M., & Weisbrod, J., (2002). Requirements engineering in automotive development-experiences and challenges. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering,* Essen, Germany.

White, J. (2005). *The generic eclipse modeling system.* Retrieved March 23, 2008, from http://www.sf.net/projects/gems

White, J., Gokhale, A., & Schmidt, D.C. (2007a). Simplifying autonomic enterprise Java Bean applications via model-driven development: A case study. *Journal of Software and System Modeling, 7*(1), 3-23.

White, J., Schmidt, D. C., Mulligan S. (2007, June). The generic eclipse modeling system. In *Proceedings of the Model-Driven Development Tool Implementer's Forum at TOOLS '07.* Zurich, Switzerland.

White, J., Nechypurenko, A., Wuchner, E., & Schmidt, D.C. (2006). Intelligence frameworks for assisting modelers in combinatorically challenging domains. In *Proceedings of the Workshop on Generative Programming and Component Engineering for QoS Provisioning in Distributed Systems*, Portland, OR.

White, J., Schmidt, D.C., Wuchner, E., & Nechypurenko, A. (2007b). Automating product-line variant selection for mobile devices. In *Proceedings of the 11th Annual Software Product Line Conference (SPLC),* Kyoto, Japan.

Willans, J.S., Sammut, P., Maskeri, G., & Evans, A. (2002). *The precise UML group.*

Yuan, W., & Nahrstedt, K. (2003). Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles,* Bolton Landing, NY.

ADDITIONAL READING

Bast, W., Kleppe, A.G., & Warmer, J.B. (2003). *MDA explained: The model driven architecture: Practice and promise.* Boston, MA: Addison-Wesley.

Beckert, B., Keller, U., & Schmitt, P.H. (2002). Translating the object constraint language into first-order predicate logic. In *Proceedings of VERIFY,* Copenhagen, Denmark.

Bézivin, J. (2001). From object composition to model transformation with the MDA. In *Proceedings of the 39th International Conference on the Technology of Object-Oriented Languages and Systems,* Santa Barbara, CA.

Bézivin, J. (2004). In search of a basic principle for model-driven engineering. *Novatica, 5*(2).

Bézivin, J. (2005). On the unification power of models. *Software and Systems Modeling, 4*(2), 171-188.

Bézivin, J., Farcet, N., Jezequel, J.M., Langolis, B., & Pollet, D. (2003). Reflective model-driven engineering. In *Proceedings of the 6th International Conference on the Unified Modeling Languages and Applications,* San Francisco, CA.

Budinsky, F. (2003). *Eclipse modeling framework.* Boston, MA: Addison-Wesley.

Clocksin, W.F., & Mellish, C.S. (1984). *Programming in prolog.* New York: Springer-Verlag

Coplien, J.O., & Schmidt, D.C. (1995). *Pattern languages of program design.* New York: ACM Press/Addison-Wesley.

Czarnecki, K., & Eisenecker, U.W. (2000). *Generative programming: Methods, tools, and applications.* New York: ACM Press/Addison-Wesley.

Frankel, D. (2003). *Model-driven architecture.* New York: John Wiley.

Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., & Natarajan, B. (2002). An approach for supporting aspect-oriented domain modeling. In *Proceedings of the Second International Conference on Generative Programming and Component Engineering*, Pittsburgh, PA.

Gray, J., Bapty, T., Neema, S., & Tuck, J. (2001). Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, *44*(10), 87-93.

Hillier, F.S. (2004). *Introduction to operations research*. New York: McGraw-Hill.

Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1), 143-168.

Mannion, M. (2002). Using first-order logic for product-line model validation. In *Proceedings of the Second International Conference on Software Product-lines*. San Diego, CA.

Mellor, S.J. (2004). *MDA distilled: Solving the integration problem with the model driven architecture*. Boston, MA: Addison-Wesley.

Northrup, L., Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longstaff, T., et al. (2006). *Ultra-large-scale systems: The software challenge of the future*. Pittsburgh, PA: Carnegie Mellon Software Engineering Institute.

Selic, B., & Rumbaugh, J. (1998). Using UML for modeling complex real-time systems. *Lecture Notes in Computer Science*, *1474*(1), 250-260.

Sterling, L.S., & Shapiro, E.Y. (1994). *The art of prolog: Advanced programming techniques*. Cambridge, MA: MIT Press.

Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. Cambridge, MA: MIT Press.

Vaziri, M., & Jackson, D. (2000). Some shortcomings of OCL, the object constraint language of UML. In *Proceedings of the 34th International Conference on the Technology of Object-Oriented Languages and Systems*, Santa Barbara, CA.

Warmer, J.B., & Kleppe, A.G. (2003). *Getting your models ready for MDA*. Boston, MA: Addison-Wesley.

White, J., Czarnecki, K., Schmidt, D.C., Lenz, G., Wienands, C., Wuchner, E., & Fiege, L. (2007). Automated model-based configuration of enterprise Java applications. In *Proceedings of the Enterprise Computing Conference (EDOC) 2007*, Annapolis, MD.

White, J., Schmidt, D.C., Wuchner, E., & Nechypurenko, A. (2007). Automating product-line variant selection for mobile devices. In *Proceedings of the 11th Annual Software Product Line Conference (SPLC)*, Kyoto, Japan.

This work was previously published in Designing Software-Intensive Systems: Methods and Principles, edited by P. Tiako, pp. 372-400, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.15
Heuristics and Metrics for OO Refactoring:
A Consolidation and Appraisal of Current Issues

Steve Counsell
Brunel University, UK

Youssef Hassoun
University of London, UK

Deepak Advani
University of London, UK

ABSTRACT

Refactoring, as a software engineering discipline, has emerged over recent years to become an important aspect of maintaining software. Refactoring refers to the restructuring of software according to specific mechanics and principles. While in theory there is no doubt of the benefits of refactoring in terms of reduced complexity and increased comprehensibility of software, there are numerous empirical aspects of refactoring which have yet to be addressed and many research questions which remain unanswered. In this chapter, we look at some of the issues which determine when to refactor (i.e., the heuristics of refactoring) and, from a metrics perspective, open issues with measuring the refactoring process. We thus point to emerging trends in the refactoring arena, some of the problems, controversies, and future challenges the refactoring community faces. We hence investigate future ideas and research potential in this area.

INTRODUCTION

One of the key software engineering disciplines to emerge over recent years is that of refactoring

(Foote & Opdyke, 1995; Fowler, 1999; Hitz & Montazeri, 1996; Opdyke, 1992). Broadly speaking, refactoring can be defined as a change made to software in order to improve its structure. The potential benefits of undertaking refactoring include reduced complexity and increased comprehensibility of the code. Improved comprehensibility makes maintenance of that software relatively easy and thus provides both short-term and long-term benefits. In the seminal text on the area, Fowler (1999) suggests that the process of refactoring is the reversal of software decay and, in this sense, any refactoring effort is worthwhile. Ironically, Fowler also suggests that one reason why developers do not tend to undertake refactoring is because the perceived benefits are too "long term." Despite the attention that refactoring has recently received, a number of open refactoring issues have yet to be tackled and, as such, are open research concerns. In this chapter, we look at refactoring from two perspectives.

This first perspective relates to the heuristics by which refactoring decisions can be made. Given that a software system is in need of restructuring effort (i.e., it is showing signs of deteriorating reliability), IS project staff are faced with a number of competing choices. To illustrate the dilemma, consider the question of whether completion of a large number of small refactorings is more beneficial than completion of a small number of large refactorings. A good example of the former type of refactoring would be a simple "rename method," where the name of a method is changed to makes its purpose more obvious. This type of refactoring is easily done. An example of the latter, more involved refactoring, would be an "extract class" refactoring where a single class is divided to become two. This type of refactoring may be more problematic because of the dependencies of the original class features.

As well as the decision as to "what" to refactor, we also look at the equally important decision as to "when" we should refactor. Throughout all of our analysis, we need to bear in mind that refactoring offers only a very small subset of the possible changes a system may undergo at any point in its lifetime. We return to this theme later on.

Combined with the need to choose refactorings and the timing of those refactorings, the need to be able to measure the refactoring process is also important. Software metrics (Fenton, 1996) provide a mechanism by which this can be achieved. A metric can defined as any quantifiable or qualitative value assigned to an attribute of a software artefact. The second perspective thus relates to the type of metric applicable for determining firstly, whether a refactoring is feasible, which of competing refactorings are most beneficial and how the effects of carrying out refactoring have impacted on the software *after* it has been completed. In terms of "when" to refactor, a metrics program implemented by an organization may provide information on the most appropriate timing of certain refactorings according to metric indicators as, for example, a rapid and unexplained rise in change requests.

For both perspectives investigated, there are a large number of issues which could possibly influence their role in the refactoring process. For example, most refactorings can at best only be achieved through a semi-automated process. For example, the decision on how to split one class into two can only be made by a developer (and aided by tool support once that decision has been made). Some metrics are subject to certain threats to their validity and are thus are largely inappropriate for judging the effect of a refactoring; the lines of code (LOC) metric is a good example of such a metric because of the unclear definition of exactly what a line of code is (Rosenberg, 1997). In our analysis, we need to consider these issues.

The objectives of the chapter are three-fold. Firstly, to highlight the current open issues in the refactoring field. In particular, some of the associated problems that may hamper or influence

any refactoring decision. Secondly, to highlight the role that software metrics of different types can play in the refactoring process and the interplay between refactoring mechanics and the measurement of refactoring. Throughout this chapter we appeal to a number of previous empirical studies to inform and support our ideas and opinions. A final objective of the chapter is to identify potential future research possibilities and concerns considering some of the problems and issues outlined.

The strategy we adopt for our analysis is as follows. We believe strongly that past and ongoing empirical evidence from a range of different systems provides the best mechanism for analyzing and achieving the goals of the chapter. Those goals are firstly, to distill from current empirical thinking (studies and metrics) the elements which impact on the theoretical and practical aspects of refactoring; secondly, to present that evidence in a relevant, interesting and meaningful way. Finally, to propose a set of heuristics from that evidence which we feel will be of value to refactoring practitioners, project managers, and researchers alike. We also feel that the results will be of interest to the wider software engineering community in terms of informing our understanding of change patterns and trends.

The chapter is arranged as follows. In the next section, we describe background and related work in the field of refactoring and discuss our viewpoint in a broad sense. Next, we focus on the "what" aspects of refactoring, drawing on previous empirical studies in the area to decide what refactorings to apply. We then look at the "when" of refactoring; when should we apply refactorings? Next, we summarize the heuristics presented in the chapter (in particular, those in previous sections) and then we describe some future research directions in the refactoring field. Finally, we draw some conclusions.

BACKGROUND

We view three interrelated areas as particularly relevant to our analysis; theory and the mechanics of refactoring, practical application and motivation of empirical studies of refactoring and finally, work in the metrics field. By "mechanics" of refactoring we mean the prescribed steps that need to be applied to complete a refactoring (Fowler, 1999).

Refactoring Theory and Mechanics

There are a number of works relevant specifically to refactoring that have contributed to the field and which could be said to be seminal. In terms of early work in the area, the main text and from which we will draw significantly in this chapter is that of Fowler (1999). In this text, Fowler describes the mechanics of 72 different refactorings and assorted "bad smells" in code. Bad smells in code have a special significance to the work in this chapter. According to Fowler, the key indicator of when refactoring is overdue is when code starts to "smell." An example of a bad smell is an inordinately long method and is thus an obvious candidate for splitting in two.

In the same text, Fowler categorizes the 72 refactorings according to four areas. These are whether a refactoring: makes method calls simpler, organizes data, moves features amongst objects, or deals with generalization. The Ph.D. work of Opdyke (1992), work by Johnson and Foote (1988) and Foote and Opdyke (1995) has also been instrumental in promoting refactoring as a discipline and demonstrating the viability of the refactoring approach. As well as investigating the "what" and "when" of refactoring, we also illustrate potential areas for novel empirical research to build on these foundations.

Most of the early refactoring literature focused on Java and Smalltalk as the target languages. The unique features of object-oriented (OO) languages (e.g., encapsulation and inheritance) make refac-

toring a particularly interesting challenge for the developer. For example, encapsulation issues and the need to conform to sound OO principles means that there is frequently a need to apply relatively *simple* refactorings. For example, the "encapsulate field" refactoring modifies the declaration of a field from public to private. The motivation according to Fowler (1998) is that:

One of the principal tenets of object-orientation is encapsulation, or data hiding. This says that you should never make your data public. (p. 206)

In the next section, we provide evidence that shows developers (for the C++ language) do not seem to attach importance to getting encapsulation "right"; the key (and worrying) point is that refactoring may require the breaking down of "bad developer habits."

In terms of the OO inheritance feature, there are a number of challenges for the developer. For example, ensuring that methods and fields are declared and used in the most appropriate place of an inheritance hierarchy. The "pull up field" refactoring for example requires that a field in a subclass is moved up to its superclass. According to Fowler, the motivation for this refactoring is that "two subclasses have the same field". In this case, the field in question should be moved to the superclass to avoid duplication of that field. This is a relatively simple refactoring related to inheritance. A less simple refactoring is the "pull up method" refactoring, where two identical methods are moved from subclasses to their superclass (again to avoid duplication of behaviour). Both the mechanics and testing effort required is significantly greater for the latter type of refactoring.

Empirical Studies

The benefits of refactoring are therefore clear in terms of qualitative (subjective) values. In terms of empirical studies, recent work by Najjar, Counsell, Loizou, and Mannock (2003) has shown that refactoring can deliver both quantitative and qualitative benefits; the refactoring "replacing constructors with factory methods" of Kerievsky (2002) was analyzed. The mechanics of the refactoring require a class to have its multiple constructors converted to normal methods, thus eliminating the code "bloat" which tends to occur around constructors. The moved methods thus have new, more meaningful names. Results showed quantitative benefits in terms of reduced lines of code due to the removal of duplicated assignments in the constructors as well as potential qualitative benefits in terms of improved class comprehension.

In Najjar, Counsell, and Loizou (2005), the problems associated with a simple refactoring such as the encapsulate field (EF) was studied. To investigate the EF refactoring, samples of classes were chosen from five different Java systems and the potential for applying the mechanics of the refactoring investigated. Results showed certain potential for applying the refactoring *per se*. In other words, no shortage of opportunity was found for applying the refactoring; public attributes were found in a number of classes in each system. However, three features exhibited by the five systems suggest that applying the EF refactoring is not as straightforward or applicable as it first seems. Firstly, the number of dependent classes requiring changes as a result of applying the refactoring may prohibit the refactoring; secondly, the large number of classes with zero attributes would seem to render the refactoring almost redundant. Finally, the features of the inheritance hierarchy in each system pose a dilemma with the use of the *protected* declaration (as opposed to private). A final finding was the practical trade-off and applicability of the EF refactoring when considering different application domains. Some of the systems studied were more amenable to the EF refactoring than others.

Recent work by Advani, Hassoun, and Counsell (2005b) describes the results of an empirical

study of the trends across multiple versions of open source Java software. A specially developed software tool extracted data related to each of fifteen refactorings from multiple versions of seven Java systems according to specific criteria. Results showed that, firstly, the large majority of refactorings identified in each system were the simpler, less involved refactorings. Very few refactorings related to structural change involving an inheritance relationship were found. Secondly, and surprisingly, no pattern in terms of refactorings across different versions of the software was found. Results thus suggested that developers tend to carry out simple "core" refactorings at the method and field level, but not as part of larger structural changes to the code (i.e., at the class level). The research in the same paper highlights an important refactoring issue. It is unlikely that we will be able to identify whether those "core" refactorings were done in a conscious effort by the developer to refactor, or as simply run-of-the-mill changes as part of the usual maintenance process. In other words, the question, "do developers refactor without realising it?" needs to be addressed. This then raises the question as to whether refactoring is subsumed by usual changes typically made by developers. Despite these issues, we feel that identification of the major refactoring categories is a starting point for understanding the types of change typically made by developers and the inter-relationships between changes typically made by developers. The same paper identified refactorings according to specific rules and heuristics. Developing heuristics for undertaking refactorings based on system change data has also been investigated by Demeyer, Ducasse, and Nierstrasz (2000).

Strategy Used for Empirical Studies

The empirical studies described as part of this chapter and from which we draw data were all undertaken over the past seven years. For each study, there was at least one underlying objective and/or hypothesis; as we describe each study, we point out what these were. This chapter represents an interleaving and distillation of these studies in a purely refactoring context. For one or two studies, the hypotheses were not stated from the outset. For example, it is difficult to reason about when most refactorings are likely to occur. On the other hand, hypotheses about cohesion and how human subjects would rate class cohesion are far easier to compose.

Many of the metrics used in the studies were collected manually and, wherever possible, collected automatically using tailored software. For example, the study described in the fourth section used human subjects as a basis and data from that study could only be collected from hand-written scripts. Data such as number of class attributes and methods, on the other hand, can easily be collected automatically. Where data was collected automatically, it was always verified afterwards through human inspection.

The threats to the validity of each study were also considered. For example, we tried to choose systems for each study that gave as wide a cross-section of application types as possible. We also tried to choose systems which were industrial-sized and which were developed by professional software engineers. Of course, we can never study too many systems and so many of results need to be supported by other studies in other research using other systems to build up a knowledge base in the area concerned. We have also provided evidence from both Java and C++ systems as a way of reflecting trends in different OO languages. Finally, we have included a variety of statistical techniques in this chapter; we chose different techniques for different studies as a way of highlighting the salient features we were trying to demonstrate.

Automation and Metrics

In terms of related work on automating the search for refactoring trends, research by Tokuda and Ba-

tory (2001) has shown that three types of design evolution, including that of hot-spot identification, are possible. A key result of their work was the automatic (as opposed to hand-coded) refactoring of fourteen thousand lines of code. Finally, the principles of refactoring are not limited to object-oriented languages. Other languages have also been the subject of refactoring effort including that of visual basic (Arsenovski, 2004).

A central feature of our analysis is the use of metrics to quantitatively capture the features of the system under study. Many metrics have been proposed and used for analyzing object-oriented and procedurally-based software both theoretically and empirically (Bieman & Ott., 1994; Briand, Devanbu, & Melo, 1997; Chidamber & Kemerer, 1994; Hitz & Montazeri, 1996). In most previous studies, we have used simple counts of the number of the class feature "number of attributes." Metrics play a central role in allowing us to measure features of systems at different levels of abstraction whether at the class or method level). In all the studies previously mentioned and studies we draw on in this chapter, metrics play a part.

Finally, as well as the need to understand "what" and "when" to refactor, it is also important to point to one other key motivation for our analysis of refactoring in this chapter. An earlier investigation by some of the authors to identify suitable candidates for refactoring failed for one simple reason. It highlighted obvious candidates for refactoring according to obvious criteria such as large numbers of class methods and attributes. Inspection of the same classes revealed very few opportunities for refactoring, because classes with large numbers of features often have those features for a good reason. For example, we found one class with several hundred attributes, a class called PageAttribute.MediaType. This class contained an attribute for each type of paper and other form of media (e.g., A4, A3, etc.). Refactoring this class would have been counterproductive in terms of benefits obtained, even it though it was identified according to specific refactorings and bad smells therein (e.g., large method, large class, and primitive obsession (Fowler, 1999). In the next section, we investigate the issue of "what" should be refactored and in the section following that the question of "when" refactoring should be done. We use results from previous empirical studies to support our arguments.

THE "WHAT" OF REFACTORING

One of the most fruitful research areas in recent years has been that of an empirical study. Carrying out empirical studies helps us to understand more in a quantitative and qualitative sense about how systems and the people using those behave (Bieman, Straw, Wang, Munger, & Alexander, 2003; Briand, Bunse & Daly, 2001; Counsell, Swift & Mendes, 2002; Harrison, Counsell & Nithi, 2000; Ostrand, Weyuker, & Bell, 2004). A multitude of empirical studies have thus been carried out covering all aspects of software engineering and related computer science fields. A particularly interesting area of empirical studies have been those which shed light on or which show how well stated theory stands up in practice. In this section we describe empirical experiences from which we can learn about rules and heuristics of what to refactor. In particular, we highlight some of the problems associated with refactoring observed through some of these empirical studies. More specifically, we highlight separately the empirical reality or refactoring and the applicability of refactorings thereof.

To facilitate an understanding of code features which the empirical studies try to evaluate, the following is a definition of a class APoint that models the operations of two coordinates x and y and two further attributes a and z. The class has a single attribute of each type of declaration (public, private and protected) and inherits from a class called BasePoint. It has a single constructor called APoint and a single method CalcDistance.

It is coupled to BasePoint through inheritance and to MathType via the return type in the method CalcDistance. We could also say that this class is reasonably cohesive because the methods are meaningful, operate on the same data and are named meaningfully. Although a simple class, the features demonstrate some of the major elements which empirical studies in later sections tackle in an empirical sense.

```
public class APoint extends BasePoint {
    public int x, y;
    private int a;
    protected int z;

 // constructor
 public APoint (int a, int x, int y, int z) {
    this.a = a;
    this.x = x;
    this.y = y;
    this.z = z;
 }

 public MathType CalcDistance() {
    return((x* y * a) + z);
 }

}
```

In the next section we investigate how the empirical reality is often different to the perceived reality with evidence to support the claims. We look at each study from a refactoring perspective.

The Empirical Reality

The applicability of certain refactorings relies, to a large extent, on the features of the application in question being present in that application. In the study where we replaced multiple constructors with a catchall constructor (Najjar, Counsell, Loizou, & Mannock, 2003), the study would have been impossible had every class had just a single constructor. The "what" to refactor is therefore dependent on the features of the refactoring being present in the systems under consideration. In subsequent sections, we use the terms "attribute" and "field" interchangeably.

Zero Attribute Classes

Figure 1 shows one of the results from the analysis of the "encapsulate field" refactoring (Fowler, 1999). The purpose of the study from which the data is taken was to empirically investigate the potential for simple refactorings. We wanted to show that even perceived trivial refactorings posed certain problems. Figure 1 shows the percentage of classes with zero attributes in samples taken from five Java systems.

One of the key impediments to this refactoring was thus the high percentage of zero-attribute classes found from the samples taken from each system. The applications ranged from a graph drawing package with the lowest proportion of zero-attribute classes (System 1) to the Swing class library (System 5) with the largest.

Table 1 shows summary data for the largest of the five systems investigated in the same research. It also shows the number of public features from the same sample. It shows that 52 of the classes from the sample size of 63 had zero attributes. Only 11 classes had more than one public attribute. Opportunities for the encapsulate field refactoring are thus limited to those 11 classes. The study thus cast doubt on the viability of even simple refactorings such as EF.

Another feature of the systems investigated in Najjar, Counsell, and Loizou (2005) is the existence of certain *key* classes (Counsell, Swift & Tucker, 2005). One obstacle to a decision as to what to refactor is the existence of certain class features which on the face of it, are excellent candidates for refactoring. One key indicator of a class which suggests it needs refactoring is a class with a large number of attributes. However, inspection of the class revealed it to require each

Figure 1. Five systems and the percentage of zero-attribute classes

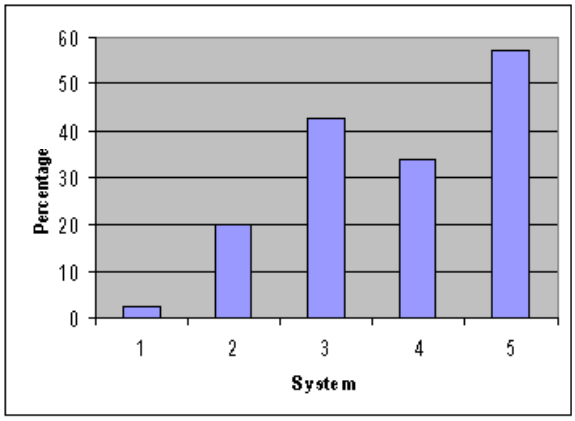


Table 1. Summary data for the Swing library

Sample Size	63
Classes with >= 1 public attr.	11
Classes with zero attributes	14
Classes with zero public attrs.	52
Max number of public attrs.	80

Table 2. Summary of dependencies for the Swing library

Sample Size	63
Median no. of dependencies	1
Max. no. of dependencies	14
Mean no. of dependencies	2.73

of those attributes as part of its essential functionality. The other problem with key classes is that they tend to have a large number of dependent classes. The purpose and underlying hypothesis of the investigation in Najjar et al. (2005) was to establish the problems associated with dependent classes; we believe that the larger the number of dependent classes, the harder it is, generally speaking, to refactor.

Inspection of the samples of classes chosen from System 3 (another library system) showed it to comprise a class called StrictMath, with

Table 3. Categorization of changes made to 52 Java classes

Change Type	Total	Distribution	Max.	Mean
New method added	32	7	14	0.62
Method call added	45	9	32	0.85
Parameter in method call added or modified	32	19	9	0.62
Method signature modified	51	26	14	0.98
"If" added or modified	67	14	11	1.26

112 attributes and 38 methods. Equally, System 1 (GraphDraw) has a class called GraphCanvas with 66 attributes and 63 methods.

Table 3 shows the summary data for the Swing system and shows that one class has 14 dependent classes. The important point about key classes is that while they may be eminently suitable for refactoring, the problem of dependent classes renders the process as problematic.

The conclusions we can draw from the data in Tables 1 and 2 relevant to the refactoring theme are that firstly, in many cases, the decision about refactoring is made for us by nature of the system itself. Deciding what to refactor is a decision aided partly by features of the systems themselves. If there are no attributes, then the number of potential refactorings is reduced significantly. In fact, the following are some of the refactorings which become impossible as a result:

- **Encapsulate field:** The declaration of a field is changed from public to private.
- **Move field:** "A field is, or will be, used by another class more than the class on which it is defined."
- **Rename field:** A field is renamed to make its purpose more obvious.
- **Push Down field:** "A field is used only by some subclasses." The field is moved to those subclasses.
- **Pull Up field:** "Two subclasses have the same field." In this case, the field in question should be moved to the superclass.

Secondly, the problem is more acute that the five refactorings above would suggest. The first three refactorings of the five listed are often part of larger refactorings (we will return to this principle in detail later). For example, the move field is part of the mechanics of the move method refactoring; if there are no attributes, then this is likely to make the move method process simpler. Absence of class features eliminates one refactoring and thus makes others simpler. Finally, in the existence of certain *key* classes may present a false impression of refactoring potential.

Ease of Automation

In a previous study described in Advani, Hassoun, and Counsell (2005b), we automatically extracted data about refactorings from seven Java systems of different types using a software tool (Advani, Hassoun, & Counsell, 2005a). The purpose of

the study was to investigate the potential for identifying refactorings automatically using a tool and some of what we perceived was common, popular refactorings. To do this, we developed rules for extracting refactoring data from the source code. For example, to detect whether the "Move Field" refactoring had taken place in the transition from one release to the next, the tool checked whether:

1. A field (name, type) that appeared in a class type (belonging to older version) appeared to be missing, that is, dropped from the corresponding type of a later version.
2. The field (name, type) did not appear in any superclass or subclass of the original type.
3. A similar field (name, type) appeared to have been added to another type (belonging to later version).

The mechanics of this refactoring are quite straightforward, and, therefore, we could easily automate it. According to Fowler, moving state and behaviour between classes is the essence of refactoring. However, certain refactorings can only be achieved through manual means and at best a tool can only assist. Consider the case of the "substitute algorithm" (SA) refactoring. The SA refactoring substitutes an algorithm for one which is clearer. The example given by Fowler is a series of "if" statements which can simply be replaced with a loop and an array. In the same way that the EF refactoring was relatively straightforward, the SA requires only that the code is changed and then run against a set of tests to ensure that the change has worked. However, changing code to meet the same requirements cannot be achieved by an automated process alone. There needs to be a manual component to the process.

In previous work (Counsell, Hassoun, Johnson, Mannock, & Mendes, 2003), it has been shown that the majority of the changes made to a Java Library system were to "if" conditions (yet interestingly not to "while" or "for" loops). The changes made to a set of fifty-two Java library classes over a three year period were investigated. The research attempted to support the hypothesis that certain types of changes made to Java code fall into distinct trends and, furthermore, are likely to be made at a high level of abstraction (i.e., at the method signature and parameter level).

Table 3 shows the distribution of changes categorized as part of the study in Counsell et al. (2003). The 67 additions or modifications to "if" statements were attributable to just fourteen classes. The maximum value in this table denotes the greatest number of changes of that type found for any single class. It shows that 32 new methods were added over the period to the classes studied. Interestingly, these 32 new methods were accounted for by only seven classes of the 52.

The problem from a refactoring point of view comes from two sources. Firstly, it is impossible for a tool to decide whether one section of code is functionally the same as another section of code unless it knows about the semantics of what each section of code does. Syntactically, we can make a wide range of observations about two sections of code. However, it would be virtually impossible to trace an instance of an SA refactoring (whether manually or automatically). This is particularly true if the change to the algorithm was complex in nature. In other words, and very much a topical refactoring issue, is that the most popular refactorings in the study described seem to be those which we cannot automate very easily. Secondly, the SA algorithm is likely to incorporate other refactorings; for example, the add parameter refactoring which may mask the SA refactoring even further. The lesson in terms of heuristics and relevant to the chapter is therefore that for convenience and speed (since developer time and resources are valuable and limited), refactorings which can be supported by a tool are likely to be a better investment of a developer's time. Some refactorings have no mechanics and this in turn make those refactorings difficult to apply.

The Role of Inheritance

Inheritance is claimed to be a fundamental principle of the OO paradigm. It is supposed to bring benefits in terms of reuse and inheritance hierarchy models information in a way which is easily understood and maintained. Despite the potential benefits of using inheritance, a number of studies have shown that the claims about ease of maintenance can be questioned (Briand et al., 2001; Counsell, Loizou, et al., 2002a).

In another recent study by the authors, it has been observed that the number of refactorings related to the category "dealing with generalization" were a very small part of the total overall number of refactorings. A tool was used to extract refactoring information from multiple versions of seven Java systems. The purpose of the study was to identify trends in core refactorings across a wide range of systems. Table 4 illustrates the number of refactorings extracted across the seven systems and *n* versions categorised according to the "dealing with generalization" type identified by Fowler.

It also shows the totals for that refactoring/version in the final row. Between versions 3-4, only 41 of the 236 refactorings were attributed to this category. Only 6 occurrences of the extract subclass were found in all versions of the systems (looking across the row). Clearly, the lack of inheritance-based refactorings is evident from Table 4. The large number of zero values suggests that

Table 4. Inheritance-based refactorings across multiple versions of seven Java systems

Refactoring Type	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10
Push Down Method	0	0	1	0	1	0	0	0	4
Extract Subclass	0	2	3	0	1	0	0	0	0
Pull Up Field	0	1	7	0	2	4	0	0	0
Extract Superclass	0	2	10	0	8	1	0	0	2
Push Down Field	0	16	3	0	7	0	0	0	0
Pull Up Method	0	9	17	0	24	5	0	0	10
Total	89	151	236	8	67	61	17	7	51

Table 5. Location of Java classes in the inheritance hierarchy

System	Leaf Classes	% of Total
Drawing Tool	5	2.45
Framework	34	20.06
Java Library	44	38.72
Compiler	73	32.59
Swing	577	54.59

across versions, inheritance-based refactorings are not common. The worrying trend is that the result suggests that developers avoid complex refactorings in favour of simpler refactorings (which accounted for the majority of the values in the final row of Table 4).

Evidence of the limited role that inheritance can play in the determination of refactorings can be found in Table 5. It shows that for the same five systems analyzed in Najjar et al. (2005), a high proportion of classes (except those for the drawing tool) are leaf classes, that is, have no subclasses. This feature should make the classes at leaves easier to refactor since, in theory they have less dependencies than classes with their own subclasses.

This would certainly apply to the encapsulate field refactoring where there are no subclasses dependent on the field in question. The same however could not be said of any refactoring which requires parts of the class to be moved — the methods of the class may use inherited features and this may cause problems.

The key lesson in terms of the chapter is that in certain circumstances, some refactorings may be trivial and easily completed. The same situation may however make other refactorings prohibitively expensive in terms of developer time and effort. Generally speaking, it seems that inheritance refactorings are so involved that they are avoided by developers. In the next section, we discuss the related issues of coupling and cohesion.

Cohesion and Coupling

We have seen already that one of the key impediments to refactoring for even a simple refactoring is the role that dependencies between classes plays. An accepted software engineering principle is that developers should try to make classes as cohesive as possible and that those classes should contain minimal coupling (Pressman, 2005). In the metrics community, cohesion is often associated with ease of comprehension. If a class is cohesive, then in theory it should be easy to understand. If a class is uncohesive then the purpose of the class is not obvious and it is difficult to understand what the purpose of that class is.

The best known of the cohesion metrics is that proposed by Chidamber and Kemerer (C&K) — the lack of cohesion of the methods of a class LCOM (Chidamber et al., 1994). The LCOM metric views a class as cohesive if the instance variables are distributed in such a way that all methods use all instance variables. Equally, a class is viewed as uncohesive if the use of instance variables amongst the methods of a class are disjoint. Various other metrics have been proposed to measure cohesion, but, as of yet, there is no general consensus on cohesion metric. In other words, we have no accepted way of measuring the benefit of a refactoring such as "extract class" whose purpose is to remove code from one class to make the source class more cohesive.

In terms of other related work, a number of attempts have been made to capture cohesion through software metrics. As well as the C&K LCOM metric, the cohesion amongst the methods of a class metric (CAMC) by Bansiya, Etzkorn, Davis, and Li (1999) based on the overlap of parameters used by a class was found to correlate with LCOM and the views of three developers on what constituted cohesion. Hitz and Montazeri (1996) also propose metrics for measuring cohesion (as well as coupling). Bieman and Ott (1994) demonstrated the measurement of functional cohesion in C software. Finally, Briand et al. (1998) propose a framework for measurement of OO cohesion and conclude that many of the cohesion metrics proposed are in most cases not validated theoretically and even fewer validated empirically.

In terms of refactoring, high cohesion would seem to be a synonym for high comprehensibility. If a class is cohesive, then the class will be easier to understand and modify. We note that any measure of cohesion based on the attributes

of a class cannot be assessed if the class has no attributes. In this chapter, we adopt the stance that coupling is a far better indicator of comprehensibility than any measure of cohesion. High coupling will make refactorings more difficult to apply. Lower coupling will make them easier to apply. This brings into question the whole issue of the coupling and cohesion interplay.

As part of an earlier study into OO (C++) cohesion, the correlation between coupling given by the number of associations metric (NAS) metric and a component of a metric called normalised hamming distance (NHD) metric was analyzed. The NAS represents the number of couplings of a class and can be counted as the number of lines emerging from a class on a UML class diagram. The NHD is based around the inter-relationship between the parameters of the methods of a class in a similar vein to the CAMC metric. The purpose of the study was to investigate the hypothesis that, in an OO sense, cohesion and coupling were so interrelated that one could be substituted for another.

Table 6 shows the strong relationship between the key component of our cohesion metric P (number of unique object "P"arameters in the class as a whole) and NAS metric; the results are a summary of the results from the study in Counsell, Swift, et al. (2002). Correlation at the 1% level is asterisked by the value in each case. We thus adopt the stance that firstly, coupling has a strong relationship with cohesion (where method parameters are assumed to be a key component of class cohesion). Secondly, that unlike coupling, cohesion is a subjective issue and cannot be measured objectively.

Reduced coupling has also been a claim of certain refactorings. For example, the motivation behind the "move field" refactoring is that "a field is, or will be, used by another class more than the class on which it is defined. It therefore makes sense to move that field to the class which needs to most and eliminate the coupling. Various metrics have been defined to measure coupling (Briand et al., 1997; Chidamber et al., 1994). The difficulty arises when the decision as to what represents too much coupling has to be made. For certain applications, a high level of coupling may be necessary. There is also the problem that certain types of coupling are more of a feature in some applications than others. A GUI-based system (with a high dependence on inheritance) lends itself well to a structure incorporating frames, panels and dialog boxes, all of which share certain generic properties. Although some amount of work has been done on finding an optimal level of coupling (El Emam, Benlarbi, Goel, Melo, Lounis & Rai, 2002), further empirical studies need to be carried out before a consensus can be reached.

When making refactoring decisions, we therefore suggest that coupling should be the prime determinant of which refactorings to carry out. We should choose refactorings which can be measured in reduced levels of coupling rather than aiming

Table 6. Pearson, Kendall and Spearman's correlation coefficients (NAS vs. P)

System	Pearson's	Kendall's	Spearman's
Framework	0.59*	0.61*	0.69*
Compiler	0.41	0.28	0.30
Graph Editor	0.83*	0.62*	0.79*

for high cohesion. Furthermore, in assessing the post-impact of any refactoring, coupling not cohesion should be used where possible. We accept that many refactorings involve no coupling issues and for these refactorings, the decision amongst competing refactorings may require subjective judgements.

Another point in terms of the overall chapter is that we should always look for refactorings which provide quantifiable benefits. In the next section, we address the issue of when to refactor; that is, we have looked at "what" to refactor, but the "when" is equally important.

THE "WHEN" OF REFACTORING

The decision as to when to refactor is as important a decision as to what to refactor. Fowler suggests that rather than being one large concerted effort, refactoring should be done in little bursts. He also suggests that a developer does not decide to refactor; the developer refactors because he/she wants to achieve something else and doing that "something else" requires that they refactor first. Similarly, Beck (2000) urges developers to refactor when the system tells them to, not through speculation alone. Very little research has empirically tackled the issue of when we should refactor.

To support the arguments about when to refactor, we return to the study of the seven open source Java systems analyzed in Advani et al. (2005b). The first question which arises is whether there any patterns within the systems studied as to "when" refactoring is carried out. Figure 2 shows that for the Antlr system, the majority of refactorings happen at versions 2-3 and 3-4. It is noticeable that some refactorings (particularly inheritance-based refactorings) are few and far between. Figure 3 for the same system shows the same trend for the HSQLDB system and similarly for the JasperReports system in Figure 4. In each

Figure 2. Refactorings for the Antlr system across five versions

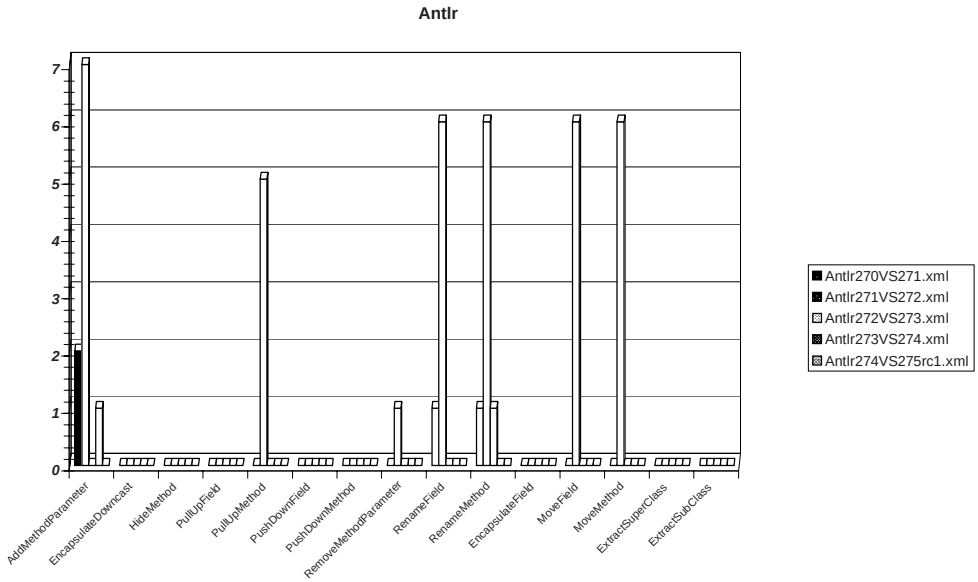


Figure 3. Refactorings for the HSQLDB system across four versions

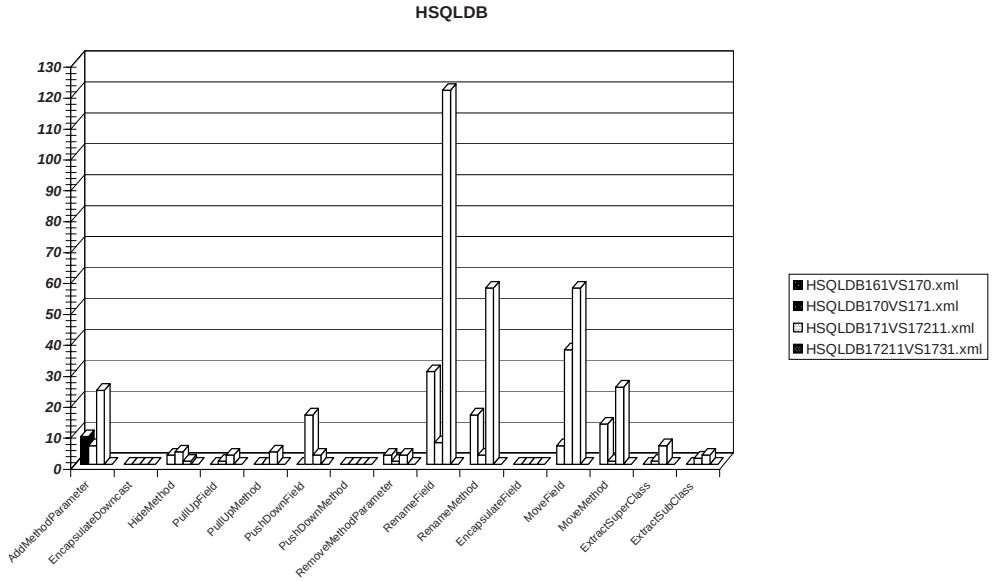


Figure 4. Refactorings for the JasperReports system across four versions

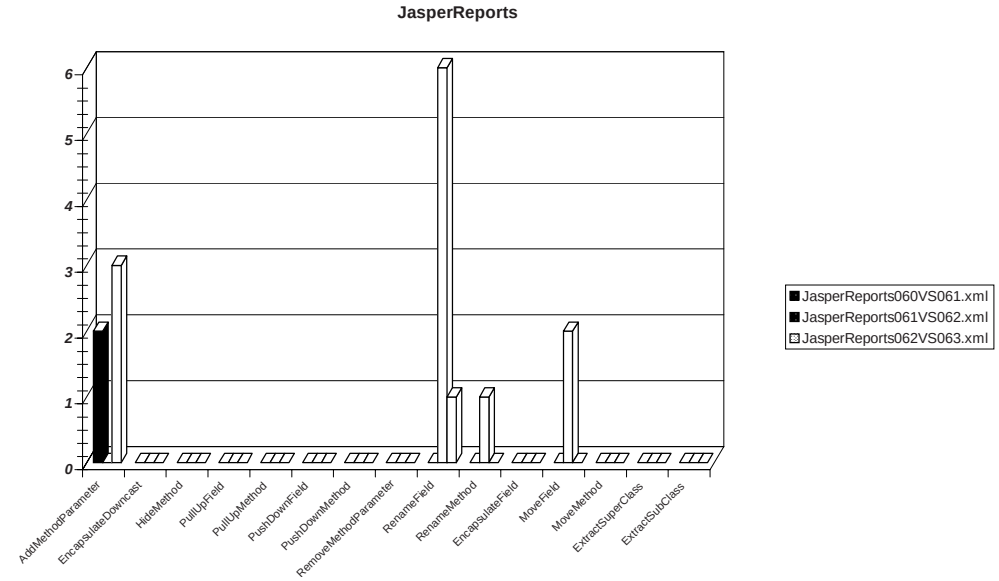
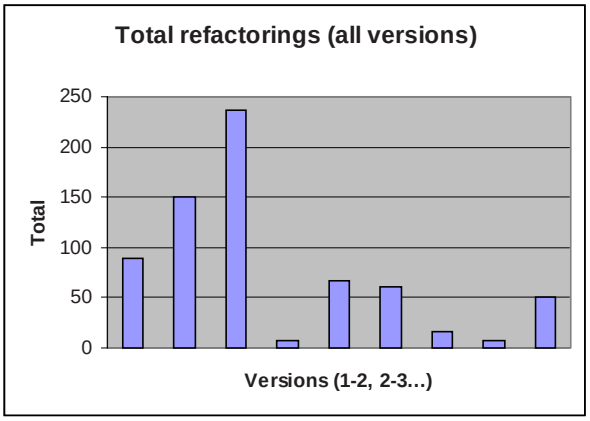


Figure 5. Refactorings across versions of the seven systems



of the figures, there seems to be a trend of refactoring being undertaken at earlier stages of the system's life rather than at later versions.

Figure 5 shows the cumulative values of refactorings across time for all seven systems studied. It supports the results for the individual systems and shows that the bulk of refactorings tend to occur not at the earliest stages of a system, but around versions 2-3 and 3-4.

The evidence from Figures 2, 3, and 4 therefore supports the view that in terms of the need for refactoring, a system starts to degrade or decay after 3 or 4 versions.

The conclusion we can draw from the evidence in Figures 2, 3, 4, and 5 is that contrary to what Fowler states would be the best time to refactor (at constant intervals), there seems to be a peak around versions 3-4 for the systems studied in terms of when refactoring takes place. From Figure 5, it is also noticeable that after version 3-4, there is a dramatic drop in the number of refactorings. Although the study described relates to only seven systems, we feel that the results give useful evidence of the type of refactoring trends in systems.

Of course, we have no means of predicting the trend in refactoring beyond the versions studied, and it would be as unreasonable to suggest that there would not be any more peaks. Although the empirical evidence suggests that we need to devote refactoring efforts at early stages of the system's lifetime, we would urge consistent refactoring throughout the period of a system's life. Of course, more empirical studies need to be undertaken before we can draw any concrete conclusions.

Monitoring Growth

One of the key benefits of carrying out refactoring is reduced class complexity. For example, the motivation behind the "extract class" refactoring is that a class should be split into two classes because it is becoming too large to understand easily. In other words, the class is becoming too complex. One current issue is therefore that as a result of carrying out refactoring, we may well attain improved comprehension, but the difficulty arises when we try to measure that benefit. One conclusion is thus that if a class has zero attributes,

then it affects the number of refactorings which can be carried out and also the means by which we can measure the outcome of any refactoring. One of the other key potential benefits of refactoring is a reduction post-refactoring in the number of lines of code. However, the lines of code added to a class together with the number of changes that have been applied to a class may give a good indication of the potential for refactoring.

A previous study by Bieman et al. (2003) investigated the hypothesis that large, object-oriented classes were more susceptible to change than smaller classes. The measure of change used in the study was the frequency with which the features of a class had been changed over a specific period of time. From a refactoring perspective, the frequency of class change in terms of its versions is of real value. However, since for any relatively simple refactoring, multiple classes may be changed without any net increase in class (and hence overall system) size, we contest that including number of added lines of code in the assessment gives a better impression of which classes are most volatile and hence more suitable candidates for refactoring.

We thus support the view that refactoring should be applied to classes with a high growth rate as well as a high change frequency. To support our investigation, data relating to changes from 161 Java classes were used as an empirical basis. Our results supported the conclusion of Bieman et al. (2003) relating to the change-proneness of large classes. Finally, we believe refactoring effort is most economically applied to classes which have both a high number of versions and a large increase in added lines of code over their lifetime. The "when" to refactor should be informed by sudden changes to the system.

A high frequency of changes made to an object-oriented class may suggest that the class was poorly written and hence required a large amount of maintenance effort applied to it. In a refactoring sense, frequency of change takes on a different meaning. It is thus likely that refactoring will not significantly increase the number of physical lines of code in the system since most refactorings require either modifications to current code or corresponding deletions for each insertion required by the refactoring. The extract class refactoring is one example where code is removed from one class to become a class in its own right without any significant net increase in system size. In the recent study by Bieman et al., number of class changes was used as the measure of change-proneness for five C++ systems. While we believe that there is merit in using number of changes as a measure of class volatility, we believe that classes which have had a large number of changes made to them (i.e., have many versions) and which have had significant numbers of lines of code added to them are better potential candidates for refactoring. A previous study by some of the authors collected the change data and growth in class size from one hundred and sixty one Java classes. It was found that much more information about the growth f the system could be gleaned by using both number of changes and changes in the lines of code in those classes. Basing the choice of classes for refactoring on alarming increases in class size supports the dictum of Beck which urges developers to refactor when the system tells them to, not through speculation alone.

Experimental Evidence

The obvious experimental means of establishing whether comprehension has improved is to carry out a formal experiment using developers, but any experiment is subject to certain threats to validity. For example, the level of experience of the subjects used may influence the results. The measurement of experience is also itself subject to a range of criticisms.

Often, a refactoring is accompanied by a reduction in the number of lines of code. This was a feature of the refactoring undertaken as part of Najjar et al. (2003). The "replace constructors with factory methods refactoring" removed

duplicate lines in constructors which had arisen due to code "bloat" (Fowler, 1999). However, the lines of code (LOC) metric has been subject to a range of criticisms (Rosenberg, 1997) and so any refactoring which reduces the number of LOC is subject to the same criticism.

Table 7 illustrates the role that size and, in this case, comment lines can have on the perception of cohesion by subjects taking part in a controlled experiment (Counsell et al., 2005). Twenty-four subjects with varying levels of experience were asked to rate on a scale 1-10 how cohesive they thought a set of ten C++ classes (10 represents the most cohesive class, and 1 the least cohesive). The 24 subjects were each given a set of the ten C++ class header files being analyzed. The ten classes were chosen at random from two industrial-sized C++ systems. The only restriction placed on the choice of these classes was that there had to be a relatively broad range of class size and make-up, but at the same time not too wide a range as to bias the outcome of the study.

Table 7 shows the "Position" of the classes in terms of the rating of cohesion by experienced subjects. Class ApplnDialog was thus rated the least cohesive and class ArcList the most cohesive. The Number of Comment Lines (NCL) in the class is followed by the position rated by subjects without experience. For example, class ApplnDialog was ranked the seventh most cohesive class. Class BagItem was rated most cohesive by subjects without experience. The number of methods in the class (NMC) also included in the table is a measure of the size of the class.

Table 7 also shows that classes with relatively larger numbers of comment lines were generally considered by the experienced subjects to be cohesive. The same is true of the inexperienced group. Clearly, the top two classes in terms of comment lines were ranked relatively highly in terms of their cohesion values (by the inexperienced group). This would seem to indicate that comment lines are an aid to the assessment of cohesion. However, in saying this, an allied factor (or even the critical factor) may be the low NAS values found for those classes.

In other words, on the one hand, size and growth are important in our determination of when to refactor, but there may be features which do not contribute to size necessarily, but yet are

Table 7. Summary data for an empirical cohesion study

Position	Class Name	NCL	Position (Inexp,)	NMC
1.	ApplnDialog	0	7	5
2.	Alert	0	5	7
3.	Dialog	2	8	4
4.	CycleItem	0	2	14
5.	Arc	29	10	5
6.	Bitmap	0	3	22
7.	BagItem	3	1	11
8.	Assoc	3	4	11
9.	ArcList	47	9	9
10.	DDGNodePtrList	54	5	9

crucial to the mechanics of most refactorings (i.e., coupling). In the next section we describe a previous analysis carried out to see if, in an empirical sense, one refactoring triggers other refactorings.

A Dependency Analysis

As part of our refactoring research, we developed a dependency diagram which showed the inter-relationships between the 72 refactorings originally stated by Fowler. The diagram was developed by hand using Fowler's text as a basis. As a result of producing this graph, it becomes possible to see the likely implications of undertaking a specific refactoring in terms of how many other potential refactorings either *must* be carried out or *may* be carried out at the same time. In terms of the question about "when" and "what" to refactor, we must accept the possibility that one refactoring may embrace *n* other refactorings, and this would be an important consideration in the choice of both what and when to refactor.

For example, for the "Encapsulate Field" refactoring, Fowler (1998) himself suggests that one possible implication of the refactoring is that once he had completed encapsulate field he would look for methods that use the new methods (i.e., accessors needed for the encapsulated field) "to see whether they fancy packing their bags and moving to the new object with a quick Move Method" (p. 206).

The encapsulate field refactoring thus has only one possible "dependency." From a developer's point of view, the encapsulate field is an attractive and relatively easy refactoring to complete. The "add parameter" refactoring falls into the same category as the encapsulate field refactoring. It does not need to use any other refactorings. The only other refactoring that it may consider using is the "introduce parameter object" refactoring where groups of parameters which naturally go together are replaced by an object.

The extract subclass refactoring, on the other hand, requires the use of six (possible) other refactorings, two of which are mandatory. It has to use "push down method" and "push down field" as part of its mechanics. It *may* (under certain conditions) also need to use the "rename method," "self encapsulate field," "replace constructor with factory method," and "replace conditional with polymorphism" refactorings. The extract superclass refactoring requires a similar number of refactorings to be considered. In fact, for most of the refactorings involving a restructuring of the inheritance hierarchy, the mechanics are lengthy (requiring many steps and testing along the way).

Connections Between Refactorings

One explanation for the result found in Advani et al. (2005b) (i.e., the high values for simple refactorings and the low values for more "complex" refactorings) could be attributed to the relative effort required in terms of activities required to complete the refactoring. The testing effort of more complex refactorings has also to be considered; the more changes made as part of the refactoring then other things remaining equal, the more testing would be required.

In terms of whether refactorings are somehow linked, we can see from Table 8 that when the extract superclass refactoring is evident, the pull up method is also a feature for those versions. The mechanics of the extract superclass refactoring insist that pull up method is part of that refactoring. Equally, there seems to be evidence of pull up field for the same refactoring, also a part of the extract superclass refactoring. Rename field and method also seem to feature when extract superclass is carried out; rename method (but not rename field) play an important role in the extract superclass refactoring. The rename field refactoring is not specified in Fowler's text. This is interesting since it suggests that may be some

Table 8. Breakdown of related refactorings from study in Advani et al. (2005b)

Refactoring Type	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10
Pull Up Field	0	1	7	0	2	4	0	0	0
Extract Superclass	0	2	10	0	8	1	0	0	2
Extract Subclass	0	2	3	0	1	0	0	0	0
Pull Up Method	0	9	17	0	24	5	0	0	10
Rename Method	19	15	71	6	16	21	1	2	16
Rename Field	31	22	137	0	2	5	1	1	10

effects of refactoring which aren't covered by the refactoring according to Fowler.

Extract subclass also requires use of the rename method refactoring, which may explain the high numbers for that refactoring. To try and explain the high numbers of rename field refactoring, one theory may be that developers automatically change the name of fields when methods are "pulled up" (in keeping with the corresponding change of method name). A conclusion that we can draw is that there may well be relationships between some of the fifteen refactorings in line with the mechanics specified by (Fowler, 1999). However, we suggest that most of the simple refactorings were not as part of any larger refactoring, based on the very low number of "larger" refactorings. When considering refactoring, we have to understand the implications of carrying out what may appear to be a straightforward refactoring. In the next section, we summarize and distill the heuristics that the previous two sections have presented.

Summary of Heuristics

In the third and fourth sections, we identified a number of experiments and empirical studies as a means of demonstrating firstly, what should be refactored and secondly, when refactoring should be undertaken. A number of key indicators were identified as a result, most based on data from those studies. In this section, we summarize and distill the heuristics and metrics identified in the third and fourth sections. We begin by proposing six heuristics which we feel could be applied in a refactoring sense.

Heuristics

The first heuristic that we propose is *Look at the trends in class features and class dependencies of your system before you attempt any refactoring.* Many systems have evolved to contain very few of the features that lend themselves to refactoring. For example, the relatively low number of attributes, severely restricting the possibility of refactorings related to attributes. The same can be said of key classes, that is, those classes which have many dependencies. Care should be exercised in any refactoring because of the potential for mistakes.

The second heuristic that we propose is *Accept that automation is realistic for a relatively small subset of refactorings. Many of the more complex refactorings can only be achieved manually with tool support.* The example which we used was

that of the Substitute Algorithm refactoring. Identifying what code has been changed and how is best undertaken manually. Tools can help, but only as support.

The third heuristic that we propose is *Within an inheritance hierarchy, dependencies of descendent class should be a prime consideration in making any refactoring decision.* Many of the relatively simple refactorings proposed by Fowler are complicated by the need to account for affected classes in the inheritance hierarchy.

The fourth heuristic that we propose is *Coupling (rather than cohesion) should be the feature of a class we aim to optimise.* Cohesion is a subjective concept and for any refactoring, we should try to eliminate where possible any subjectivity.

The fifth heuristic that we propose is *Consistent effort should be applied to the refactoring process whenever possible; growth of the system should also be monitored.* Despite the fact that empirical evidence suggests a surge in refactorings at version 3-4 of a system, we would encourage a smooth and consistent use of refactoring techniques.

Finally, we propose that: *When considering any refactoring, we need to appreciate that other refactorings may also be necessary due to a dependency between refactorings.* There is some empirical evidence of a nesting of refactorings; this may have implications for the cost both in time and financially of making a refactoring change.

Metrics

From the analysis of the different empirical studies, and in the same sense that we proposed heuristics for refactoring, we can propose six metrics which would provide the refactorer with an indication of "what" to refactor and "when" to refactor. These can be summarized as:

1. The number of attributes (public, private and protected)
2. The number of methods (public, private and protected)
3. Number of descendents of a class in the inheritance hierarchy
4. Number of classes to which a particular class is coupled
5. Changes in LOC to classes
6. Changes in "the number of changes" applied to a class

Interestingly, the set of six metrics include both product metrics, aimed at the static program code (metrics 1-4) and process metrics, aimed at what happens to the program over its lifetime (metrics 5-6). We note also that the metrics should be used in combination and not in isolation. For example, metrics 1 and 2 are very often related in terms of refactoring mechanics. Metric 4 includes inheritance coupling (metric 3), and as we suggested in the fourth section, metrics 5 and 6 should both be used to target classes growing at a relatively higher rate than other classes. Finally, the existence of key classes (the third section) embraces and requires the monitoring of all six metrics. (It is noteworthy that metrics 2-4 have corresponding equivalents in the C&K set of metrics and metric 1 is used in the computation of the LCOM metric, also of C&K.) In the next section, we point to future directions in the refactoring sense.

FUTURE DIRECTIONS

Some of the issues outlined in previous sections have tended to cast doubt on the viability of certain refactorings. Some have shown how quantitative and qualitative benefits can accrue from undertaking refactorings. Some of the issues have shown that we should focus on the coupling levels of the classes as a mechanism for deciding whether to refactor or not. The first area which the refactoring area could benefit from is a series of tools to guide the refactoring process. These tools should

indicate the quantitative and qualitative effects of carrying out that refactoring in terms of other refactorings also applicable; simulating the effect and mechanics would provide the developer with valuable information about the structure and state of the application being considered. Software metrics could be used at each stage of the refactoring mechanics to inform any such decision. This may help in the quest for "what" to refactor. The metrics which guide this process would need to be chosen carefully, however.

In terms of when to refactor, we would envisage that useful future research would be to investigate the key indicators which would help the development staff to know that refactoring is overdue. For example, if a subset of classes appears to require a disproportionate amount of maintenance effort then this should be a warning signal to the development staff.

In terms of *whether* to refactor (are the costs outweighed by the benefits?), a significant piece of research, already started by the authors, and a relatively short-term future direction would be to identify the relationships between the different refactorings (including those herein) and the occurrence of faults. In other words, is there a correlation between, let us say, the changes made to method signatures and consequent occurrence of faults directly related to that change? This research could lead to a refactoring order which states the relative possibility of faults arising should a particular refactoring be made. On the other hand, it may also indicate typical areas for refactoring effort to be directed and invested.

A further future direction would be use of appropriate intelligent data analysis techniques for simplifying computationally difficult problems. Identification of many of the more complex refactorings, for example, the Substitute Algorithm refactoring (third section) are very difficult to automate; they would require some form of heuristic search to be tackled effectively. Future research could investigate the potential for applying the different algorithms in a refactoring sense. Such techniques may also be able to provide predictive features for estimating the likely impact of undertaking a single or combination of refactorings. Use of simulation techniques may also be a fruitful research topic in this context for demonstrating the benefits of refactoring. Software metrics could play a key role in this sense.

This chapter has used a series of ongoing experiments and empirical studies as a basis of many of its claims. Finally, an important direction which the empirical research community should take in the future is thus to build up a body of experimental knowledge from which we can learn about refactoring (in general), the possibilities for applying new refactorings and the dissemination of information about refactoring. This knowledge should form a freely-available repository of data and other resources to inform the process of what and when to refactor.

CONCLUSION

In this chapter, we have tried to show how empirical studies have informed our understanding of refactoring. More empirical studies of various types need to be undertaken to build up a body of knowledge about refactoring before any conclusions can be drawn. We have also not included in this chapter any discussion about the role and relationship that refactoring has with the occurrence of faults. In other words, does *not* undertaking certain types of refactoring cause faults to arise? Equally, does refactoring uncover faults through the extra testing necessary as part of the refactoring mechanics?

In this chapter, we have also hypothesized on a number of occasions that developer habits may cause systems to deteriorate such that refactoring is then necessary. Future research directions may also include an analysis of developer habits as a good indication of where systems are beginning to decay. We have also described some of the current open issues in the field of refactoring. We

have investigated the features of refactoring and looked at the area from two perspectives. Firstly, we attempted to answer the question of "what" to refactor and looked at a number of issues from an empirical viewpoint which either lend themselves or do not lend themselves to refactoring. We have also investigated the question of "when" to refactor. Secondly, we have shown that empirical evidence suggests that refactoring is done in bursts towards the start of the system's lifetime, rather than as Fowler suggests that refactoring needs constant and consistent effort.

Finally, the question which we haven't been able to answer in this chapter is the relationship between refactorings as we've described them and the wide range of other changes made to software as part of the run-of-the-mill maintenance process. This could be an interesting area of potential research, not least because of the discussion on the link between certain refactorings in the previous section.

The key conclusions from this chapter are that we need a good understanding of the features and trends in systems at different levels of detail before we should attempt refactoring. The decision on which refactorings to carry out also needs to be planned carefully since they will require significant effort both in their mechanics and subsequent testing effort. We also feel that development staff should "listen" to the system and use metrics to provide information on what is happening to a system and hence inform the refactoring process. Ultimately, we would want to minimize the amount of time developers spend carrying out maintenance. Although refactoring takes time and effort, the general consensus is that effort expended in the short-term will provide real benefits in the long-term.

REFERENCES

Advani, D., Hassoun, Y., & Counsell, S. (2005a). *Heurac: A heuristic-based tool for extracting refactoring data from open-source software versions* (Tech. Rep. No. BBKCS-05-03-01). SCSIS-Birkbeck, University of London.

Advani, D., Hassoun, Y., & Counsell, S. (2005b). *Refactoring trends across N versions of N Java open source systems: An empirical study* (Tech. Rep. No. BBKCS-05-03-02). SCSIS-Birkbeck, University of London.

Arsenovski, D. (2004). Refactoring — elixir of youth for legacy VB code. Retrieved April 15, 2006, from *http://www.codeproject.com/vb/net/Refactoring_elixir.asp*

Bansiya, J., Etzkorn, L., Davis, C., & Li, W. (1999, January) A class cohesion metric for object-oriented designs. *Journal of Object-Oriented Programming*, 47-52.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Bieman, J., Straw, G., Wang, H., Munger, P. W., & Alexander, R. (2003, September 3-5). Design patterns and change proneness: An examination of five evolving systems. In *Proceedings of the 9th International Software Metrics Symposium (Metrics 2003)*, Sydney, Australia (pp. 40-49).

Bieman, J., & Ott., L. (1994). Measuring functional cohesion. *IEEE Transactions on Software Engineering, 20*(8), 644-657.

Briand, L., Bunse, C., & Daly, J. (2001). A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering, 27*(6), 513-530.

Briand, L., Daly, J., & Wust, J. (1998). A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering Journal, 3*(1), 65-117.

Briand, L., Devanbu, P., & Melo, W. (1997, May 17-23). An investigation into coupling measures for C++. In *Proceedings of the 19th International*

Conference on Software Engineering (ICSE 97), Boston (pp. 412-421).

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering, 20*(6), 467-493.

Counsell, S., Hassoun, Y., Johnson, R., Mannock, K., & Mendes, E. (2003, June16-18). Trends in Java code changes: The key identification of refactorings. In *Proceedings of the ACM 2nd International Conference on the Principles and Practice of Programming in Java*, Kilkenny, Ireland (pp. 45-48).

Counsell, S., Loizou, G., Najjar, R., & Mannock, K. (2002a). On the relationship between encapsulation, inheritance and friends in C++ software. In *Proceedings of the International Conference on Software System Engineering and Its Applications (ICSSEA'02),* Paris.

Counsell, S., Swift, S., & Mendes, E. (2002b). Comprehension of object-oriented software cohesion: The empirical quagmire. In *Proceedings of the IEEE International Workshop on Program Comprehension* (pp. 27-29), Paris, France.

Counsell, S., Swift, S., & Tucker, A. (2005). *Subject perceptions of object-oriented cohesion: An empirical study* (Tech. Rep. No. BBKCS-05-03-03). SCSIS-Birkbeck, University of London.

Demeyer, S., Ducasse, S., & Nierstrasz, O. (2000, Ocotober 15-19). Finding refactorings via change metrics. In *Proceedings of the ACM Conference on Object-oriented Programming Systems Languages and Applications (OOPSLA),* Minneapolis, MN (pp. 166-177).

El Emam, K., Benlarbi, S., Goel, N., Melo, W., Lounis, H., & Rai, S. N. (2002). The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering, 28*(5), 494-509.

Fenton, N., & Pfleeger, S. (1996). *Software metrics: A rigorous and practical approach*. London: Thomson International Publishing.

Foote, B., & Opdyke, W. (1995). Life cycle and refactoring patterns that support evolution and reuse. In J. O. Coplien & D. C. Schmidt (Eds.), *Pattern languages of programs.* Boston: Addison-Wesley.

Fowler, M. (1999). *Refactoring (improving the design of existing code).* Addison-Wesley.

Harrison, R., Counsell, S., & Nithi, R. (2000). Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software, 52,* 173-179.

Hitz, M., & Montazeri, B. (1996). Chidamber and Kemerer's metrics suite: A measurement theory perspective. *IEEE Transactions on Software Engineering, 11*(4), 267-271.

Johnson, R., & Foote, B. (1998, June-July). Designing reusable classes. *Journal of Object-Oriented Programming, 1*(2), 22-35.

Kerievsky, J. (2002). Refactoring to patterns, industrial logic. Retrieved April 15, 2006, from *http://www.industriallogic.com*

Najjar, R., Counsell, S., & Loizou, G. (2005). *Encapsulation and the vagaries of a simple refactoring: An empirical study* (Tech. Rep. No. BBKCS-05-03-02). SCSIS-Birkbeck, University of London.

Najjar, R., Counsell, S., Loizou, G., & Mannock, K. (2003, March 26-28). The role of constructors in the context of refactoring object-oriented software. In *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR '03),* Benevento, Italy (pp. 111-129).

Opdyke, W. (1992). *Refactoring object-oriented frameworks.* PhD thesis, University of Illinois.

Ostrand, T., Weyuker, E., & Bell, R. (2004, July 11-14). Where the bugs are. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 86-96), Boston.

Pressman, R. (2005). *Software engineering: A practitioner's approach* (6th ed.). Maidenhead, UK: McGraw-Hill.

Rosenberg, J. (1997, November 5-7). Some misconceptions about lines of code. In *Proceedings of the 4th IEEE International Software Metrics Symposium*, Albuquerque, New Mexico (pp. 137-142).

Tokuda, L., & Batory, D. (2001). Evolving object-oriented designs with refactorings. *Automated Software Engineering, 8, 89-120.*

This work was previously published in Object-Oriented Design Knowledge: Principles, Heuristicsand Best Practices, edited by M. Piattini, pp. 250-281, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Chapter 8.16
Model-Driven Software Refactoring

Tom Mens
University of Mons-Hainaut, Belgium

Gabriele Taentzer
Philipps-Universität Marburg, Germany

Dirk Müller
Chemnitz University of Technology, Germany

ABSTRACT

In this chapter, we explore the emerging research domain of model-driven software refactoring. Program refactoring is a proven technique that aims at improving the quality of source code. Applying refactoring in a model-driven software engineering context raises many new challenges such as how to define, detect and improve model quality, how to preserve model behavior, and so on. Based on a concrete case study with a state-of-the-art model-driven software development tool, AndroMDA, we explore some of these challenges in more detail. We propose to resolve some of the encountered problems by relying on well-understood techniques of meta-modeling, model transformation and graph transformation.

INTRODUCTION

In the current research and practice on software engineering, there are two very important lines of research for which tool support is becoming widely available. The first line of research is *program refactoring*, the second one is *model-driven software engineering*. To this date, however, the links and potential synergies between these two lines of research have not been sufficiently explored. This will be the main contribution of this chapter.

Model-Driven Software Engineering

In the realm of software engineering, we are witnessing an increasing momentum towards the

use of models for developing software systems. This trend commonly referred to as model-driven software engineering, emphasizes on models as the primary artifacts in all phases of software development, from requirements analysis over system design to implementation, deployment, verification and validation. This uniform use of models promises to cope with the intrinsic complexity of software-intensive systems by raising the level of abstraction, and by hiding the *accidental complexity* of the underlying technology as much as possible (Brooks, 1995). The use of models thus opens up new possibilities for creating, analyzing, manipulating and formally reasoning about systems at a high level of abstraction.

To reap all the benefits of model-driven engineering, it is essential to install a sophisticated mechanism of *model transformation*, that enables a wide range of different automated activities such as translation of models (expressed in different modeling languages), generating code from models, model refinement, model synthesis or model extraction, model restructuring etc. To achieve this, languages, formalisms, techniques and tools that support model transformation are needed. More importantly, their impact on the quality and semantics of models needs to be better understood.

Program Refactoring

Refactoring is a well-known technique to improve the quality of software. Martin Fowler (1999) defines it as "A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior".

The research topic of refactoring has been studied extensively at the level of programs (i.e., source code). As a result, all major integrated software development environments provide some kind of automated support for program refactoring.

As a simple example of a program refactoring, consider the refactoring *Extract Method*, one of the more than 60 refactorings proposed by Fowler. Essentially, it is applied to a method in which part of the method body needs to be extracted into a new method that will be called by the original one. The situation before this program refactoring on a piece of Java source code is shown in Figure

Figure 1. Java source code example before applying the Extract Method program refactoring (©2007 Tom Mens, UMH. Used with permission)

```
    protected LectureVO[] handleFindLecture
      (java.lang.String title, domain.Weekday day, domain.Time time)
      throws java.lang.Exception
  *  { SearchCriteria c = new SearchCriteria();
  *    c.setDay(day);
  *    c.setTitle(title);
  *    c.setTime(time);
      Collection coll =
            getLectureDao().findLecture(LectureDao.TRANSFORM_
LECTUREVO,c);
      LectureVO[] lectures = new LectureVO[coll.size()];
```

Figure 2. Java example after applying the Extract Method refactoring (©2007 Tom Mens, UMH. Used with permission)

```
     protected LectureVO[] handleFindLecture
          (java.lang.String title, domain.Weekday day, domain.Time
time)
        throws java.lang.Exception
    *  { SearchCriteria c = this.initialise(title,day,time);
       Collection coll =
            getLectureDao().findLecture(LectureDao.TRANSFORM_
LECTUREVO,c);
       LectureVO[] lectures = new LectureVO[coll.size()];
       return (LectureVO[])coll.toArray(lectures); }
    * protected SearchCriteria initialise
    *      (java.lang.String title, domain.Weekday day, domain.Time
time)
    *   throws java.lang.Exception
    * { SearchCriteria c = new SearchCriteria();
    *   c.setDay(day);
```

1, the situation after is shown in Figure 2. The code lines that differ between both versions are marked with an asterisk.

For program refactoring, a wide variety of formalisms has been proposed to gain a deeper understanding, and to allow formal analysis. One of these formalisms is graph transformation theory (Mens *et al*., 2005). We mention it here explicitly, as we will show later in this chapter how this formalism can be applied to support model refactoring as well. It is, however, not our goal to provide a detailed overview of existing work on program refactoring here. For the interested reader, we refer to a detailed survey of the state-of-the-art in this domain (Mens & Tourwé, 2004).

Model-Driven Software Refactoring

A natural next step seems to explore how the idea of refactoring may be applied in a model-driven software development context. We will refer to this combination as *model-driven software refactoring* and we will explore the ramifications of this synergy in the current chapter.

One of the straightforward ways to address refactoring in a model-driven context is by raising refactorings to the level of models, thereby introducing the notion of *model refactoring*, which is a specific kind of model transformation that allows us to improve the structure of the model while preserving its quality characteristics. To the best of our knowledge, Sunyé *et al*. (2001) were the first to apply the idea of refactoring to models expressed in the Unified Modeling Language (UML).

A simple yet illustrative example of a UML model refactoring is shown in Figure 3. It depicts a class model in which two classes having attributes of the same type have been identified. The model refactoring consists of removing the redundancy

Figure 3. Example of a model refactoring on UML class diagrams (©2007 Tom Mens, UMH. Used with permission)

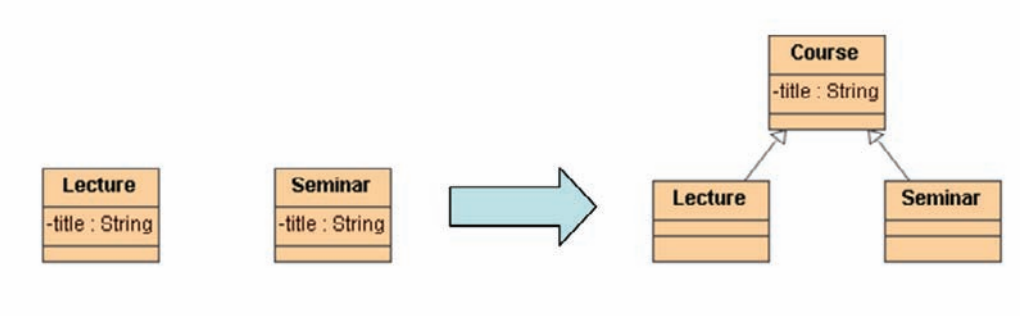
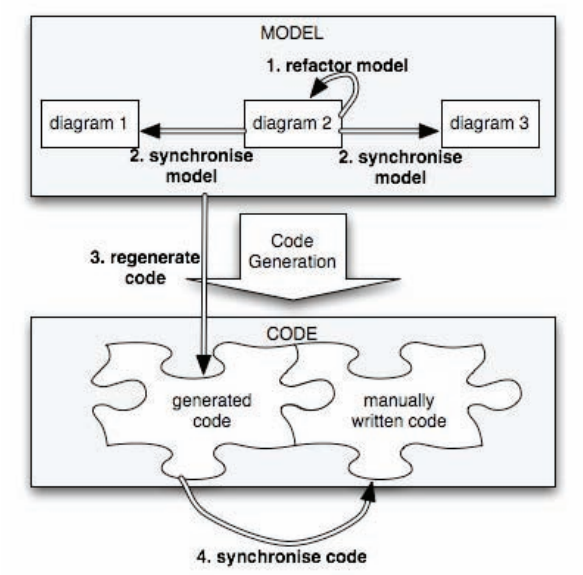


Figure 4. A scenario for model-driven software refactoring (©2007 Tom Mens, UMH. Used with permission)



by introducing an abstract super class of both classes, and moving up the attribute to this new super class.

The above example may look simple, but it should be seen in a more general context, which makes dealing with model refactorings considerable less trivial. Consider the scenario depicted in Figure 4. It clearly illustrates the potentially high impact a simple refactoring may have on the software system. We assume that a

model is built up from many different views, typically using a variety of different diagrammatic notations (e.g., class diagrams, state diagrams, use case diagrams, interaction diagrams, activity diagrams, and many more). We also assume that the model is used to generate code, while certain fragments of the code still need to be implemented manually. Whenever we make a change (in this case, a refactoring) to a single view or diagram in the model (step 1 in Figure 4), it is likely that we need to synchronize all related views, in order to avoid them becoming inconsistent (step 2 in Figure 4) (Grundy *et al.*, 1998). Next, since the model has been changed, part of the code will need to be regenerated (step 3 in Figure 4). Finally, the manually written code that depends on this generated code will need to be adapted as well (step 4 in Figure 4).

STATE-OF-THE-ART IN MODEL REFACTORING

At the level of models, research on refactoring is still in its infancy. Little research has been performed on model refactoring, and many open questions remain that are worthy of further investigation. For example, the relation between model refactoring and its effect on the model quality remains a largely unanswered question. From a practical point of view, only very few tools provide integrated support for model refactoring. Also, the types of models for which refactoring is supported is very limited.

In research literature, mainly UML models are considered as suitable candidates for model refactoring (Sunyé *et al.*, 2001; Astels, 2002; Boger *et al.*, 2002). In particular, refactoring of class models (e.g., UML class diagrams) has been investigated by various researchers. The advantage of such models is that they provide a representation that is relatively close to the way object-oriented programs are structured. As such, many of the refactorings known from object-oriented programming (Fowler, 1999) can be ported to UML class diagrams as well. For example, the refactoring shown in Figure 1 can also be considered as a class diagram refactoring, since a new method is created that will be visible in a class diagram. Of course, additional techniques are needed in order to ensure traceability and consistency between class diagrams and their corresponding source code when applying class diagram refactorings (Bouden, 2006).

When it comes to reasoning about the behavior preservation properties of class diagram refactorings, however, things become more difficult for various reasons. The main problem is that class diagrams provide an essentially structural description of the software architecture. Hence, behavioral information has to be expressed in a different way, either by resorting to OCL constraints, behavioral models (e.g., state diagrams or interaction diagrams), or by program code.

With respect to refactoring of behavioral models, not much work is available. We are only aware of a few approaches that address the problem of refactoring state diagrams, and try to prove their behavior preservation properties in a formal way. Van Kempen *et al.* (2005) use a formalism based on CSP to describe statechart refactorings, and show how this formalism can be used to verify that a refactoring effectively preserves behavior. Pretschner and Prenninger (2006) provide a formal approach for refactoring state machines based on logical predicates and tables. Integrating these ideas into tool support is left for future work. Apart from some limitations imposed by the formalisms used, a more general problem is that there is still no generally accepted formal semantics for (UML) state diagrams. Many different interpretations exist and, obviously, this has an important effect on how the behavior is formally defined.

Though research on model refactoring is still in its infancy, a number of formalisms have already been proposed to understand and explore model

refactoring. Most of these approaches suggest expressing model refactoring in a declarative way. Van Der Straeten *et al.* (2004) propose to use description logics; Van Der Straeten & D'Hondt (2006) suggest the use of a forward-chaining logic reasoning engine to support composite model refactorings. Gheyi *et al.* (2005) specify model refactorings using Alloy, a formal object-oriented modeling language. They use its formal verification system to specify and prove the soundness of the transformations. Biermann *et al.* (2006) and Mens *et al.* (2007) use graph transformation theory as an underlying foundation for specifying model refactoring, and rely on the formal properties to reason about and analyze these refactorings.

An important aspect of refactoring in a model-driven software development context that is typically neglected in research literature is how it interferes with code generation. Most contemporary tools for model-driven software development allow generating a substantial part of the source code automatically from the model, while other parts still need to be specified manually (see Figure 4). This introduces the need to synchronize between models and source code when either one of them changes. How such synchronization can be achieved in presence of automated refactoring support is a question that has not been addressed in detail in research literature. If a model is being refactored, how should the corresponding source code be modified accordingly? Vice versa, if source code is being refactored, how will the models be affected? These are the kind of questions that will be addressed in this chapter. To this extent, we will report on our experience with AndroMDA, a state-of-the-art tool for model-driven software development based on UML.

MOTIVATING EXAMPLE: MODEL-DRIVEN DEVELOPMENT WITH ANDROMDA

This section presents the model-driven development of a small web application for a simple university calendar. We will develop this calendar in two iteration steps using AndroMDA[1]. First the underlying data model is designed and a web application with a default web presentation is generated. Second, application-specific services and the web presentation are developed with AndroMDA. This means that use cases are defined and refined by activity diagrams that can use controllers and services. The development is not hundred percent model-driven, since service and controller bodies have to be coded by hand.

For both iteration steps, we first present the UML model using the AndroMDA profile and then discuss a refactoring step useful in that context.

Getting Started with Developing a University Calendar Using AndroMDA

One of the main tools for model-driven software development is AndroMDA. Its transformation engine is structured by cartridges. A number of pre-defined cartridges is already available realizing the generation of web applications from UML models. We illustrate model-driven software development based on AndroMDA by the example of a very simple university calendar.

In principle, the model-driven development process of AndroMDA is based on use cases. But in this initial example, we start with an even simpler way of using AndroMDA. We just design the underlying data model and AndroMDA generates a complete web application with a default web presentation from that.

A web application generated by AndroMDA has a three-tier architecture consisting of a service

layer building up on a data base, controllers using the services defined, and a web presentation. The underlying data model, services and controllers are defined by an UML class diagram. Additionally, visual object classes are modeled, which are used for presenting data to the user, decoupled from the internal data model.

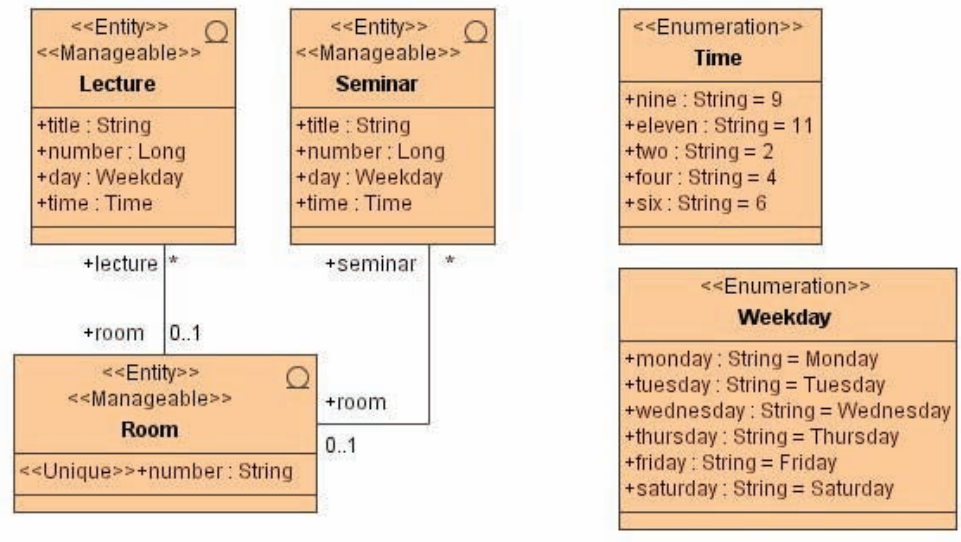
An example of an AndroMDA class diagram is shown in Figure 5. It depicts a simple data model for a university calendar. We can observe that the basic entities are `Rooms` that can be occupied for giving a Lecture or a Seminar. Based on this class diagram, AndroMDA can generate a default web interface for managing lectures, seminars and rooms. Users can add and delete instances, change attribute values and perform searches. The webpage for managing lectures is shown in Figure 6.

The UML profiles used in connection with AndroMDA can be considered as a domain-specific language, dedicated to the generation of web applications. This is achieved by giving a specific semantics to UML models by relying on that dedicated UML profiles. They extend the semantics of the UML by introducing specific stereotypes, to which additional constraints and tagged values are attached. For example, the stereotype «Entity» attached to a class is used to represent a data entity to be stored in a database. If, additionally, the «Manageable» stereotype is used, it causes AndroMDA to generate a default web presentation for managing the corresponding entities. The use of such manageable entities has been illustrated in Figure 5.

First Refactoring of the University Calendar

Due to their compactness, large parts of AndroMDA UML models are used for generating user interfaces. Thus, model refactorings in this context are likely to cause changes in user inter-

Figure 5. Data model for a simple university calendar (©2007 Tom Mens, UMH. Used with permission)



faces as well. Following Fowler (1999) in a strict sense, refactorings should not change the user interface of software, since they are supposed to "preserve the observable behavior". This strict interpretation of refactoring, however, makes little sense if applied in a model-driven software development context, due to the side-effects that model refactorings may cause on the generated code, especially user interfaces. Thus, Fowler's definition of refactoring should be interpreted in a more liberal way, in the sense that it should not change the functionality offered by software. Modifications to the *usability* of the software or to other non-functional properties (such as interoperability, portability, reusability, adaptability and the like) should be allowed, if the goal of these modifications is to improve the software quality.

In the remainder of this section we will show a concrete refactoring on our university calendar case study to clarify what refactoring can mean in the context of model-driven development.

Since «Entity» classes Lecture and Seminar contain several attributes in common (see Figure 5), it would make sense to refactor this data model by adding a new abstract superclass, called Course, and pulling up all common attributes to this new class. The result of this refactoring is shown in Figure 7.

Note that tagged value @andromda.hibernate.inheritance has to be set to interface for restricting the management facilities for courses to searching functionalities only.

When regenerating a web application from the refactored data model in Figure 7, most of the user interface remains unaltered. But a new webpage

Figure 6. Webpage for managing lectures (©2007 Tom Mens, UMH. Used with permission)

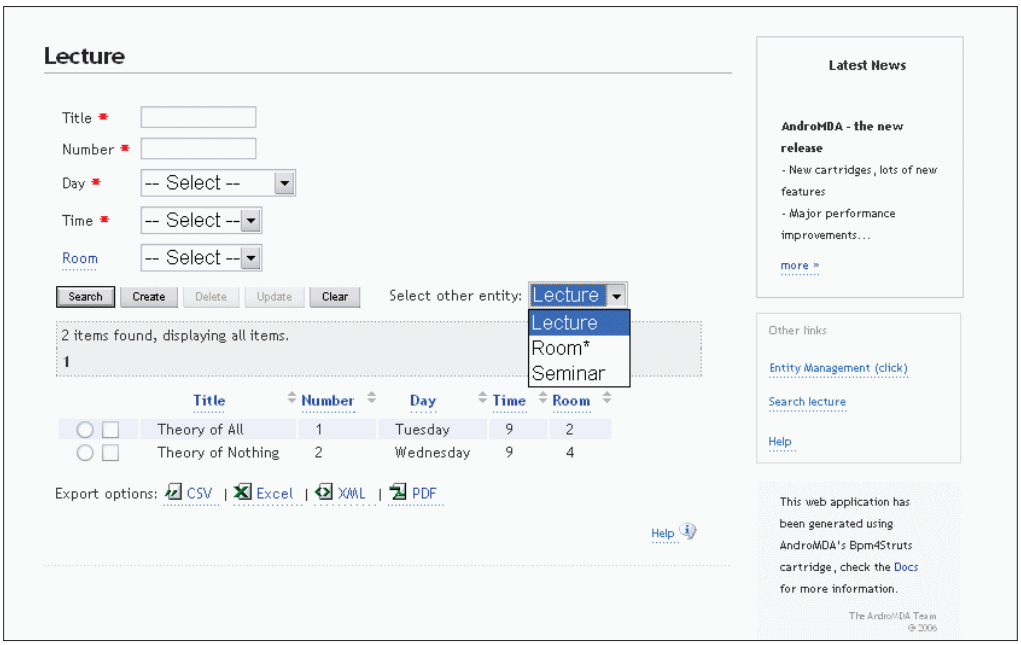
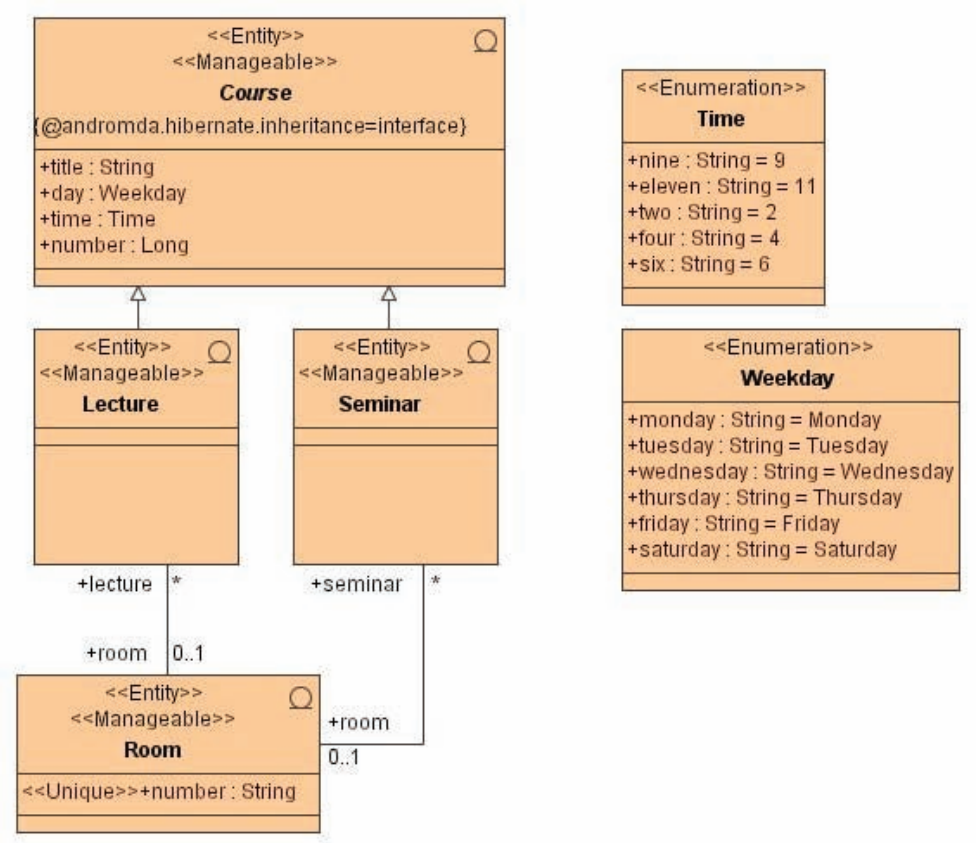


Figure 7. Data model for a simple university calendar after having applied the Pull Up Attribute refactoring (©2007 Tom Mens, UMH. Used with permission)



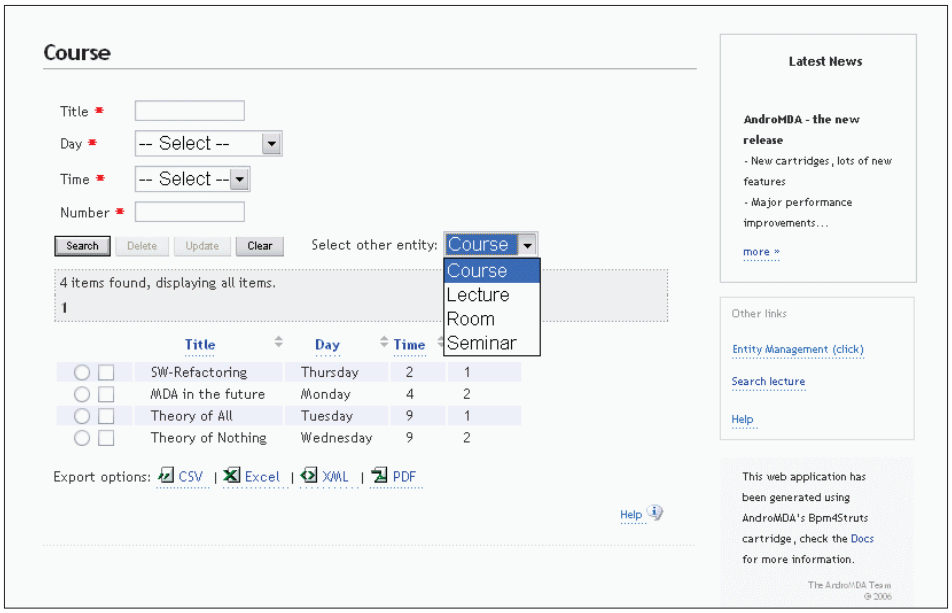
will appear for managing courses, as shown in Figure 8. Because of the tagged value attached to Course, this webpage only offers search functionality, but does not allow the addition or deletion of course instances.

In the example explained in section 3.1, all application code is generated from the model. Thus, refactoring the model alone appears to be sufficient to refactor the whole software application. However, it should be noted that, due to the refactoring applied to the model, the behavior has been changed slightly, since AndroMDA has generated a new kind of webpage.

Developing Application-Specific Use Cases with AndroMDA

In this section, we will consider additional stereotypes and tagged values in the AndroMDA UML profile, but only as far as we need them to

Figure 8. Webpage for managing courses (©2007 Tom Mens, UMH. Used with permission)



develop our example. For a complete overview of all available stereotypes and how to use them we refer to the AndroMDA website.

«Service» is a class stereotype used to specify application-specific services. These services typically use one or more entities that store the data used by the services. For the model-driven development of a web presentation, we extend the model by use cases that are refined by activity diagrams. This model part describes the web presentation and its usage of controllers based on services. The development is not hundred percent model-driven, since service and controller bodies have to be coded by hand.

To illustrate the development of specific web applications we reconsider the university calendar and develop a specific use case diagram for lectures (see Figure 9). Use case Search lectures has two stereotypes being «FrontEndUseCase», which determines the use case to be visible to the user in form of a webpage, and «FrontEndApplication», which defines this use case to be the starting one.

Use case Search lectures is refined by an activity diagram that supports a search activity and the presentation of filtered lectures (see Figure 10). Activity Search lectures is an internal activity that calls the controller method showLectures(). Activity Present lectures has stereotype «FrontEndView» implying that this activity models a webpage. Both activities are connected by two transitions arranged in a cyclic way. After calling method showLectures() the result is transferred to the webpage by signal show, which has the resulting value object array as parameter. Signal search and its parameters

Figure 9. Example of a use case model in AndroMDA (©2007 Tom Mens, UMH. Used with permission)

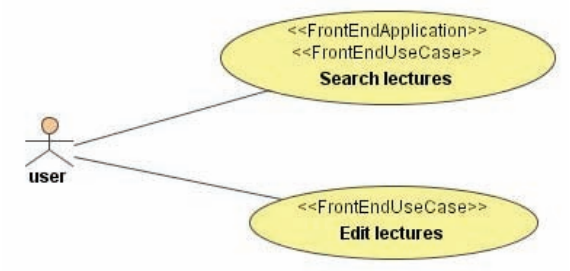
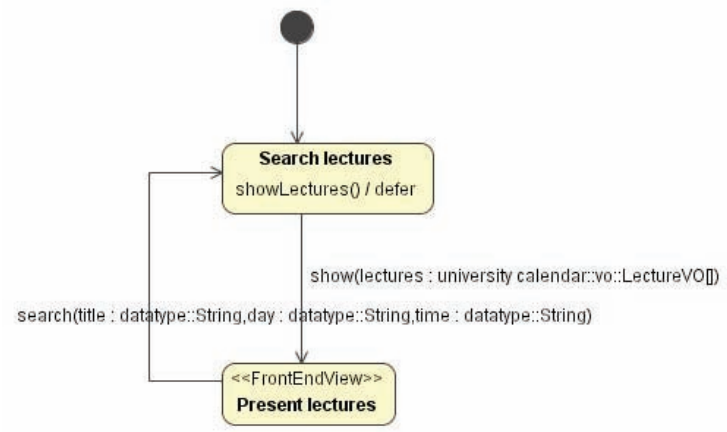


Figure 10. Example of an activity diagram specifying the Search lectures use case (©2007 Tom Mens, UMH. Used with permission)



are used to model the web form for filtering the lectures.

The class model in Figure 5 is again used as data model. To show lectures, a special value object class for lectures is used, which is specified by stereotype «ValueObject» (see Figure 11). This makes sense in terms of encapsulation (think of security, extensibility, etc.) and corresponds to the layered model-view-controller approach. Necessary information of the business layer is packaged into so-called "value objects", which are used for the transfer to the presentation layer.

Figure 11. Value Object and Criteria classes (©2007 Tom Mens, UMH. Used with permission)

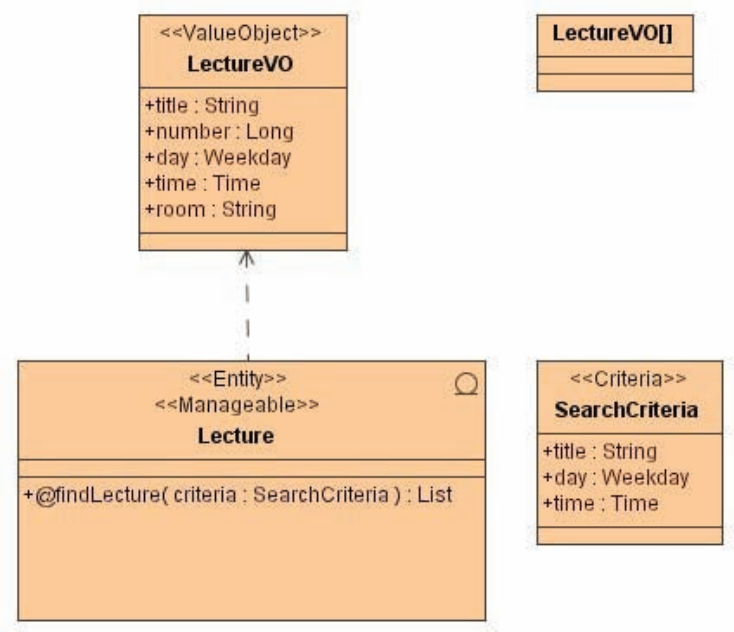
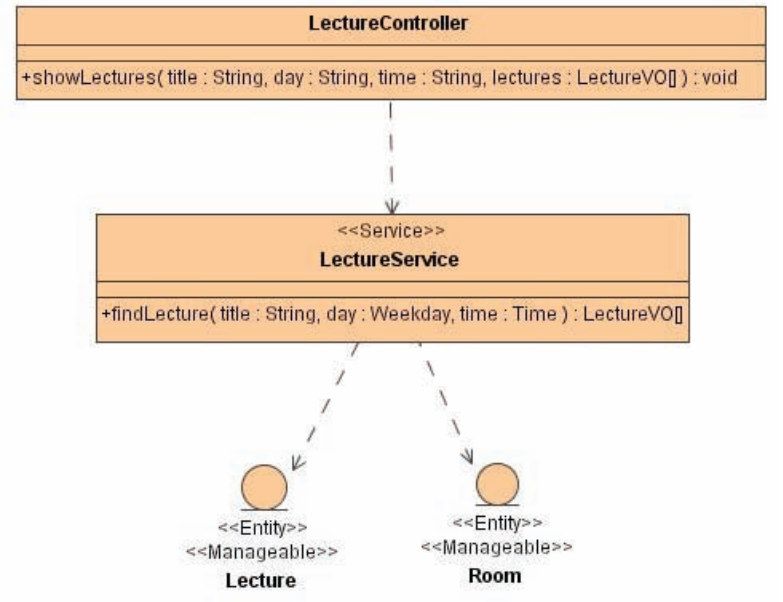


Figure 12. Service and controller classes (©2007 Tom Mens, UMH. Used with permission)



Passing real entity objects to the client may pose a security risk. Do you want the client application to have access to the salary information inside the Lecturer entity?

An attribute room of type String was added to LectureVO in order to allow a connection to the unique number of the Room class. Since value objects are used at the presentation layer, the types used are primitive ones; entity types are not used in that layer. A dependency relation between an entity and a value object is used to generate translation methods from the entity to its corresponding value object. Moreover, search criteria can be defined by a class of stereotype «Criteria».

Method showLectures() that is called from activity Search lectures in Figure 10, is defined in LectureController, a class that relies on class LectureService. This class is stereotyped as «Service» and relies on entities Lecture and Room (see Figure 12). However, the bodies of service and controller methods cannot be modeled, but have to be coded directly by hand. For example, the implementation of service method findLecture()is shown in Figure 1. Because of special naming conventions of AndroMDA it has to be named handleFindLecture().

The web application generated by AndroMDA from the complete model given in the previous figures (together with manually written code parts) produces the webpage shown in Figure 13. Please note that the names used as page title, in the search form and for the buttons are generated from the model.

Figure 13. Webpage for searching lectures (©2007 Tom Mens, UMH. Used with permission)

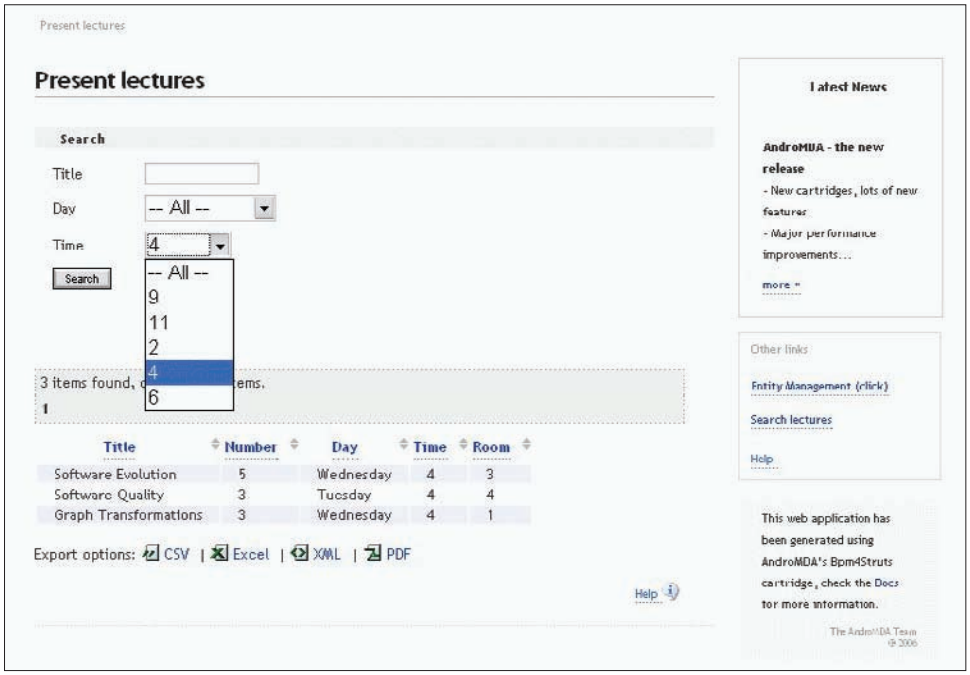


Figure 14. Value Object and Entity classes after renaming (©2007 Tom Mens, UMH. Used with permission)

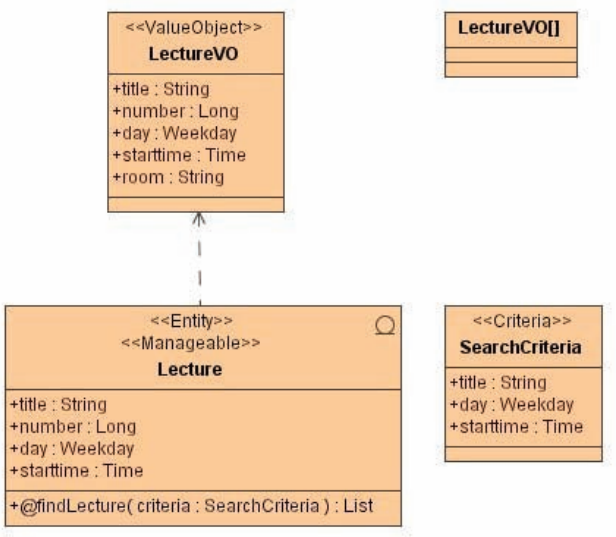
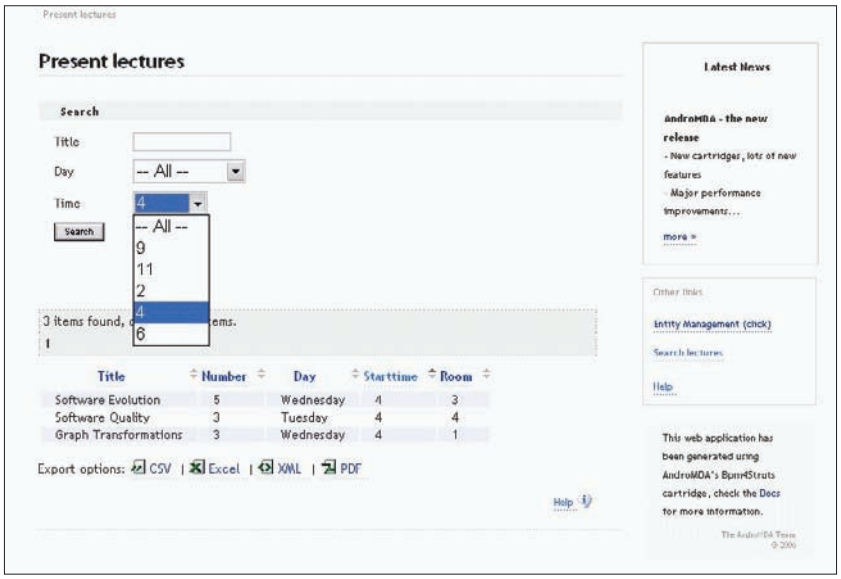


Figure 15. Webpage for searching lectures after renaming (©2007 Tom Mens, UMH. Used with permission)



Further Refactoring of the University Calendar

As a second model refactoring[2], we will discuss the renaming of attribute time to startime based on the model given in Section 3.3. We will argue that this refactoring affects the usability of the generated software. The refactoring is primarily performed on entity class Lecture, but since there is a value object class LectureVO for that entity, the corresponding value class attribute time has to be renamed into starttime, too (see Figure 14). The same is true in SearchCriteria. Thus, the standard refactoring method *Rename Attribute* becomes domain-specific and affects several classes in this domain-specific context.

Since the value object attribute is not used directly in other parts of the model, the model does not have to be updated any further. But the hand-written code (given in Figure 1) is affected, since accessor method setTime() is no longer available after regenerating the code. Thus, it has to be renamed as well, by calling method setStarttime() instead. After having performed this refactoring, the webpage for searching lectures has been changed slightly. As a result, the usability is affected, though not dramatically, since the column named "Time" of the presented table presented has changed into "Starttime" (see Figure 15).

Based on the analysis of both model refactorings carried out in this section, we can derive the following important preliminary conclusions:

- Generic model refactorings need to be adapted and refined in order to work properly in a domain-specific modeling language.
- Model refactorings may also affect, and require changes to the hand-written source code.
- Model refactorings may change external qualities as perceived by the user, such as usability aspects.

CHALLENGES IN MODEL-DRIVEN SOFTWARE REFACTORING

In this section, we will discuss some important challenges in model refactoring that have to do with the relation between model refactoring and model quality. It is not our ambition to solve all these challenges in the current chapter. In Sections 5 and 6 we will therefore only focus on those challenges that we consider being most urgent and most important and we will exemplify our proposed solution using the case study introduced in the previous section.

Model Quality

A first challenge is to provide a precise definition of model quality. A model can have many different non-functional properties or quality characteristics that may be desirable (some examples are: usability, readability, performance and adaptability). It remains an open challenge to identify which qualities are necessary and sufficient for which type of stakeholder, as well as how to specify these qualities formally, and how to relate them to one another.

Since the main goal of refactoring is to improve certain aspects of the software quality, we need means to assess this quality at the model level in an objective way. On the one hand, this will allow software modelers to identify which parts of the model contain symptoms of poor quality, and are hence potential candidates for model refactoring. On the other hand, quality assessment techniques can be used to verify to which extent model refactorings actually improve the model quality.

One of the ways to assess model quality is by resorting to what we will call *model smells*. These are the model-level equivalent of *bad smells*, a term originally coined by Kent Beck in (Fowler, 1999) to refer to structures in the code that suggest opportunities for refactoring. Typical model

smells have to do with redundancies, ambiguities, inconsistencies, incompleteness, non-adherence to design conventions or standards, abuse of the modeling notation, and so on. A challenge here is to come up with a comprehensive and commonly accepted list of model smells, as well as tool support to detect such smells in an automated way. What is also needed is a good understanding of the relation between model smells and model refactoring, in order to be able to suggest, for any given model smell, appropriate model refactorings that can remove this smell.

A second way to assess and control model quality is by resorting to *model metrics*. In analogy with software metrics (Fenton & Pfleeger, 1997) they are used to measure and quantify desirable aspects of models. It remains an open question, however, how to define model metrics in such a way that they correlate well with external model quality characteristics. Another important issue is to explore the relation between model metrics and model refactoring, and in particular to assess to which extent model refactorings affect metric values. These issues have been addressed by (Demeyer *et al.*, 2000; Du Bois, 2006; Tahvildari & Kontogiannis, 2004) though mainly at code level.

A final way to improve model quality is by introducing design patterns, which are proven solutions to recurring problems (Gamma *et al.*, 1994). At code level, Kerievsky (2004) explored the relation between refactorings and design patterns. It remains to be seen how similar results may be achieved at the level of models.

Kamthan (2004) provided a quality framework for UML models. It systematically studies the quality goals, how to assess them, as well as techniques for improving the quality, similar to the ones discussed above.

Model Synchronization

With respect to model refactoring, one of the key questions is how it actually differs from program refactoring. Can the same ideas, techniques and even tools used for program refactoring be ported to the level of models? If not, what is it precisely that makes them different?

One answer to this question is that models are typically built up from different views, using different types of diagrams, that all need to be kept consistent. This in contrast to programs, that are often (though not always) expressed within a single programming language.[3]

Perhaps a more important difference is that models are abstract artifacts whose main purpose is to facilitate software development by generating a large portion of the source code that would otherwise need to be written manually. However, full code generation is unfeasible in practice for most application domains. The additional challenge therefore consists in the need to synchronize and maintain consistency between models and their corresponding program code, especially when part of this program code has been specified or modified manually. In the context of model transformation, this implies that automated model refactorings (or other transformations) may need to be supplemented with code-level transformations in order to ensure overall consistency. Vice versa, program refactorings may need to be supplemented with model-level transformations to ensure their consistency.

Though no general solutions exist yet, the problem of model synchronization and model consistency maintenance is well known in literature. For example, (Van Gorp *et al.*, 2003) discuss the problem of keeping the UML models consistent with their corresponding program code. (Correa & Werner, 2004) explain how OCL constraints need to be kept in sync when the class diagrams are refactored and vice versa. Egyed (2006) proposes an incremental approach to model consistency checking that scales up to large industrial models. Liu *et al.* (2002) and Van Der Straeten & D'Hondt (2006) rely on a rule-based approach for detecting and resolving UML model inconsistencies, respectively. Van Der Straeten *et*

al. (2003) bear on the formalism of description logics to achieve the same goal. Mens *et al.* (2006) propose to resolve inconsistencies in an incremental fashion by relying on the formalism of graph transformation. Grundy *et al.* (1998) report on how tool support can be provided for managing inconsistencies in a software system composed of multiple views. Goedicke *et al.* (1999) address the same problem by relying on the formalism of distributed graph transformation.

Behavior Preservation

Another important challenge of model refactoring has to do with *behavior preservation*. By definition, a model refactoring is supposed to preserve the observable behavior of the model it is transforming. In order to achieve this, we need a precise definition of "behavior" in general, and for models in particular. In addition, we need formalisms that allow us to specify behavioral invariants, i.e., properties that need to be preserved by the refactoring. The formalism should then verify which of these invariants are preserved by the model refactoring. Although formal research on behavior preservation is still in its infancy, in Section 2 we already pointed to a few approaches that carried out initial research in this direction. Another approach that is worthwhile mentioning is the work by Gheyi *et al.* (2005). They suggest specifying model refactorings in *Alloy*, an object-oriented modeling language used for formal specification. It can be used to prove semantics-preserving properties of model refactorings.

A more pragmatic way to ensure that the behavior remains preserved by a refactoring is by resorting to *testing* techniques. Many researchers have looked at how to combine the ideas of testing with model-driven engineering (Brottier *et al.*, 2006; Mottu *et al.*, 2006). Test-driven development is suggested by the agile methods community as good practice for writing high-quality software. In combination with refactoring, it implies that before and after each refactoring step, tests are executed to ensure that the behavior remains unaltered.

Domain-Specific Modeling

A final challenge is the need to define model refactorings in domain-specific extensions of the UML (such as AndroMDA), or even in dedicated domain-specific modeling languages. These refactorings should be expressible in a generic yet customizable way. Indeed, given the large number of very diverse domain-specific languages, it is not feasible, nor desirable, to develop dedicated tools for all of them from scratch.

Zhang *et al.* (2004) therefore proposed a generic model transformation engine and used it to specify refactorings for domain-specific models. Their tool is implemented in the Generic Modeling Environment (GME), a UML-based meta-modeling environment. A model refactoring browser has been implemented as a GME plug-in. Their tool enables the automation and user-defined customization of model refactorings using ECL (Embedded Constraint Language), an extension of the declarative OCL language with imperative constructs to support model transformation. As an example of the expressiveness of their approach, they illustrated how it can be applied to class diagrams, state diagrams and Petri nets. The solution that we will explore later in this chapter is related, in the sense that we will propose a generic approach for UML-based model refactoring based on graph transformation concepts.

In general, the main challenge remains to determine, for a given domain-specific modeling language, which transformations can be considered as meaningful refactorings. On the one hand, they will need to preserve some notion of "behavior" and, on the other hand, they need to improve some quality aspect. These notions of behavior and quality can differ widely depending on the domain under study. For domains that do not refer to software (e.g., business domains, technical domains, etc.) it is much harder to come

to a meaningful definition of behavior, implying that the notion of refactoring would become much harder to define in that context.

Analyzing Model Refactorings

Even more advanced support for model refactorings can be envisaged if we have a precise means to analyze and understand the relationships between refactorings. This will enable us to build up complex refactorings from simpler ones; to detect whether refactorings are mutually exclusive, in the sense that they are not jointly applicable and to analyze causal dependencies between refactorings. These techniques have been explored in detail by Mens *et al.* (2007), and promise to offer more guidance to the developer on what is the most appropriate refactoring to apply in which context. A short introduction to this line of research will be given in Section 6.

MOTIVATING EXAMPLE REVISITED

In Section 3 two concrete model refactorings have been applied to AndroMDA models: pulling up an attribute into a new superclass and renaming an entity. In this section, we explore some more refactorings for AndroMDA models.[4] We start by considering a set of "standard" model refactorings widely used to restructure class diagrams. As it will turn out, most of these refactorings have side-effects due to constraints imposed by AndroMDA's code generator. Therefore, these model refactorings need to be customized to take into account more domain-specific information. Next to these "standard" refactorings, we will also discuss entirely new "domain-specific" refactorings for AndroMDA models.

In the following, we will take a slightly broader view, and we discuss three categories of model transformations as follows:

(1) model refactorings that do not affect the user interface at all;
(2) model refactorings that do affect the user interface with respect to the usability, but that do not affect what the user can do with the application;
(3) model transformations that also affect the actual behavior/functionality of the application.

The latter category does not contain refactorings in the strict sense of the word, but it is nevertheless useful and necessary to deal with them. For example, it could be the case that what is perceived as a normal refactoring will actually *extend* the behavior as a side effect of the code generation process.

Pull Up Attribute

When pulling up an attribute to a super class, as explained in Section 3.2, the code generator will generate a new webpage corresponding to this super class, with search functionality for each manageable entity. Thus, this model transformation belongs to category (3).

Rename

The refactoring example in Section 3.4 is concerned with renaming an attribute of an entity class. This refactoring affects the user interface, if the entity is manageable. In this case, one of the columns in the table of the webpage has been renamed. Furthermore, in case that the entity class comes along with a value object class that is derived from the entity class, a renaming of an entity attribute has to be accompanied by a renaming of the corresponding attribute in its value object class. If, in addition, this value object attribute is used in some activity diagram, the name has to be adapted there as well. Furthermore, this value object attribute can occur in hand-written code,

which implies that renaming has to be performed also in that part of the code.

A similar situation would arise if we renamed the entity class itself, as it would be reflected by a change in the title of the corresponding webpage for manageable entities. In case that the renamed entity class comes along with a value object class whose name is derived from the entity class name (e.g., in Figure 14, "LectureVO" is derived from "Lecture" by suffixing "VO"), renaming has to be accompanied by a renaming of its corresponding value object class. Furthermore, the renaming has to be propagated as discussed for attributes. In all cases presented, although the user interface changes slightly, the functionality of the application is not affected. Hence, these refactorings belong to category (2).

Similar to entities, use cases can be renamed as well. This might have an effect on activity diagrams, since AndroMDA supports the connection of several activity diagrams via use case names. For example, an end activity of one activity diagram may be named as a use case, which means that the control flow would continue at the start activity of the corresponding activity diagram. In the generated web applications, use cases are listed on the right-hand side of each webpage. Again, a renamed use case would change the usability of the web application, but not its functionality, so the refactoring belongs to category (2).

In summary, we see that renaming in AndroMDA may have a high impact. Due to the fact that the code generator automatically produces new types of elements based on the names of existing elements, a seemingly simple change (in casu renaming) will propagate to many different places. A tool that would implement this model refactoring would therefore need to take these issues into account to ensure that the renaming does not lead to an inconsistent model or code. Furthermore, because the changes affect hand-written code, the refactoring may require a certain amount of user interaction.

Create Value Object

A domain-specific refactoring for AndroMDA models is the creation of value objects for entities. An example is visually represented in Figure 16. Given a class with stereotype «Entity» (for example, class Lecture), a new class with stereotype «Value Object» is created and the entity class becomes dependent on this new class. The value object class is named after its entity class followed by suffix "VO" (for example, value object class LectureVO). The entity attributes are copied to the value object class, keeping names and types, by default. If internal information should be hidden from the client, the corresponding attribute would not be copied. This refactoring belongs to category (1) and does not affect any other part of the model, since the value object class is only created without being used yet.

Merge Services

Another domain-specific model refactoring is *Merge Services*. It takes two «Service» classes and merges them as well as all their incoming and outgoing dependencies. Consider the following example where both a LecturService and RoomService exist (see Figure 17). If we do not consider remote services and have only one controller class, it does not make sense to have two service classes. Therefore, both should be merged into LectureService. After refactoring, the controller class will have only one outgoing dependency. As a result, the hand-written code for the controller method will be affected. Nevertheless, this restructuring will not modify the external behavior, so users of the generated web application will not notice any change. Hence, this refactoring falls into category (1).

Split Activity

The front-end of a web application is modeled by use cases and activity diagrams. A refactoring

Figure 16. Example of the domain-specific model refactoring CreateValueObject (©2007 Tom Mens, UMH. Used with permission)

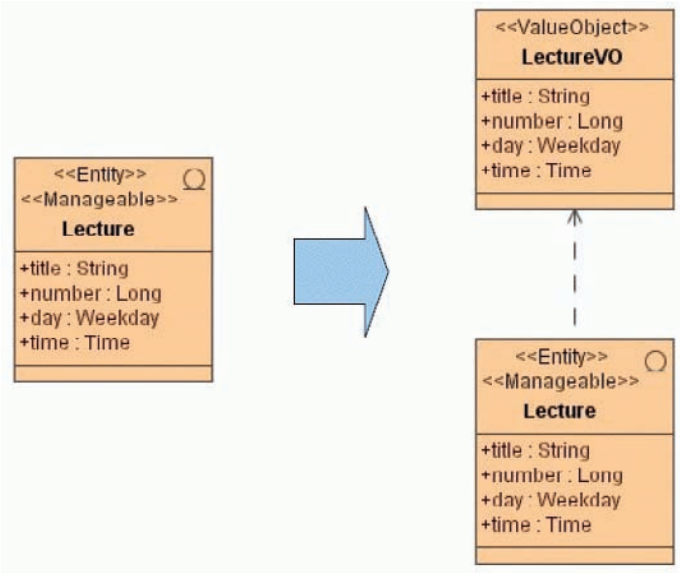
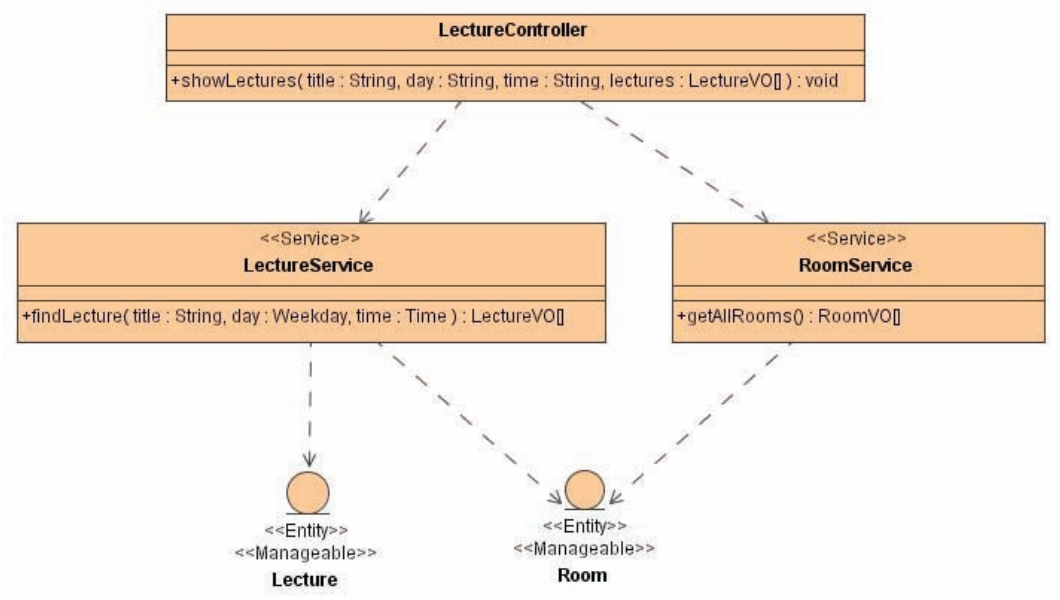


Figure 17. Service classes LectureService and RoomService with dependencies (©2007 Tom Mens, UMH. Used with permission)



like the splitting of activities into two consecutive ones, linked by a transition, can directly affect the web presentation. If the original activity was a «FrontEndView», the corresponding webpage is split into two pages. If an internal activity was split, this refactoring has to be accompanied by a splitting of the corresponding controller method called. In the first case, the refactoring belongs to category (2), in the second case it belongs to category (1).

Extract Method

Extract Method is a refactoring from the standard catalogue established by Fowler. In the context of model-driven development, and AndroMDA in particular, it can have new effects. Consider the scenario in Figure 19. First, we perform the extract method refactoring to the hand-written code, as illustrated in Figure 2 where a method, called initialise(), is extracted from a given service method handleFindLecture. To reflect this change at model level, we modify the class diagram by adding the extracted method to the class LectureService as well (see Figure 18). Consequently, the code generator will generate extra code for this method, which requires the manually written code to be adapted to make it consistent again. In particular, method initialise()needs to be renamed into handleInitialise(), because this is the convention used by the code generator: all service methods need to be prefixed with "handle" at source code level. We can use this knowledge to constrain the Extract Method refactoring to make it domain-specific: When extracting a method, the name that the user needs to provide for the extracted method needs to follow the naming conventions imposed by the code generator. Not doing so will cause the precondition of the refactoring to fail.

The above scenario is generalized and visualized in Figure 19. It shows how a refactoring at source code level (step 1) may require

Figure 18. Changes to the class diagram as a result of applying the Extract Method program refactoring (see Figure 2) (©2007 Tom Mens, UMH. Used with permission)

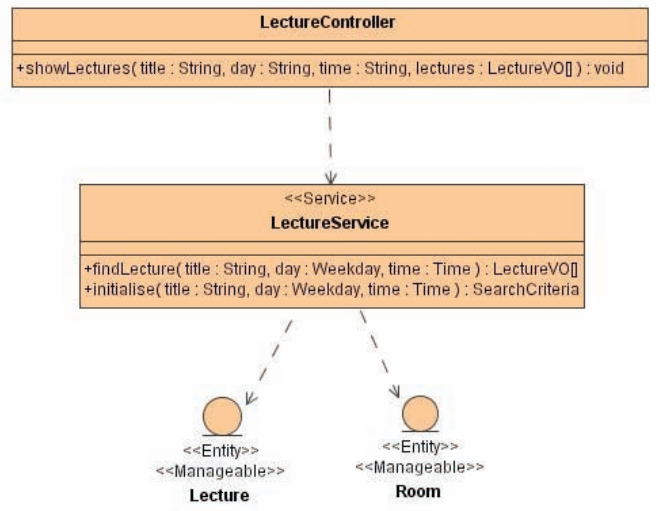
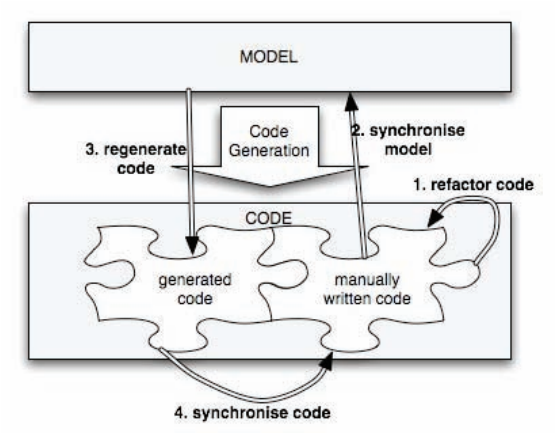


Figure 19. Another scenario of model-driven software refactoring, initiated by a refactoring of the hand-written source code (©2007 Tom Mens, UMH. Used with permission)



synchronization of the corresponding model (step 2) which, after regenerating the code (step 3) involves another modification to the hand-written part of the code (step 4). The last step is not needed, if the user obeys the naming convention for the new method as discussed above.

SPECIFYING AND ANALYZING MODEL REFACTORINGS

In Section 5, the important challenge of domain-independent support for model refactoring was discussed. A possible formalism that can be used to specify and also analyze refactorings is the theory of *graph transformation* (Ehrig *et al.* 2006). Compared to other approaches it has a number of advantages: it allows one to specify program refactorings and model refactorings for various languages in a uniform and generic way, by representing the software artifact under consideration as a graph, and by specifying the refactorings as graph transformation rules. In addition, one can benefit from the formal properties of graph transformation theory to reason about refactoring in a formal way. For example, properties such as termination, composition, parallel dependencies, and sequential dependencies can be analyzed.

Since the Eclipse Modeling Framework (EMF) has become a key reference for model specification in the world of model-driven development, we rely our approach to model refactoring on EMF model transformation. This approach is presented in Section 6.1. To perform a formal analysis of EMF transformations we translate them to graph transformations, which is possible under certain circumstances. In Section 6.2, a conflict and dependency analysis of model refactorings is presented, assuming that the model refactorings are defined by graph transformation rules.

Technical Solution

From a technical point of view, we will discuss how to implement and execute model refactorings.

In particular, we will consider how to realize model refactoring within the Eclipse Modeling Framework (EMF). As a prerequisite, a specification of the underlying modeling language is needed, which will be given by a meta-model. Figure 20 shows an EMF model that represents a simplified extract of the AndroMDA meta-model. Figure 21 shows an instance of this EMF model for the entity class Lecture of the simple university calendar.

Biermann *et al.* (2006) explain in detail how EMF model refactoring can be expressed by EMF model transformation. This kind of model transformation is specified by rules and is performed in-place, i.e., the current model is directly changed and not copied. Each transformation rule consists of a left-hand side (LHS), indicating the preconditions of the transformation, a right-hand side (RHS), formulating the post conditions of the transformations, and optional negative application conditions (NAC), defining forbidden structures that prevent application of the transformation rule. Objects that are checked as precondition preserved during a transformation are indicated by colors. Object nodes of the same color present one and the same object in different parts of a rule. While attributes in the LHS may have constant values or rule variables only, they are allowed to carry Java expressions in the RHS, too. The same variable at different places in the rules means the same value at all places. In the following, we use this approach to EMF model transformation for specifying UML model refactorings.

In Figure 22 and Figure 23, two model transformation rules are shown, which both are needed to perform refactoring *Create Value Object* explained in Figure 16 of Section 5. Rule *CreateValueObjectClass* is applied once, creating a new value object class and a dependency of the entity class on this new class. A class model with an entity class is needed to create a value object class and a dependency in between. The name of

Figure 20. Extract of AndroMDA meta-model as EMF model (©2007 Tom Mens, UMH. Used with permission)

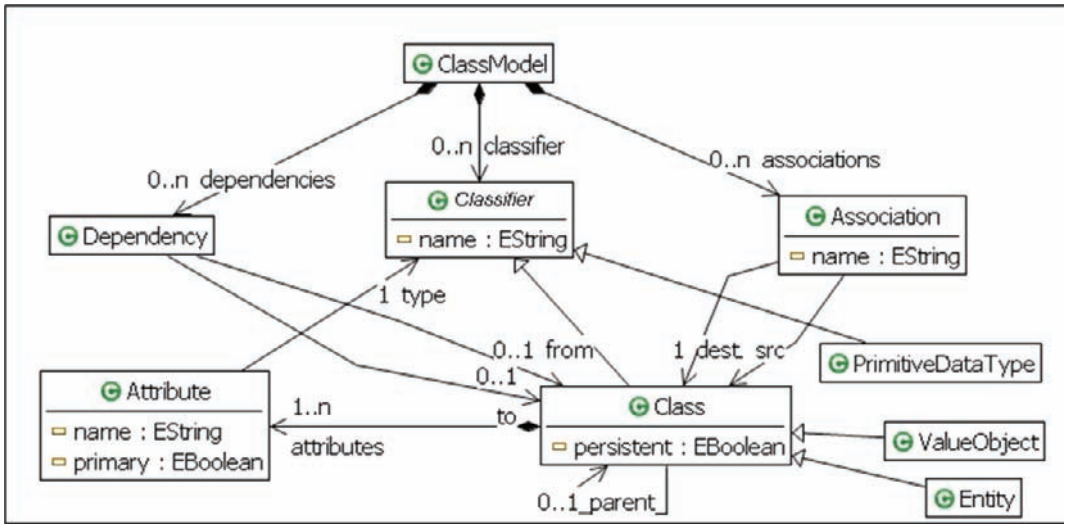


Figure 21. Entity class Lecture *with attributes in abstract syntax as EMF model instance (©2007 Tom Mens, UMH. Used with permission)*

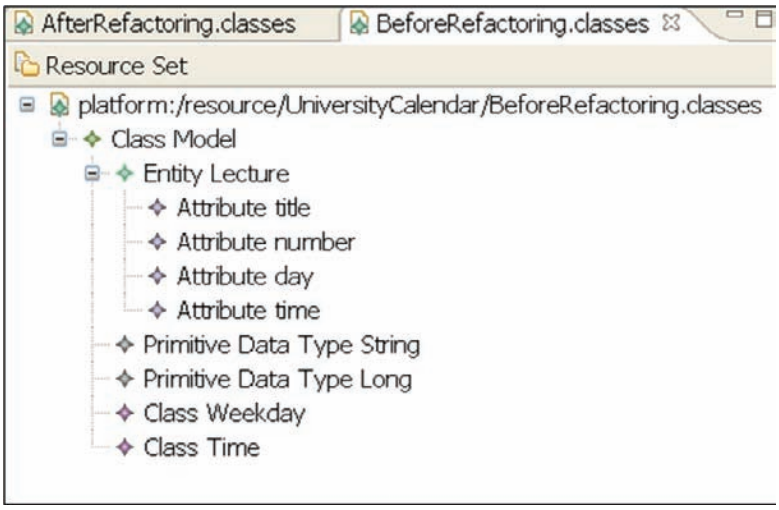


Figure 22. EMF model transformation rule CreateValueObjectClass for refactoring method Create Value Object (©2007 Tom Mens, UMH. Used with permission)

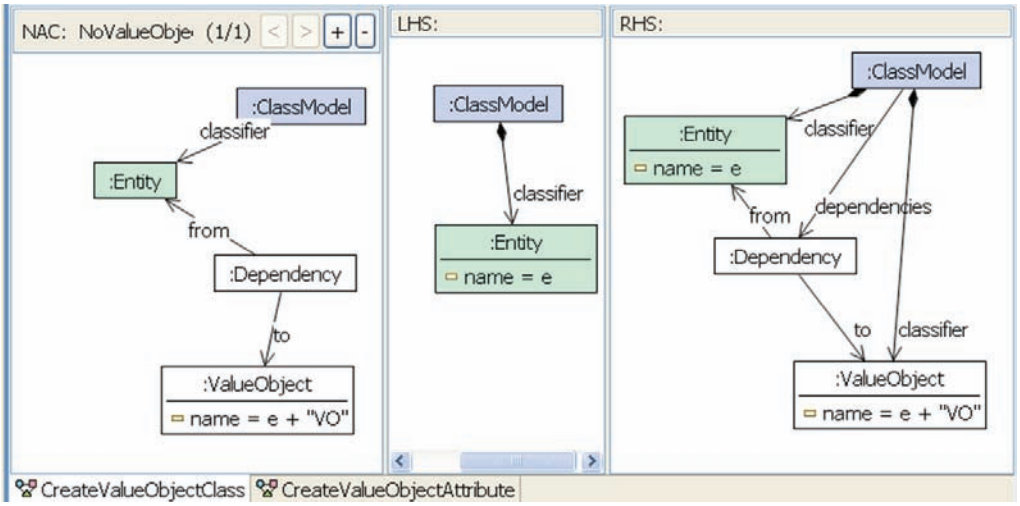


Figure 23. EMF model transformation rule CreateValueObjectAttribute for refactoring method Create Value Object (©2007 Tom Mens, UMH. Used with permission)

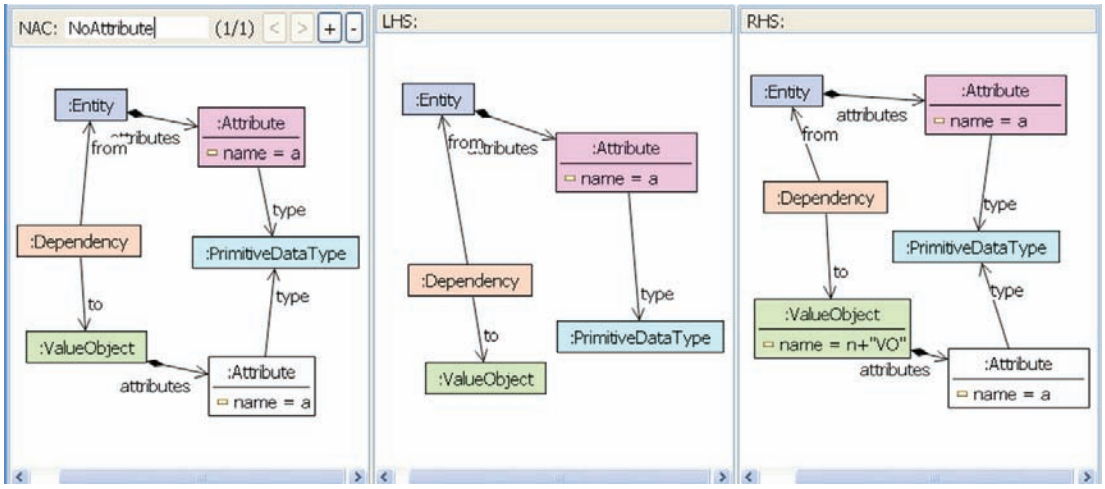
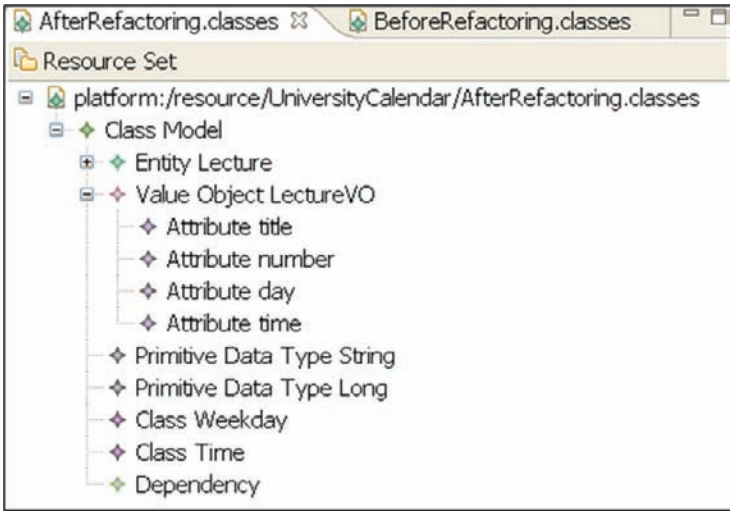


Figure 24. Entity class Lecture with value object class LectureVO in abstract syntax as EMF model instance (©2007 Tom Mens, UMH. Used with permission)



this new value object class is constructed by taking the entity class name e and adding suffix "VO". This rule is applied only if a value object class of this name has not already been created.

Thereafter, rule *CreateValueObjectAttribute* is applied for each of the attributes of the entity class that should occur also in the value object class. Each time it is applied, it copies an attribute that has not yet been copied into the value object.

Applying rule *CreateValueObjectClass* once and rule *CreateValueObjectAttribute* as often as entity class Lecture has attributes (i.e., four times in this case) to the EMF model instance in Figure 21, we obtain the EMF model instance in Figure 24.

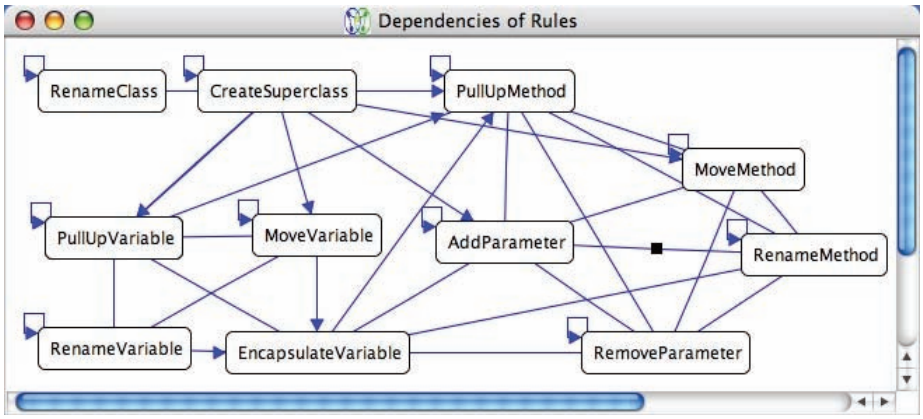
To open up the possibility for analyzing EMF model refactorings, we translate them to graph transformations. In this way, the formal analysis for graph transformation becomes available for EMF model refactoring. Although EMF models show a graph-like structure and can be transformed similarly to graphs, there is an important difference between both. In contrast to graphs, EMF models have a distinguished tree structure that is defined by the containment relation between their classes. Each class can be contained in at most one other class. Since an EMF model may have non-containment references in addition, the following question arises: What if a class, which is transitively contained in a root class, has non-containment references to other classes not transitively contained in some root class? In this case we consider the EMF model to be inconsistent.

A transformation can invalidate an EMF model, if its rule deletes one or more objects. To ensure consistent transformations only, rules that delete objects or containment links or redirect them, have to be equipped with additional NACs.

Formal Solution

As an illustration of how refactoring dependency analysis may increase the understanding of refactoring, consider the following scenario. Assume that a software developer wants to know which refactoring rules need to be applied in order to restructure a software system. Typically, many different refactoring rules may be applicable, and it is not easy to find out what would be the most

Figure 25. Sequential dependencies computed by AGG for a representative set of refactorings implemented as graph transformations (©2007 Tom Mens, UMH. Used with permission)



optimal way to apply these rules. Joint application of some refactoring rules may not be possible due to parallel dependencies between them, and some refactoring rules may sequentially depend on other ones. Graph transformation theory allows us to compute such dependencies by relying on the idea of *critical pair analysis.* The general-purpose graph transformation tool AGG[5] provides an algorithm implementing this analysis.

Figure 25 gives an example of all sequential dependencies that have been computed between a representative, yet simplified, subset of refactorings expressed as graph transformation rules. For example, we see that there is a sequential dependency between the *CreateSuperclass* refactoring and the *PullUpVariable* refactoring. *CreateSuperclass* inserts a new intermediate superclass (identified by node number 2) in between a class (node 1) and its old superclass (node 3). *PullUpVariable* moves a variable contained in a class up to its superclass. The dependency between both transformation rules, as computed by AGG, is visualized in Figure 26. The effect of applying *CreateSuperclass* before *PullUpVariable* will be that the variable will be pulled up to the newly introduced intermediate superclass instead of the old one. As such, there is a sequential dependency between both refactoring rules. It is even the case, in this example, that the application of both refactorings in a different order will produce a different result.

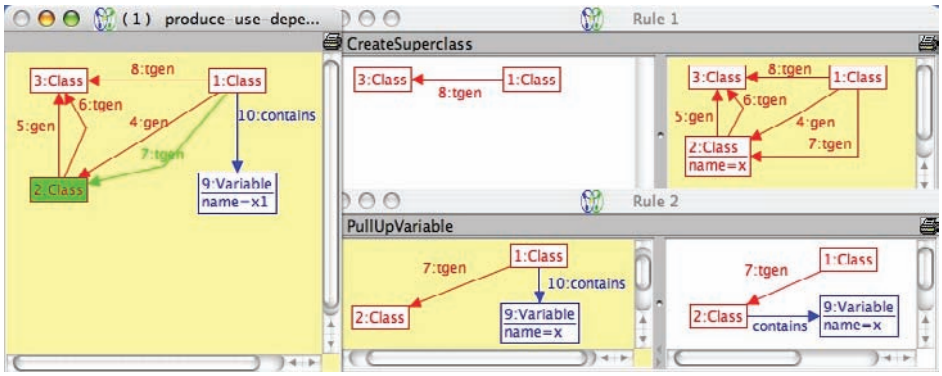
For a more detailed discussion of how critical pair analysis can be used to reason about refactoring dependencies, we refer to (Mens *et al.*, 2007) that provides a detailed account on these issues.

Related Work

Various authors have proposed to use some kind of rule-based approach to specify model refactorings, so it appears to be a natural choice:

Grunske *et al.* (2005) show an example in Fujaba[6] of how model refactoring may be achieved using graph transformation based on story-driven modeling. Bottoni *et al.* (2005) use distributed graph transformation concepts to specify coherent refactorings of several software artifacts, especially UML models and Java programs. Both kinds of artifacts are represented by their abstract syntax structures. Synchronized rules are defined to specify not only refactoring

Figure 26. Example of a sequential dependency between the CreateSuperclass and the PullUpVariable refactoring (©2007 Tom Mens, UMH. Used with permission)



on models and programs separately, but to update also the correlation between different model parts and program. Synchronized rules are applied in parallel to keep coherence between model and program. Considering the special case where exactly two parts (one model diagram and the program or two model diagrams) are related, the triple graph grammar (TGG) approach could also be used (Schürr 1994; Königs & Schürr 2006). Originally formulated for graphs, TGGs are also defined and performed on the basis of MOF models by the modeling environment MOFLON[7].

(Porres, 2003) uses the transformation language SMW to specify model refactorings. This script language is also rule-based and resembles the Object Constraint Language (OCL). SMW is oriented at OCL for querying patterns, but also provides basic operations to realize transformations. A prototypical refactoring tool for UML models has been implemented based on SMW.

Van Der Straeten & D'Hondt (2006) suggest using a rule-based approach to apply model refactorings, based on an underlying inconsistency detection and resolution mechanism implemented in the description logics engine RACER[8].

We decided to specify model refactorings based on EMF model transformation, since EMF is developing to a standard format for models and to be compatible with upcoming UML CASE tools based on EMF. Moreover, our approach opens up the possibility for analyzing model refactorings, since EMF model transformations can be translated to algebraic graph transformations.

SUMMARY

Software complexity is constantly increasing, and can only be tamed by raising the level of abstraction from code to models. With the model-driven software engineering paradigm, automated code generation techniques can be used to hide the accidental complexity of the underlying technology (Brooks, 1995). This enables one to deal with complex software in a systematic way.

To guarantee high-quality software, it is also important to address concerns such as readability, extensibility, reusability and usability of software. Software refactoring is a proven technique to reach these goals in a structured, semi-automated manner.

By integrating the process of refactoring into model-driven software development, we arrive at what we call *model-driven software refactoring.* Analogously to program refactoring, the first phase is to determine potential candidates for model refactorings, which can be obtained using "model smells" and "model metrics". The second phase consists of applying the selected refactorings. This would be a relatively straightforward issue, if hundred-percent code generation were achievable. In practice, for large and complex software systems, this is not the case. Full code generation is not even desirable in many situations since – at least for describing algorithms or data conversions – source code seems to be more adequate than behavioral models. An additional difficulty is the lack of a general accepted semantics of UML. This makes it very difficult to determine whether a given model transformation is behavior preserving, which is the main criterion to decide whether something can be called refactoring or not, according to Fowler (1999).

As a feasibility study, we have chosen AndroMDA to illustrate the model-driven development of web applications. We illustrated and discussed a number of standard and domain-specific restructurings. Since they often change the observable behavior of the software in some sense, we explored to what extent they can be considered as refactorings. All restructurings were categorized into three groups, ordered by the fulfillment degree of Fowler's criterion. The obtained results show that we should address the notion of model refactoring with care, and may serve as suggestions for better tool support:

- We may want to support refactorings that do not fully preserve behavior, as long as they improve other important software quality aspects. This also implies that we need techniques to assess the effect of a model transformation on the software quality.
- We need to find a balance between, and provide user support for the ability to specify generic model refactorings, and the ability to adapt and refine these refactorings to work properly in a domain-specific modeling language;
- We need to provide an interactive round-trip engineering approach to refactoring. When performing model refactorings, it turns out that manual intervention is frequently required in order to keep the abstraction levels of *source code* and *model* consistent. Model refactorings may also affect and require changes to the hand-written source code.

From a theoretical point of view, we have suggested to use graph transformation to provide a formal specification of model refactorings. It has the advantage of defining refactorings in a generic way, while still being able to provide tool support in commonly accepted modeling environments such as EMF. In addition, the theory of graph transformation allows us to formally reason about dependencies between different types of refactorings. Such a static analysis of potential conflicts and dependencies between refactorings can be helpful for the user during the interactive process of trying to improve the software quality by means of disciplined model transformations.

FUTURE RESEARCH DIRECTIONS

In Section 4, we identified many important challenges in model-driven software refactoring. We only worked out some of these challenges in more detail: the need for a formal specification of model refactorings, the need to reason about behavior preservation, the need to synchronize models and source code whilst applying refactorings, the need to relate and integrate the aspects of model refactoring and model quality. There are still many other challenges that remain largely unaddressed:

When developing large software systems in a model-driven manner, several development teams might be involved. In this case, it would be advantageous if the model could be subdivided into several parts that could be developed in a distributed way. Considering refactoring in this setting, model elements from different submodels might be involved. Thus, several distributed refactoring steps have to be performed and potentially synchronized if they involve common model parts. Distributed refactoring steps could be considered as distributed model transformations (Goedicke *et al.*, 1999; Bottoni *et al.*, 2005).

The usual way to test refactorings is by testing the code before and after refactoring steps. Clearly, the code has to satisfy the same test cases before and after refactoring it. Considering refactoring within model-driven development, the same testing procedure should be possible, i.e., test cases for the generated code before and after refactoring should produce the same results. As we discussed within this chapter, model-driven software refactoring often does not fulfill Fowler's criterion in a stringent way. Future investigations should clarify the impact of this kind of restructuring on test suites (Van Deursen *et al.*, 2002).

An important pragmatic challenge that has not been addressed in this chapter has to do with performance and scalability. Is it possible to come up with solutions that scale up to industrial software? Egyed (2006) provided initial evidence that this is actually the case, by providing an instant model synchronization approach that scales up to large industrial software models.

Another interesting research direction is to

apply refactorings at the meta-model level. This raises the additional difficulty of needing to convert all models that conform to this meta-model accordingly, preferably in an automated way.

REFERENCES

Astels, D. (2002). Refactoring with UML. In M. Marchesi, G. Succi (Eds.), *Proceedings of 3rd International Conference eXtreme Programming and Flexible Processes in Software Engineering* (pp. 67-70), Alghero, Italy.

Biermann, E., Ehrig, K., Köhler, C., Taentzer, G., & Weiss, E. (2006). Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In O. Nierstrasz (Ed.), *Proceedings of International Conference on Model Driven Engineering Languages and Systems* (pp. 425-439), Lecture Notes in Computer Science 4199, Heidelberg: Springer.

Boger, M., Sturm, T., & Fragemann, P. (2002). Refactoring Browser for UML. In M. Marchesi, G. Succi (Eds.), *Proceedings 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering* (pp. 77-81), Alghero, Italy.

Bottoni, P., Parisi-Presicce, F., Mason, G., & Taentzer, G. (2005). Specifying Coherent Refactoring of Software Artefacts with Distributed Graph Transformations. In P. van Bommel (Ed.), *Handbook on Transformation of Knowledge, Information, and Data: Theory and Applications* (pp. 95-125). Hershey, PA: Information Science Publishing.

Bouden, S. (2006). *Étude de la traçabilité entre refactorisations du modèle de classes et refactorisations du code*. Unpublished masters dissertation, Université de Montréal, Canada.

Brooks, F. P. (1995). No Silver Bullet: Essence and accidents of software engineering. In *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Reading, MA: Addison-Wesley.

Brottier, E., Fleurey, F., & Le Traon, Y. (2006). Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In *Proceedings 17th International Symposium on Reliability Engineering* (pp. 85-94), IEEE Computer Society.

Correa, A., & Werner, C. (2004). Applying Refactoring Techniques to UML/OCL Models. In *Proceedings International Conference UML 2004* (pp. 173-187), Lecture Notes in Computer Science 3273, Heidelberg: Springer.

Demeyer, S., Ducasse, S., & Nierstrasz, O. (2000). Finding Refactorings Via Change Metrics. In *Proceedings International Conference OOPSLA 2000* (pp. 166-177). ACM SIGPLAN Notices 35(10), ACM Press.

Du Bois, B. (2006). *Quality-Oriented Refactoring.* Unpublished doctoral dissertation, Universiteit Antwepen, Belgium.

Egyed, A. (2006). Instant consistency checking for the UML. In *Proc. International Conference on Software Engineering* (pp. 31-390), ACM.

Ehrig, H, Ehrig, K. Prange, U. & Taentzer, G. (2006). *Fundamental Approach to Graph Transformation*. EATCS Monographs, Heidelberg: Springer.

Ehrig, H., Tsioalikis, A. (2000). Consistency analysis of UML class and sequence diagrams using attributed graph grammars. In *ETAPS 2000 workshop on graph transformation systems* (pp. 77-86).

Fenton, N., & Pfleeger, S. L. (1997). *Software Metrics: A Rigorous and Practical Approach* (2nd edition). London, UK: International Thomson Computer Press.

Fowler, M. (1999) *Refactoring: Improving the*

Design of Existing Code. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Languages and Systems*. Addison-Wesley.

Gheyi, R., Massoni, T., & Borba, P. (2005). A Rigorous Approach for Proving Model Refactorings. In *Proceedings 20th IEEE/ACM International Conference Automated Software Engineering (*pp. 372-375). IEEE Computer Society.

Gheyi, R., Massoni, T., & Borba, P. (2005). Type-safe Refactorings for Alloy. In *Proceedings 8th Brazilian Symposium on Formal Methods (*pp. 174-190). Porto Alegre, Brazil.

Goedicke, M., Meyer, T., & Taentzer, G. (1999). Viewpoint-oriented software development by distributed graph transformation: Towards a basis for living with inconsistencies. In *Proceedings International Conference Requirements Engineering* (pp. 92-99). IEEE Computer Society.

Grundy, J. C., Hosking, J.G., & Mugridge W. B. (1998). Inconsistency Management for Multiple-View Software Development Environments, IEEE *Transactions on Software Engineering*, 24(11), 960-981.

Grunske, L., Geiger, L., Zündorf, A., Van Eetvelde, N., Van Gorp, P., & Varro, D. (2005). Using Graph Transformation for Practical Model Driven Software Engineering. In S. Beydeda, M. Book, & V. Gruhn (Eds.), *Model-driven Software Development* (pp. 91-118). Heidelberg: Springer.

Kamthan, P. (2004). A Framework for Addressing the Quality of UML Artifacts. *Studies in Communication Sciences*, 4(2), 85-114.

Kerievsky, J. (2004). *Refactoring to Patterns*. Addison-Wesley.

Königs, A. & Schürr, A. (2006). Tool Integration with Triple Graph Grammars - A Survey . In R. Heckel (Ed.), *Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques* (pp. 113-150). Electronic Notes in Theoretical Computer Science 148, Amsterdam: Elsevier.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry D. E., & Turski, W. M. (1997). Metrics and laws of software evolution: The nineties view. In *Proceedings of International Symposium on Software Metrics* (pp. 20-32). IEEE Computer Society Press.

Liu, W., Easterbrook, S., & Mylopoulos, J. (2002). Rule-based detection of inconsistency in UML models. In *Proceedings UML Workshop on Consistency Problems in UML-based Software Development* (pp. 106-123). Blekinge Insitute of Technology.

Markovic, S., & Baar, T. (2005). Refactoring OCL Annotated UML Class Diagrams. In L. Briand, C. Williams (Eds.), *Proceedings International Conference Model Driven Engineering Languages and Systems* (pp. 280-294). Lecture Notes in Computer Science 3713, Heidelberg: Springer

Mens, T. (2006). On the use of graph transformations for model refactoring. In *Generative and Transformational Techniques in Software Engineering* (pp. 219-257). Lecture Notes in Computer Science 4143, Heidelberg: Springer.

Mens, T., & Tourwé, T. (2004). A Survey of Software Refactoring. IEEE *Transactions on Software Engineering*, 30(2), 126-162.

Mens, T., Van Eetvelde, N., Demeyer, S., & Janssens, D. (2005). Formalizing refactorings with graph transformations. *Journal on Software Maintenance and Evolution*, 17(4), 247-276.

Mens, T., Van Der Straeten, R., & D'Hondt, M. (2006). Detecting and resolving model inconsistencies using transformation dependency analysis, In O. Nierstrasz (Ed.), *Proceedings International Conference on Model-Driven Engineering Languages and Systems* (pp. 200-

214). Lecture Notes in Computer Science 4199, Heidelberg: Springer.

Mens, T., Taentzer, G., & Runge, O. (2007). Analyzing Refactoring Dependencies Using Graph Transformation. *Journal on Software and Systems Modeling,* 6(3), 269-285.

Mottu, J.-M., Baudry, B., & Le Traon, Y. (2006). Mutation Analysis Testing for Model Transformations. In *Proceedings 2nd European Conference on Model Driven Architecture – Foundations and Applications* (pp. 376-390). Lecture Notes in Computer Science 4066, Heidelberg: Springer.

Parnas, D.L. (1994). Software Aging. In *Proceedings of International Conference on Software Engineering* (pp. 279-287). IEEE Computer Society Press.

Porres, I. (2003). Model refactorings as rule-based update transformations. In: P. Stevens, J. Whittle, G. Booch (Eds.), In *Proceedings of 6th International Conference UML 2003 - The Unified Modeling Language. Model Languages and Applications* (pp. 159-174). Lecture Notes in Computer Science 2863, Heidelberg: Springer.

Pretschner, A., & Prenninger, A. (2007). Computing Refactorings of State Machines, *Journal on Software and Systems Modeling.* To appear.

Schürr, A. (1994). Specification of Graph Translators with Triple Graph Grammars. In: G. Tinhofer (Ed.), *WG94 20th International Workshop on Graph-Theoretic Concepts in Computer Science* (pp. 151-163). Lecture Notes in Computer Science 903, Heidelberg: Springer.

Spanoudakis, G., & Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. In *Handbook of Software Engineering and Knowledge Engineering* (pp. 329-80). World Scientific

Sunyé, G., Pollet, D., Le Traon, Y., & Jézéquel, J.-M. (2001). Refactoring UML models, In *Proceedings International Conference Unified Modeling Language* (pp. 134-138). Lecture Notes in Computer Science 2185, Heidelberg: Springer.

Tahvildari, L., & Kontogiannis, K. (2004). Improving Design Quality Using Meta-Pattern Transformations: A Metric-Based Approach, *Journal of Software Maintenance and Evolution*, 16(4-5), 331-361.

van Deursen, A., & Moonen, L. (2002). The Video Store Revisited: Thoughts on Refactoring and Testing, In M. Marchesi, G. Succi (Eds.), *Proceedings 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering* (pp. 71-76). Alghero, Italy.

van Deursen, A., Moonen, L., van den Bergh, A., & Kok, G. (2002). Refactoring Test Code, In G. Succi, M. Marchesi, D. Wells, & L. Williams (Eds.), *Extreme Programming Perspectives* (pp. 141-152). Addison-Wesley.

Van Der Straeten, R. (2005). *Inconsistency Management in Model-driven Engineering: An Approach using Description Logics.* Unpublished doctoral dissertation, Vrije Universiteit Brussel, Belgium.

Van Der Straeten, R., & D'Hondt, M. (2006). Model refactorings through rule-based inconsistency resolution. In *Proceedings Symposium on Applied computing* (pp. 1210-1217). New York: ACM Press

Van Der Straeten, R., Mens, T., Simmonds, J., & Jonckers, V. (2003). Using description logics to maintain consistency between UML models. In *Proceedings International Conference on The Unified Modeling Language* (pp. 326-340). Lecture Notes in Computer Science 2863, Heidelberg: Springer.

Van Der Straeten, R., Jonckers, V., & Mens, T. (2004). Supporting Model Refactorings through Behaviour Inheritance Consistencies. In T. Baar, A. Strohmeier, & A. Moreira (Eds.), *Proceedings of International Conference on The Unified Mod-*

eling Language (pp. 305-319). Lecture Notes in Computer Science 3273, Heidelberg: Springer.

Van Gorp, P., Stenten, H., Mens, T., & Demeyer, S. (2003). Towards automating source-consistent UML refactorings. In P. Stevens & J. Whittle & G. Booch (Eds.), *Proceedings International Conference on The Unified Modeling Language* (pp. 144-158). Lecture Notes in Computer Science 2863, Heidelberg: Springer.

Van Kempen, M., Chaudron, M., Koudrie, D., & Boake, A. (2005). Towards Proving Preservation of Behaviour of Refactoring of UML Models. In *Proceedings SAICSIT 2005* (pp. 111-118).

Zhang, J., Lin, Y., & Gray, J. (2005). Generic and Domain-Specific Model Refactoring using a Model Transformation Engine. In *Model-driven Software Development - Research and Practice in Software Engineering.* Springer.

ADDITIONAL READING

General and up-to-date information about **graph transformation** *can be obtained via the website http://www.gratra.org/. For those readers wishing to get more in-depth information about what graph transformation is all about, we refer to the 3-volume "bible" of graph transformation research. Volume 1 focuses on its theoretical foundations; Volume 2 addresses applications, languages and tools; and Volume 3 deals with concurrency, parallelism and distribution.*

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1.* World Scientific.

Ehrig, H., Engels, G., Kreowski, H.-J., & Rozenberg G. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2.* World Scientific.

Ehrig, H., Kreowski, H.-J., Montanari, U., & Rozenberg, G. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3,* World Scientific.

Background information about **model-driven software engineering** *can be obtained via the website http://www.planetmde.org/. This includes tool support and events devoted to this very active research domain. Many books on this topic have been published. In particular, we found the following ones to be very useful and relevant:*

Beydeda, S., Book, M., & Gruhn, V. (2005). *Model-Driven Software Development,* Springer.

Stahl, T., & Völter, M. (2006). *Model-Driven Software Development,* Wiley.

With respect to **software evolution** *research, we suggest to consult the website http://www.planet-evolution.org/. Many books on this topic have been published. In particular, we found the following ones to be very useful and relevant:*

Grubb, P., & Takang, A.A. (2003). *Software Maintenance: Concepts and Practice* (Second Edition). World Scientific.

Madhavji, N. H., Fernandez-Ramil, J., & Perry, D. (2006). *Software Evolution and Feedback: Theory and Practice.* Wiley.

Mens, T., & Demeyer, S. (2008). *Software Evolution.* Springer.

Seacord, R., Plakosh, D. & Lewis, G. (2003). *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices* (SEI Series in Software Engineering). Addison-Wesley.

Regarding **software refactoring** *in particular,*

we would like to point to some of the early work on refactoring, which has been published in the following PhD dissertations:

Griswold, W.G. (1991). *Program Restructuring as an Aid to Software Maintenance.* Unpublished doctoral dissertation, University of Washington.

Opdyke, W.F. (1992). *Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign.

Roberts, D. (1999). *Practical Analysis for Refactoring.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign.

O Cinnéide, M. (2000). *Automated Application of Design Patterns: A Refactoring Approach.* Unpublished doctoral dissertation, Trinity College, University of Dublin.

Tichelaar, S. (2001). *Modeling Object-Oriented Software for Reverse Engineering and Refactoring.* Unpublished doctoral dissertation, University of Bern.

There are many useful standards that have been published for software maintenance and software evolution. As is frequently the case, some of these standards may be somewhat outdated compared to the current state-of-the-art in research:

The ISO/IEC 14764 standard on ``Software Maintenance'' (1999)

The IEEE 1219 standard on ``Software Maintenance'' (1999)

The ISO/IEC 12207 standard (and its amendments) on ``Information Technology - Software Life Cycle Processes'' (1995)

The ANSI/IEEE 1042 standard on ``Software Configuration Management'' (1987)

ENDNOTES

1. http://galaxy.andromda.org
2. This model refactoring is actually domain-specific, as will be discussed later in this chapter.
3. Of course, programs also need to be synchronized with related software artefacts such as databases, user interfaces, test suites and so on. Each of these kinds of artefacts may have been expressed using a different language.
4. It is not our goal to be complete here.
5. http://tfs.cs.tu-berlin.de/agg
6. http://www.fujaba.de
7. http://www.moflon.org
8. http://www.racer-systems.com

This work was previously published in Model-Driven Software Development: Integrating Quality Assurance, edited by J. Rech & C. Bunse, pp. 170-203, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.17
Benchmarking in the Semantic Web

Raúl García-Castro
Universidad Politécnica de Madrid, Spain

Asunción Gómez-Pérez
Universidad Politécnica de Madrid, Spain

ABSTRACT

The Semantic Web technology needs to be thoroughly evaluated for providing objective results and obtaining massive improvement in its quality; thus, the transfer of this technology from research to industry will speed up. This chapter presents software benchmarking, a process that aims to improve the Semantic Web technology and to find the best practices. The chapter also describes a specific software benchmarking methodology and shows how this methodology has been used to benchmark the interoperability of ontology development tools, employing RDF(S) as the interchange language.

INTRODUCTION

The Semantic Web technology has considerably improved since the 1990's, when the first tools were developed; although it has mainly been applied in research laboratories, in recent years companies have started to be interested in this technology and its application.

To transfer the Semantic Web technology from the academia, its current niche, to the industrial world it is necessary that this technology reaches a maturity level that enables it to comply with the quality requirements of the industry. Therefore, the Semantic Web technology needs to be thoroughly evaluated both for providing objective results and for attaining a massive improvement in its quality.

Until recently, the Semantic Web technology was seldom evaluated; now, however, this technology is widely used and numerous studies concerning its evaluation have appeared in the last few years. So now it seems quite necessary that researchers increase the quality of their evaluations and improve the technology collec-

tively by benchmarking it, employing for this a methodological process.

Evaluating and benchmarking this technology within the Semantic Web can be quite costly because most of the people involved do not know how to carry out these processes and also because no standard nor agreed methods to follow now exist. On the other hand, since it is quite difficult to reuse the results and put into practice the lessons learnt in previous activities, it is necessary to develop new methods and tools every time this technology has to be evaluated or benchmarked.

Software benchmarking is presented in this chapter as a continuous process whose aim is to improve software products, services, and processes by evaluating and comparing them with those considered the best. Although software evaluations are performed inside the benchmarking activities, benchmarking provides some benefits that cannot be obtained from evaluations, as for example, the continuous improvement of software, or the extraction of the best practices used to develop the software.

Within the Knowledge Web[a] European Network of Excellence a new methodology for benchmarking Semantic Web technology has been developed; this methodology is now being adopted in different benchmarking studies and applied to the different types of Semantic Web technologies (ontology development tools, ontology alignment tools, ontology-based annotation tools, and reasoners). The methodology focuses on the special interests of the industry and research fields and on their different needs. At the end of the chapter, we describe how we have followed this methodology during one of the activities performed to benchmark the interoperability of ontology development tools, employing RDF(S) as the interchange language.

EVALUATION AND BENCHMARKING IN THE LITERATURE

Software Evaluation

Software evaluation plays an important role in different areas of Software Engineering, such as Software Measurement, Software Experimentation or Software Testing. In this section, we present a general view of these areas.

According to the ISO 14598 standard (ISO/IEC, 1999), software evaluation is *the systematic examination of to which extent an entity is capable of fulfilling specified requirements*; this standard considers software not just as a set of computer programs but also as a set of procedures, documentation and data.

Software evaluation can take place all along the software life cycle. It can be performed during the software development process by evaluating intermediate software products or when the development has finished.

Although evaluations are usually carried out inside the organisation that develops the software, other independent groups such as users or auditors can also make them. When independent third parties evaluate software, they are usually very effective, though their evaluations can become very expensive (Rakitin, 1997).

The goals of evaluating software vary since they depend on each specific case, but in general, they can be summarised (Basili et al., 1986; Park et al., 1996; Gediga et al., 2002) as follows:

- To **describe** the software in order to understand it and establish baselines for comparisons.
- To **assess** the software with respect to some quality requirements or criteria and determine the degree of quality required from the software product and its weaknesses.

- To **improve** the software by identifying opportunities and, thus, improving its quality. This improvement is measured by comparing the software with the baselines.
- To **compare** alternative software products or different versions of a same product.
- To **control** software quality by ensuring that it meets the required level of quality.
- To **predict** new trends in order to take decisions and establish new goals and plans for accomplishing them.

It is understood that the methods to follow to evaluate software vary from one author to another and from one Software Engineering area to another. However, from the methods proposed in the areas of a) Software Evaluation (ISO/IEC, 1999; Basili, 1985), b) Software Experimentation (Basili & Selby, 1991; Perry et al., 2000; Freimut et al., 2002), c) Software Measurement (Park et al., 1996; IEEE, 1998), and d) Software Testing (Abran et al., 2004) we can extract a common set of tasks that must be carried out in software evaluations. These tasks are the following:

1. To establish the evaluation requirements by setting its goals, the entities to evaluate, and their relevant attributes.
2. To define the evaluation by explaining the data to collect, the evaluation criteria to follow, and the mechanisms to collect data and implement these mechanisms.
3. To produce the evaluation plan.
4. To execute the evaluation and to collect data.
5. To analyse the collected data.

Benchmarking in the Literature

In the last decades, the word benchmarking has become relevant within the business management community. The most well-known definitions of the term are those by Camp (1989) and Spendolini (1992). Camp defines benchmarking as *the search for industry best practices that lead to superior performance*; on the other hand, Spendolini has expanded Camp's definition by adding that benchmarking is *a continuous, systematic process for evaluating the products, services, and work processes of organisations that are recognised as representing best practices for the purpose of organisational improvement.* In this context, best practices are good practices that have worked well elsewhere, are proven and have produced successful results (Wireman, 2003).

These definitions highlight the two main benchmarking characteristics:

- Continuous improvement.
- Search for best practices.

The Software Engineering community does not share a common benchmarking definition but several. Here we present some of the most representative:

- Both Kitchenham (1996) and Weiss (2002) define benchmarking as a software evaluation method suitable for system comparisons. Whereas for Kitchenham benchmarking is *the process of running a number of standard tests using a number of alternative tools/methods and assessing the relative performance of the tools in those tests*, for Weiss benchmarking is *a method of measuring performance against a standard, or a given set of standards.*
- Wohlin et al. (2002) have adopted the benchmarking definition from the business world, that is, they consider benchmarking as a continuous improvement process that strives to be the best of the best through the comparison of similar processes in different contexts.

Software Benchmarking

In this section, we have followed the notions that support continuous improvement and search for best practices within business management benchmarking; these notions have led us to consider software benchmarking as a continuous improvement process instead of as a punctual activity. Equally important for us are the concept of comparing software through evaluations and that of carrying out the benchmarking activity through a systematic procedure.

All these concepts permit us to define **software benchmarking** as a continuous process whose aim is to improve software products, services, and processes by systematically evaluating and comparing them with those considered to be the best.

This definition, however, does not specify the type of the entities considered in benchmarking (software products, services or processes), nor does it determine the software life-cycle phase when benchmarking is performed, and nor does it explain who is responsible for benchmarking. However, software benchmarking is usually performed on software products already developed and is executed by their developers.

The reason for benchmarking software products instead of just evaluating them is to gain those benefits that cannot be obtained from software evaluations. When we evaluate software we can observe its weaknesses and its compliance to quality requirements. If, on the other hand, several software products are involved in the evaluation, then we can have a comparative analysis of these products and provide some recommendations. But when we benchmark several software products, in addition to all the benefits commented, we obtain products that are continuously improved, recommendations for developers on the practices used and, from these practices, those that can be considered the best.

Software Evaluation in Benchmarking Activities

To evaluate software is not a straightforward task; however, as this is an issue that has been thoroughly examined both in theory and practice, several authors have proposed different recommendations to consider (Basili et al., 1986; Perry et al., 2000; Freimut et al., 2002; Juristo & Moreno, 2001; Kitchenham et al., 2002).

These recommendations are also applicable to the software evaluations made during the benchmarking activities. However, when we have to define this kind of software evaluations, we must take into account some additional recommendations.

And the most important recommendation is that the evaluation of the benchmarking procedure must be an **improvement-oriented activity**. Its intended results will not only be used for comparing the different software products, but for learning how to improve such products. This requires that the evaluation yield not only some comparative results but also that it show the practices that produced these results.

Another recommendation is that benchmarking evaluations should be as **general** as possible since they will be performed by different groups of people in different locations and on different software.

Benchmarking is a process driven by a community; therefore, to gain credibility, effectiveness and impact, its evaluations should also be **community-driven**.

Additionally, benchmarking evaluations should be **reproducible** since they are intended to be used not only by the people participating in the benchmarking, but by the whole community. This requires that the evaluation be thoroughly detailed, providing public data and procedures.

To perform the evaluations consumes significant resources; these evaluations, on the other

hand, must be made by several groups of people. Therefore, evaluations should be as **economical** as possible, employing for this activity common evaluation frameworks and, when this is not possible, limiting the scope of the evaluation.

Furthermore, as benchmarking is a continuous process, benchmarking evaluations should have a **limited scope**, leaving other objectives for the next benchmarking iterations and incrementing progressively the complexity of the evaluations. We must add here that a broader evaluation scope does not entail better results but more resources.

As the next section shows, most of these recommendations should also be adopted in the benchmark suites. Therefore, it is advisable to **use benchmark suites** in the evaluations.

Benchmark Suites

A benchmark suite is a collection of benchmarks, being a benchmark *a test or set of tests used to compare the performance of alternative tools or techniques* (Sim et al., 2003).

A benchmark definition must include the following:

- The **context** of the benchmark, namely, which tools and which of their characteristics are measured with it (efficiency, interoperability, portability, usability, etc.).
- The **requirements** for running the benchmark, namely, the tools (hardware or software), data, or people needed.
- The **input variables** that affect the execution of the benchmark and the values that the variables will take.
- The **procedure** to execute the benchmark and obtain its results.
- The **evaluation criteria** followed to interpret these results.

A benchmark suite definition must include the definitions of all its benchmarks. Generally, all these benchmarks share some of their characteristics, such as context or requirements. These characteristics, therefore, must be specified at the benchmark suite level, and not individually for each benchmark.

Desirable Properties of a Benchmark Suite

The properties below, which are extracted from the works of different authors (Bull et al., 1999; Shirazi et al., 1999; Sim et al., 2003; Stefani et al., 2003), can help both to develop new benchmark suites and to assess the quality of the existing ones before being used.

Although a good benchmark suite should have most of these properties, each evaluation will require considering some of them previously. However, we must not forget that achieving these properties completely is not possible since the increment of some properties has a negative effect on the others.

- **Accessibility.** A benchmark suite must be accessible to anyone interested. This involves providing (a) the necessary software to execute the benchmark suite, (b) the software documentation, and (c) the software source code to increase transparency. Then the results obtained from executing the benchmark suite should be made public so that anybody can execute it and then compare his/her results with those available.
- **Affordability.** Using a benchmark suite normally entails a number of costs regarding human, software, and hardware resources. Thus, the costs of using a benchmark suite must be lower than those of defining, implementing, and carrying out any other experiments that fulfil the same goal. On the other hand, the resources consumed in the execution of a benchmark suite can be reduced by (a) automating the execution of

the benchmark suite, (b) providing components for data collection and analysis, and (c) facilitating its use in different heterogeneous systems.

- **Simplicity.** The benchmark suite must be simple and interpretable and should be well documented; therefore, whoever wants to use it must be able to understand how it works and the results that it yields. If the benchmark suite is not transparent enough, its results will be questioned and may be interpreted incorrectly. To avoid this, the elements of the benchmark suite should have a common structure and use, and common inputs and outputs. Measurements, on the other hand, should have the same meanings across the benchmark suite.
- **Representativity.** The actions that perform the benchmarks composing the benchmark suite must be representative of the actions normally performed on the system.
- **Portability.** The benchmark suite should be executed on a wide range of environments and should be applicable to as many systems as possible. Besides, it should be specified at a high enough level of abstraction to ensure that it can be transferred to different tools and techniques and that is not biased against other technologies.
- **Scalability.** The benchmark suite should be parameterised to allow scaling the benchmarks with varying input rates. In addition, it should work with tools or techniques of different levels of maturity and should be applicable to research prototypes and commercial products.
- **Robustness.** The benchmark suite must allow for unpredictable environment behaviours and should not be sensitive to factors irrelevant to the study. When running the same benchmark suite on a given system and under the same conditions several times, the results obtained should not vary considerably.
- **Consensus.** The benchmark suite must be developed by experts capable of applying their knowledge of the domain and of identifying the key problems. Additionally, it should be assessed and agreed on by the whole community.

EVALUATION AND BENCHMARKING WITHIN THE SEMANTIC WEB

This section provides an overview of the evaluation and benchmarking trends now occurring in the Semantic Web area; it also describes to what extent the evaluation and benchmarking activities performed on the Semantic Web technology can be partially or totally reused in different tools, and the facilities provided for doing so.

To this end, we have performed a survey of the main conferences on the Semantic Web field and of the workshops whose main topic is Semantic Web technology evaluation. We have examined the proceedings of five conferences: International Semantic Web Conference (ISWC), European Semantic Web Conference (ESWC), Asian Semantic Web Conference (ASWC), International Conference on Knowledge Capture (K-CAP), and International Conference on Knowledge Engineering and Knowledge Management (EKAW); and we have studied five workshops: Workshop on Evaluation of Ontology-based Tools (EON), Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS), Workshop on Practical and Scalable Semantic Systems (PSSS), International Workshop on Ontology Matching (OM), and Workshop on Integrating Ontologies (IntOnt). The survey includes all the papers accepted in these conferences and workshops from 2000 to 2006.

It is clear that the papers examined, which were presented in the conferences and workshops above commented, do not provide exhaustive information, but they can provide an overview of the current trends.

We consider that fulfilling the desirable properties of a benchmark suite and the recommendations for defining evaluations in benchmarking activities, both defined in the previous section, is an indication of evaluation reusability. And thus, with these desirable properties and recommendations in mind we have produced a questionnaire that should be filled in for each of the selected papers. As an example, the questions asked for assessing the portability of an evaluation approach are the following:

- *In which type of tools can the evaluation be performed?*
- *Can the evaluation approach be transferred to other tools?*
- *On which of the operating systems/platforms can the evaluation be performed?*

Although software evaluations are frequent in research papers, we focused on those papers where the evaluation approaches followed are reusable to a certain extent. However, we did not distinguish between evaluations that are performed within the benchmarking activities and evaluations that are not, nor did we distinguish between these benchmarking activities that use benchmark suites and those that do not. Thus, the criteria we followed to select the papers are:

- The paper describes how to evaluate several tools, or it shows the results of the evaluation over several tools, or both.
- The evaluation described in the paper involves more than one tool or is intended to be applied to more than one tool.
- The evaluation described in the paper is targeted to software tools or to methods implemented by some software.

With these criteria we selected 61 papers and filled in the questionnaire. Of these papers, 21 deal with the description and application of an evaluation, 7 simply describe an evaluation, and 33 show how an evaluation is made. Among the papers selected, we included those workshop papers that present proposals for performing a common evaluation and the papers written on this evaluation by the participants together with the results of their findings.

Figure 1 and Figure 2 show the trend concerning papers related to evaluation that were published in the last few years. As for the 17 papers presented in the conferences, they have led us to conclude, first, that the number of papers increases every year and, second, that no evaluation-related papers were submitted to the EKAW and K-CAP conferences; this may occur because these conferences are more oriented to knowledge management than to the Semantic Web. With regard to the 44 papers presented in workshops, we have noticed that only 7 of them are regular papers and that the other papers are either evaluation proposals or evaluation results. There is a call for participation in evaluations every year; we have observed that in these evaluations the number of evaluation contributions varies, whereas the number of regular papers is more or less constant.

Our survey shows the results of the reusability of the evaluation approaches in which it can be observed that some of them are positive and some, negative.

The positive results confirm that, in general, the evaluation approaches are easy to understand because they clearly establish both the input data, according to a common structure, and the evaluation criteria for analysing the results. In addition, in most cases, performing the evaluations is not expensive since the evaluation approaches provide common evaluation frameworks and, quite often, the whole evaluation or part of it can be automated. In other cases, however, some software supports the evaluation, being this software and its source code usually accessible. Scalability and robustness have also been taken into account throughout the evaluations.

Figure 1. Evaluation-related papers in conferences

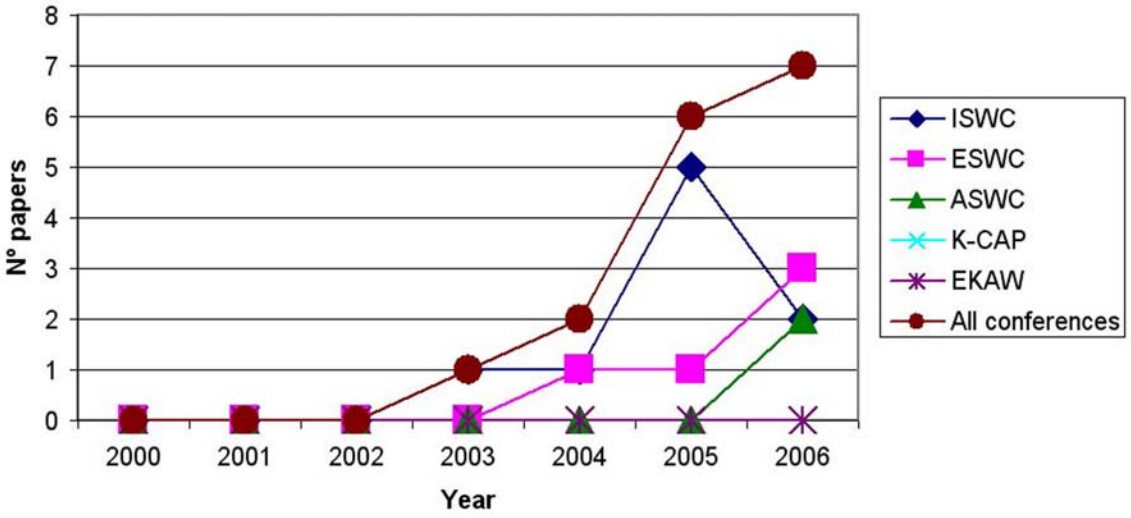
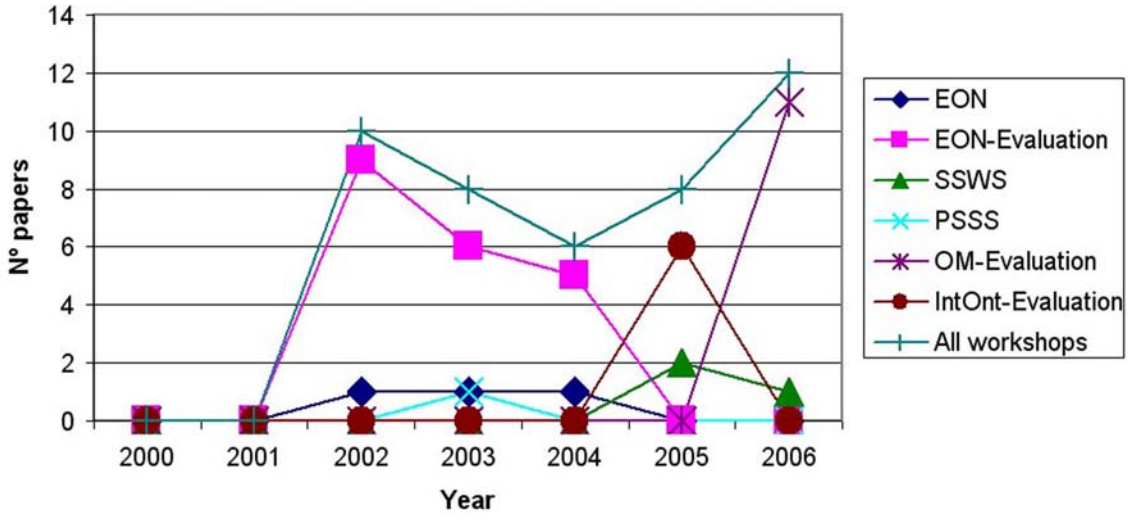


Figure 2. Evaluation-related papers in workshops



The negative results show that most of the papers deal with a small group of Semantic Web tools (ontology alignment tools, ontology development tools and ontology repositories) and that, in general, evaluations are not applicable to other types of tools. Besides, the accessibility of these evaluations is not high, the procedure to perform the evaluation is not always clearly defined, and

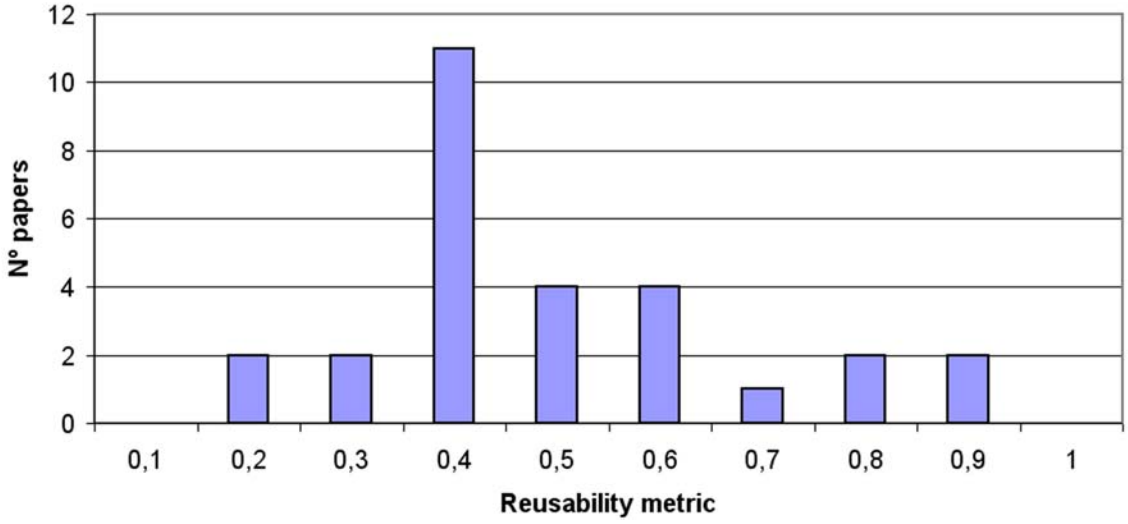
the input data is not always accessible. We can add that only in a few cases the papers provide a web page with information on the evaluation. The input data is not representative of the actions performed on the system and, occasionally, the evaluation has been developed and agreed on by a community. Furthermore, the existing evaluation approaches are not transferrable and the cost of performing the evaluation is never considered; most of the evaluations are defined as one-time activities. Only in a few cases improvement is a goal and, quite often, the practices that lead to the results or improvement recommendations cannot be obtained directly from these results.

As we mentioned before, each of the questions of the questionnaire is related to one of the desirable properties of a benchmark suite. Therefore, we have described the reusability of the evaluation presented in each paper taking into account the number of questions answered positively for each of these properties. For each paper, we have calculated the percentage of positive answers for each dimension and the mean of all these percentages. The resulting value ranges from 0 (not reusable at all) to 1 (totally reusable). Even though this is not a robust metric, it gives us a hint of the reusability of the evaluations.

Figure 3 presents the histogram of the reusability metric used with the 28 papers that describe how to perform an evaluation over several tools. The horizontal axis represents the different ranges of the reusability metric, whereas the vertical axis represents the number of papers in each range. We can clearly observe that most of the papers have low values and are far from the ideal situation.

As a summary of this section we can conclude that although the number of evaluation and benchmarking activities is continuously increasing in the Semantic Web area, such number is still not good enough to ensure a high quality technology, and that the activities carried out involved just a few types of Semantic Web technologies. Consequently, the reusability of the evaluation approaches is not high enough, which is a hindrance for their use.

Figure 3. Histogram of the reusability metric



THE BENCHMARKING METHODOLOGY FOR SEMANTIC WEB TECHNOLOGY

This section summarises the benchmarking methodology for Semantic Web technology developed by the authors in the Knowledge Web European Network of Excellence. A detailed description of this methodology can be found in (García-Castro et al., 2004).

The methodology has been inspired by works of quality improvement from different fields. The main inputs for this methodology were the benchmarking methodologies of the business management community and their notions of continuous improvement and best practices. We have also taken into account the evaluation and improvement processes proposed in the Software Engineering area such as those cited in Section 2.1.

Benchmarking Actors

The tasks of the benchmarking process are carried out by different actors according to the kind of roles to be performed in each task. The different types of actors involved are the following:

- **Benchmarking initiator.** The benchmarking initiator is the member (or members) of an organisation who makes the first tasks of the benchmarking process. His/her work consists in preparing a proposal for carrying out the benchmarking activity in the organisation and in obtaining the management approval to perform it.
- **Organisation management.** The organisation management plays a key role in the benchmarking process: this actor must approve the benchmarking activity and the changes that result from it. The organization management must also assign resources to the benchmarking and integrate the benchmarking planning into the organisation planning.
- **Benchmarking team.** Once the organisation management approves the benchmarking proposal, the benchmarking team is composed. Their members have to belong to the organisation and are responsible for performing most of the remaining benchmarking processes.
- **Benchmarking partners.** The benchmarking partners are the organisations participating in the benchmarking activity. All the partners must agree on the steps to follow during the benchmarking, and their needs must be considered.
- **Tool developers.** The developers of the tool must implement the necessary changes in the tool to improve it. Some of the developers may also be part of the benchmarking team and, if so, care must be taken to minimise bias.

Benchmarking Process

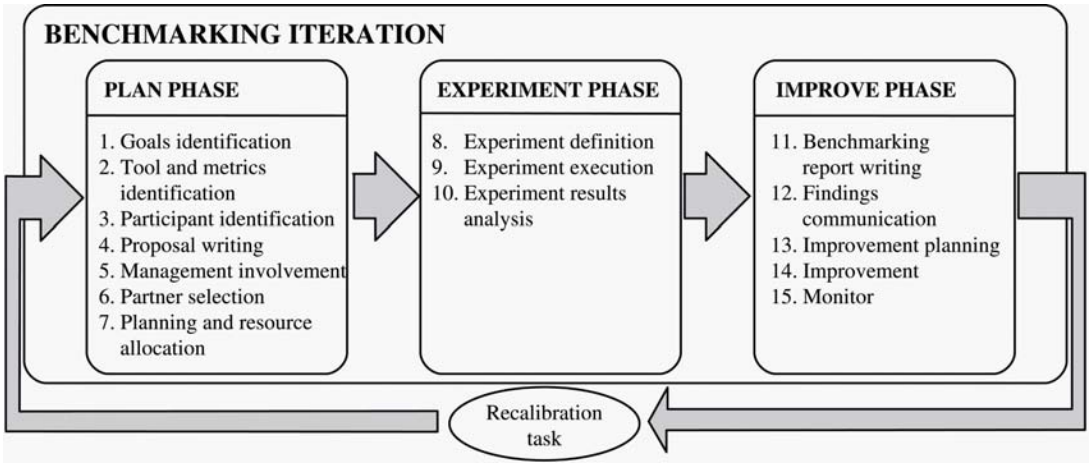
The benchmarking methodology for Semantic Web technology describes a benchmarking process together with the main phases to follow when benchmarking this technology and it also provides a set of guidelines. Therefore, this methodology has a twofold use: to help carry out software benchmarking activities, and to know, at a specific time, which is the actual progress of a benchmarking activity.

The benchmarking process defined in this methodology is composed of a benchmarking iteration that is repeated forever. Each iteration, as shown in Figure 4, is composed of three phases (*Plan*, *Experiment* and *Improve*) and ends with a *Recalibration* task.

Plan Phase

The *Plan* phase is composed of the different tasks that must be performed (1) to prepare the benchmarking proposal, (2) to obtain support from the organisation management, (3) to find

Figure 4. The software benchmarking methodology



other organisations willing to participate in benchmarking, and 4) to plan benchmarking. These tasks are the following:

P1. Goals identification. In this task, the *benchmarking initiator* (the member or members of the organisation who are aware of the need for benchmarking) must identify the benchmarking goals according to the goals and strategies of the organization as well as the benefits and costs involved in carrying out benchmarking. However, every organization may have its own goals and these can be quite different. For example, some may be interested in assessing the performance and improving the quality of the software over its lifetime, others, in comparing their software with the software that is considered the best, whereas some others are interested in establishing or creating standards by analysing the different existing software.

P2. Software and metrics identification. In this task, the *benchmarking initiator* should make an analysis of the software products developed in the organisation in order to understand and document them, identifying the weaknesses and functionalities that require improvement. Then, he/she must select the products to be benchmarked, the functionalities relevant to the study and the evaluation criteria to follow to assess these functionalities; these criteria must take into account the organisation's software, the benchmarking goals, the benefits and costs identified in the previous task as well as other factors considered critical by the organisation, such as quality requirements, end-user needs, etc.

P3. Participant identification. In this task, the *benchmarking initiator* must identify and contact the members concerned with the software and the functionalities selected (managers, developers, end users, etc.) and other relevant participants that do not belong to the organisation (customers or consultants). The benchmarking initiator is responsible for organizing the benchmarking team and, quite often, he is a member of the team. The team should be small and include those organisation members whose work and interest are related to the software, who have a thorough

understanding of the software and have gained valuable experience with it. They must be aware of the time they will spend in the benchmarking activity and of their responsibilities, and they should be trained in the tasks they will have to perform.

P4. Proposal writing. In this task, the *benchmarking team* (and the *benchmarking initiator*, if he does not belong to the team) must write a document with the benchmarking proposal. The proposal will be used as a reference along the benchmarking process and must include all the relevant information on the process: the information identified in the previous benchmarking tasks (goals, benefits, costs, software, metrics, members involved, and benchmarking team), a description of the benchmarking process, and a full detailed description of the benchmarking costs along with the resources needed. To do this, the benchmarking team should take into consideration the different intended readers of the benchmarking proposal, namely, the organisation managers, the organisation developers, the members of partner organisations, and the members of the benchmarking team.

P5. Management involvement. In this task, the *benchmarking initiator* must bring the benchmarking proposal to the *organisation management.* This task is of great significance because the management approval is required if we want to continue with the benchmarking process. Management support will also be requested in the future, when the changes required for benchmarking will have to be implemented, either in the software or in the organisation processes that affect the software.

P6. Partner selection. In this task, the *benchmarking team* must collect and analyse information on the software products that are to be compared with the software selected, and on the organisations that develop the products. The benchmarking team must also select the software employed in the benchmarking study taking into account its relevance and use in the community or in the industry, its public availability, how the software has adopted the latest technological tendencies, etc. In order to obtain better results with benchmarking, the software selected should be the software considered the best. Then, the benchmarking team must contact the people from the organisations that develop these software products to find out whether they are interested in becoming *benchmarking partners.* These partners will also have to establish a team and to take the proposal to their own organisation management for approval. During this task, the benchmarking proposal will be modified by incorporating the partners' opinions and requirements. This will result in an updated proposal that, depending on the magnitude of the modifications, should be presented again to each organisation management for approval.

P7. Planning and resource allocation. In this task, the *organisation managements* and the *benchmarking teams* must specify the planning of the remainder of the process, considering the different resources that will be devoted to it, and finally they must reach a consensus. This planning must be given careful consideration and should be integrated into each organisation planning.

Experiment Phase

The *Experiment* phase is composed of the tasks in which the experimentation on the software products is performed. These tasks are the following:

E1. Experiment definition. In this task, the *benchmarking teams* of each organization must establish the experiment that will be performed on each of the software products, and then the members must agree on it. The experiment must be defined according to the benchmarking goals, the software functionalities selected, and their corresponding criteria, as stated in the benchmarking proposal. The experiment must also provide objective and reliable software data not just of its performance, but also of the reasons of

its performance; in addition, its future reusability must be also be considered. The benchmarking teams must determine and agree on the planning to follow during the experimentation; this new planning must be decided according to the benchmarking planning established previously.

E2. Experiment execution. As indicated in the experimentation planning, explained in the previous task, the *benchmarking teams* must perform the established experiments on their software products. Then the data obtained from all the experiments must be compiled, documented, and written in a common format to facilitate its future analysis.

E3. Experiment results analysis. In this task, the *benchmarking teams* must analyse the results, detect and document any significant differences found in them, and determine the practices leading to these different results in order to identify whether, among the practices found, some of them can be considered the best practices. Then, the benchmarking teams should write a report with all the findings of the experimentation, that is, the experimentation results, the differences in the results, the practices and the best practices found, etc.

Improve Phase

The *Improve* phase comprises the tasks where the results of the benchmarking process are produced and then communicated to the benchmarking partners; it also comprises the tasks where, in several cycles, the improvement of the different software products takes place. The tasks are the following:

I1. Benchmarking report writing. In this task, the *benchmarking teams* must write the benchmarking report. This report is intended to provide an understandable summary of the benchmarking carried out, and it should be written bearing in mind its different audiences: managers, benchmarking teams, developers, etc. The benchmarking report must include a) an explanation of the process followed, together with all the relevant information of the updated version of the benchmarking proposal; and b) the results and conclusions of the experiments presented in the experiment report, including the practices found and highlighting the best practices. The report should also contain the recommendations provided by the benchmarking teams for improving the software products according to the experiment results, the practices found, and the best practices implemented by the community.

I2. Findings communication. Here, the *benchmarking teams* must communicate, in successive meetings, the results of the benchmarking to their organisations and, particularly, to all the members concerned and identified when planning benchmarking. The goals of these meetings are:

- To obtain feedback from the members concerned on the benchmarking process, the results, and the improvement recommendations.
- To obtain support and commitment from the organisation members for implementing the improvement recommendations on the software.

Any feedback received during these communications must be collected, documented and analysed. This analysis may finally involve having to review the work done and to update the benchmarking report.

I3. Improvement planning. The last three tasks of the *Improve* phase (*Improvement planning, Improvement* and *Monitor*) form a cycle that must be carried out separately in each organisation. The *benchmarking teams* and the *organisation managements* must identify, from the benchmarking report and the monitoring reports, which are the changes needed to improve their software products. Besides, they must forecast the improvements to be achieved after performing these changes. Both the organisation management and the benchmarking team must provide

mechanisms for ensuring improvements in their organisation and for measuring the software functionalities. These last mechanisms can be obtained from the *Experiment* phase. Then, the organisation management and the benchmarking team must establish a planning for improving the software benchmarked, taking into account the different resources devoted to the improvement. This planning must be then integrated into the organisation planning.

I4. Improvement. It is in this task where the *developers* of each of the software product must implement the necessary changes to achieve the desired results. For this, they must measure the state of the software before and after implementing any changes, using for that purpose the measurement mechanisms provided by the benchmarking team in the previous task. Then, the developers must compare the resulting measurements with those that were obtained before implementing the changes and with the improvement forecasted.

I5. Monitor. In each organisation, the *benchmarking team* must provide *software developers* with means for monitoring the organisation's software. Software developers must periodically monitor the software and write a report with the results of this process. These results may show the need for new improvements in software and may also mean the beginning of a new improvement cycle which involves having to perform again the two tasks previously mentioned: *Improvement Planning* and *Improvement.*

Recalibration Task

The recalibration task is performed at the end of each iteration. Here the *benchmarking teams* must recalibrate the process by applying the lessons learnt while performing it. Thus, the organisation (and the whole community) achieves improvement not just in the software, but also in the benchmarking process. This recalibration is needed because both the software and the organisations evolve and innovate over time.

BENCHMARKING THE INTEROPERABILITY OF ONTOLOGY DEVELOPMENT TOOLS USING RDF(S) AS THE INTERCHANGE LANGUAGE

This section presents how we have applied the software benchmarking methodology, presented in the previous section, to one important problem of the Semantic Web: technology interoperability.

Ontologies permit interoperability among heterogeneous Semantic Web technologies, and ideally, one could use all the existing technologies seamlessly. The technologies appear in different forms (ontology development tools, ontology repositories, ontology alignment tools, reasoners, etc.), but although all these tools use different kinds of ontologies, not all of them share a common knowledge representation model.

This diversity in the representation formalisms of the tools causes problems when the tools try to interoperate. This is so because the tools require translating their ontologies from their own knowledge models to a common model and vice versa, and these problems occur even when we employ standard APIs for managing ontologies in the common knowledge model.

As we have observed in previous workshops on Evaluation of Ontology-based Tools (EON) (Sure & Corcho, 2003), interoperability among different ontology tools is not straightforward. Finding out why interoperability fails is cumbersome and not at all trivial because any assumption made for translating ontologies within one tool may easily prevent the successful interoperability with other tools.

To solve this problem, the Knowledge Web European Network of Excellence organized a benchmarking of interoperability of ontology development tools using RDF(S) as the interchange language. Its goal was to assess and improve the interoperability of the tools.

The section that follows describes such benchmarking activity. The methodology presented in

the previous section provides the general guidelines that can be adapted to this case. So we present here how this new benchmarking was organized, the experiments conducted on the participating tools, and its results.

Organising the Benchmarking

The goals of benchmarking the interoperability of ontology development tools are related to the benefits pursued through it, and these are:

- To *evaluate and improve* their interoperability.
- To acquire a *deep understanding* of the practices used to develop the importers and exporters of these tools, and to extract from these practices those that can be considered the *best practices.*
- To produce *recommendations* on their interoperability for users.
- To create *consensual processes* for evaluating their interoperability.

These goals concern different communities that somehow are related to the ontology development tools, namely, the research community, the industrial community, and the tool developers.

Participation in the benchmarking was open to any organisation irrespective of being a Knowledge Web partner or not. To involve other organisations in the process with the aim of having the best-in-class tools participating, several actions were taken:

- The benchmarking proposal, the document being used as a reference along the benchmarking, was published as a public web page[b] and included all the relevant information about the benchmarking: motivation, goals, benefits and costs, tools and people involved, planning, related events, and a complete description of the experimentation and the benchmark suites.
- Research was carried out on the existing ontology development tools, both the freely available and the commercial versions, which could export and import to and from RDF(S); In addition, their developers were contacted. Any tool capable of importing and exporting RDF could participate in the benchmarking or will benefit from the created benchmarks in a near future.
- The interoperability benchmarking was announced with a call for participation through the main mailing lists of the Semantic Web area and through lists specific to ontology development tools.

Six tools took part in the benchmarking, three of which are ontology development tools: KAON[c], Protégé[d] (using its RDF backend), and

Table 1. Ontology tools participating in the benchmarking

Tool	Knowledge model	Version	Developer	Experimenter
Corese	RDF(S)	2.1.2	INRIA	INRIA
Jena	RDF(S)	2.3	HP	U. P. Madrid
KAON	RDF(S) extension	1.2.9	U. Karlsruhe	U. Karlsruhe
Sesame	RDF(S)	2.0 alpha 3	Aduna	U. P. Madrid
Protégé	Frames	3.2 beta build 230	Stanford U.	U. P. Madrid
WebODE	Frames	2.0 build 109	U. P. Madrid	U. P. Madrid

WebODE[e]; the other three are RDF repositories: Corese[f], Jena[g] and Sesame[h]. As Table 1 shows, the tools do not share a common knowledge model and benchmarking was not always performed by the tool developers.

The experimentation conducted on the tools aimed to obtain results for interoperability improvement. Therefore, other quality attributes such as performance, scalability, interoperability, robustness, etc. were not considered. However, an approach for benchmarking the performance and scalability of ontology development tools can be found in (García-Castro & Gómez-Pérez, 2005).

The experimentation was carried out taking into account the most common ways of interchanging ontologies that ontology tools provide, such as the following:

- Interchanging ontologies by exporting them from a tool into an interchange language and then importing them into the other tool.
- Using RDF(S) as the interchange language, and serializing the ontologies into the RDF/XML syntax. A future benchmarking activity inside Knowledge Web will cover the case of using OWL as the interchange language.

The interoperability of ontology tools using an interchange language depends on the capabilities of the tools to import and export ontologies from/to this language. Therefore, the experimentation included not only the evaluation of the interoperability but also of the RDF(S) import and export functionalities.

The evaluation criteria must describe in depth the import, export and interoperability capabilities of the tools, whereas the experiments to be performed in the benchmarking must provide data explaining how the tools comply with these criteria. Therefore, to obtain detailed information about these capabilities, we need to know:

- The elements of the *internal knowledge model* of an ontology development tool that can be imported from RDF(S), exported to RDF(S) and interchanged with other tool, using RDF(S) as the interchange language.
- The *secondary effects* of importing, exporting, and interchanging these components, such as insertion or loss of information.
- The *subset of elements* of the internal knowledge models that these tools may use to interoperate correctly.

To obtain these experimentation data, we defined three benchmark suites that evaluate the capabilities of the tools (García-Castro et al., 2006), which were common for all the tools. Since the quality of the benchmark suites to be used is essential for the results, the first step was to agree on the definition of the suites. Then, we decided to make the import and export experiments before the interoperability one because the results of the first experiments affected those of the second.

A benchmark execution comprises (a) the definition of the expected ontology that results from importing, exporting or interchanging the ontology described in the benchmark, (b) the import, export, or interchange of the ontology defined in the benchmark, and (c) the comparison of the expected ontology with the imported, exported or interchanged ontology, checking whether there is some addition or loss of information. The steps to follow to execute the three benchmark suites are similar.

The benchmark suites were intended to be executed manually but, as they contained many benchmarks, it was highly recommended to execute them (or part of them) automatically. In the cases of Corese, Jena, Sesame, and WebODE, most of the experiment was automated. In the other cases, it was performed manually.

The benchmarking web page[i] contains the results of the experiments and a complete and

detailed description of (a) the benchmark suites, (b) all the files to be used in the experiments, and (c) the templates for collecting the results.

Benchmark Suites

The benchmark suites check the correct import, export and interchange of ontologies that model a simple combination of ontology components (classes, properties, instances, etc.). Because one of the goals of benchmarking is to improve the tools, the benchmark ontologies are kept simple on purpose in order to isolate the causes of the problems and to identify possible problems.

As the ontology tools that participated in benchmarking had different internal knowledge models, both the experimentation and the analysis of the results were based on a common group of ontology modelling primitives, available both in RDF(S) and in these tools. On the other hand, covering this common group exhaustively would yield a huge number of benchmarks; so we only considered the components most widely used for modelling ontologies in ontology development tools: classes, instances, properties with domain and range, literals, and class and property hierarchies. The remainder of the components has not been dealt with so far.

The **RDF(S) Import Benchmark Suite** contains 82 benchmarks[j], which define a simple RDF(S) ontology serialized in a RDF/XML file, which must be loaded into the ontology development tool.

In order to isolate the factors that affect the correct import of an ontology, we defined two types of import benchmarks: one that evaluates the import of the different combinations of components of the RDF(S) knowledge model, and the other type that evaluates the import of the different variants of the RDF/XML syntax, as stated in the RDF/XML specification.

Table 2 shows the categories of the RDF(S) Import Benchmark Suite, the number of benchmarks, and the components used. All the RDF(S) files to be imported can be downloaded from a single file; besides, templates are provided for collecting the execution results.

Table 2. Categories of the import benchmarks

Category	No.	Components used
Class	2	*rdfs:Class*
Metaclass	5	*rdfs:Class*, *rdf:type*
Subclass	5	*rdfs:Class*, *rdfs:subClassOf*
Class and property	6	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*
Property	2	*rdf:Property*
Subproperty	5	*rdf:Property*, *rdfs:subPropertyOf*
Property with domain and range	24	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*, *rdfs:domain*, *rdfs:range*
Instance	4	*rdfs:Class*, *rdf:type*
Instance and property	14	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*
Syntax and abbreviation	15	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*

The **RDF(S) Export Benchmark Suite** comprises 66 benchmarks[k], which describe an ontology that must be modelled in the tool and saved to a RDF(S) file.

We have defined two types of benchmarks for isolating the two factors that affect the correct export of an ontology: one type evaluates the correct export of the combinations of components of the ontology development tool knowledge model, and the other evaluates the export of ontologies using concepts and properties whose names have characters restricted by RDF(S), such as those characters that are forbidden when representing RDF(S) or XML URIs.

Table 3 shows the categories of the benchmark suite. The table contains the number of benchmarks and the components used in each category. Templates are also provided for collecting the execution results.

Since the factors that affect both the correct interchange of an ontology (besides the correct functioning of the importers and exporters) and the knowledge model (used for defining the ontologies) are the same as those that affect the RDF(S) Export Benchmark Suite, the ontologies described in the RDF(S) Interoperability Benchmark Suite are identical to those of the RDF(S) Export Benchmark Suite.

The evaluation criteria are common for the three benchmark suites, and are defined as follows:

- **Modelling** (*YES*/*NO*). The ontology tool can model the ontology components described in the benchmark.
- **Execution** (*OK*/*FAIL*). The execution of the benchmark is normally carried out seamlessly, and the tool always produces the expected result. However, when an execution fails, the following information is required:
  - The causes of the failure.
  - The changes performed if the tool had been previously corrected to pass a benchmark correctly.
- **Information added or lost.** The information added to or lost in the ontology interchange when executing the benchmark.

Table 3. Categories of the export benchmarks

Category	No.	Components used
Class	2	*class*
Metaclass	5	*class, instanceOf*
Subclass	5	*class, subClassOf*
Class and object property	4	*class, object property*
Class and datatype property	2	*class, datatype property, literal*
Object property	14	*object property*
Datatype property	12	*datatype property*
Instance	4	*class, instanceOf*
Instance and object property	9	*class, instanceOf, object property*
Instance and datatype property	5	*class, instanceOf, datatype property, literal*
URI character restrictions	4	*class, instanceOf, object property, datatype property, literal*

In the export and interoperability benchmark suites, if a benchmark describes an ontology that cannot be modelled in a certain tool, such benchmark cannot be executed in the tool, being the *Execution* result *N.E.* (Non Executed). However, in the import benchmark suites, even if a tool cannot model some components of the ontology, it should be able to import the rest of the components correctly.

Import and Export Results

The results obtained when importing from and exporting to RDF(S) depend mainly on the knowledge model of the tool that executes the benchmark suite. The tools that natively support the RDF(S) knowledge model (Corese, Jena and Sesame, essentially the RDF repositories) do not need to perform any translation in the ontologies when importing/exporting them from/to RDF(S). The RDF repositories import and export correctly all the combinations of components from/to RDF(S) because these operations do not require any translation.

In the case of tools with non-RDF knowledge models (KAON, Protégé and WebODE, the ontology development tools), some of their knowledge model components can also be represented in RDF(S), but some others cannot, and these tools do need to translate ontologies between their knowledge models and RDF(S). Finally, we must add that not all the combinations of components of the RDF(S) knowledge model that have been considered can be modelled into all the tools, as Table 4 shows.

Next, we present an analysis of the results of importing and exporting in the ontology development tools that participated in the benchmarking.

Table 4. Combinations of components modelled by the tools

Combination of components	RDF repos.	KAON	Protégé	WebODE
Classes	Y	Y	Y	Y
...instance of metaclasses	Y	Y	Y	N
Class hierarchies without cycles	Y	Y	Y	Y
...with cycles	Y	N	N	N
Datatype properties without domain or range	Y	Y	Y	N
...with multiple domains	Y	Y	N	N
...whose range is String	Y	Y	Y	Y
...whose range is a XML Schema datatype	Y	Y	N	Y
Object properties without domain or range	Y	Y	Y	N
...with multiple domains or ranges	Y	Y	N	N
...with a domain and range	Y	Y	Y	Y
Instances of a single class	Y	Y	Y	Y
...of multiple classes	Y	Y	Y	N
...related via object or datatype properties	Y	Y	Y	Y
...related via datatype properties whose range is a XML Schema datatype	Y	N	N	Y

Import Results

In general, the ontology development tools import correctly from RDF(S) all or most of the combinations of components that they model; they seldom add or lose information. The only exceptions are:

- Protégé, which presents problems, but only when it imports classes or instances that are instances of multiple classes.
- WebODE, which presents problems, but only when it imports properties with a XML Schema datatype as range.

When the tools import ontologies with combinations of components that they cannot model, they lose the information about these components. Nevertheless, these tools usually try to represent such components partially using for this other components from their knowledge models. In most cases, the importing is performed correctly. The only exceptions are:

- KAON, which causes problems when it imports class hierarchies with cycles.
- Protégé, which causes problems when it imports class and property hierarchies with cycles and properties with multiple domains.
- WebODE, which causes problems when it imports properties with multiple domains or ranges.

When dealing with the different variants of the RDF/XML syntax, we can observe that the ontology development tools

- Import correctly resources with the different URI reference syntaxes.
- Import correctly resources with different syntaxes (shortened and unshortened) of empty nodes, of multiple properties, of typed nodes, of string literals, and of blank nodes. The only exceptions are: KAON when it imports resources with multiple properties in the unshortened syntax; and Protégé when it imports resources with empty and blank nodes in the unshortened syntax. Do not import language identification attributes (*xml:lang*) in tags.

Export Results

In general, the ontology development tools export correctly to RDF(S) all or most of the combinations of components that they model with no loss of information. In particular:

- KAON causes problems only when it exports to RDF(S) datatype properties without range and datatype properties with multiple domains plus a XML Schema datatype as range.
- Protégé causes problems only when it exports to RDF(S) classes or instances that are instances of multiple classes and template slots with multiple domains.

When ontology development tools export components present in their knowledge model that cannot be represented in RDF(S), such as their own datatypes, they usually insert new information in the ontology, but they also lose some.

When dealing with concepts and properties whose names do not fulfil URI character restrictions, each ontology development tool behaves differently:

- When names do not start with a letter or "_", some tools leave the name unchanged, whereas others replace the first character with "_".
- Spaces in names are replaced by "-" or "_", depending on the tool.
- URI reserved characters and XML delimiter characters are left unchanged, replaced by "_", or encoded, depending on the tool.

Interoperability Results

The RDF repositories (Corese, Jena and Sesame) interoperate correctly between themselves, because they always import and export from/to RDF(S) correctly. This produces that the interoperability between the ontology development tools and the RDF repositories depends only on the capabilities of the former to import and export from/to RDF(S); therefore, the results of this interoperability are identical to those presented in the previous section.

The import and export results presented in previous sections indicate that some problems arise in the process of importing and exporting ontologies, whereas the interoperability results, on the other hand, show more problems.

As a general comment we can say that interoperability between the tools depends on

a. the correct functioning of their RDF(S) importers and exporters and
b. the way chosen for serializing the exported ontologies in the RDF/XML syntax.

Furthermore, we have observed that the problems affecting any of these factors also affect the results of not just one but several benchmarks. This means that, in some cases, to correct a single import or export problem, or to change the way of serializing ontologies can produce significant interoperability improvements.

Below we list the components that can be interchanged between the tools. These components are summarized in Table 5; each column of the table shows whether the combination of components can be interchanged between a group of tools[l]. The "-" character means that the component cannot

Table 5. Components interchanged between the tools

Combination of components	K-K	P-P	W-W	K-P	K-W	P-W	K-P-W
Classes	Y	Y	Y	Y	Y	Y	Y
...instance of a single metaclass	Y	Y	-	N	-	-	-
...instance of a multiple metaclasses	Y	N	-	N	-	-	-
Class hierarchies without cycles	Y	Y	Y	Y	Y	Y	Y
Datatype properties without domain or range	Y	Y	-	N	-	-	-
...with multiple domains	Y	-	-	-	-	-	-
...whose range is String	Y	Y	Y	N	N	Y	N
...whose range is a XML Schema datatype	Y	-	Y	-	Y	-	-
Object properties without domain or range	Y	Y	-	Y	-	-	-
...with multiple domains or ranges	Y	-	-	-	-	-	-
...with a domain and range	Y	Y	Y	Y	Y	Y	Y
Instances of a single class	Y	Y	Y	Y	Y	Y	Y
...of multiple classes	Y	N	-	N	-	-	-
...related via object properties	Y	Y	Y	Y	Y	Y	Y
...related via datatype properties	Y	Y	Y	N	Y	Y	N
...related via datatype properties whose range is a XML Schema datatype	-	-	Y	-	-	-	-

be modelled in some of the tools and, therefore, cannot be interchanged between them.

Interoperability Using the Same Tool

Ontology development tools seem to pose no problems when the source and the destination of an ontology interchange are the same tool. The only exception is Protégé when it interchanges classes that are instances of multiple metaclasses and instances of multiple classes; this is so because Protégé does not import resources that are instances of multiple metaclasses.

Interoperability between Each Pair of Tools

The interoperability between different tools varies depending on the tools. As the detailed interoperability results show, in some cases, the tools are able to interchange certain components from one tool to another, but not the other way round.

When **KAON** interoperates with **Protégé**, they can interchange correctly some of the common components that these tools are able to model. However, with components such as classes that are instances of a single metaclass or of multiple metaclasses, datatype properties without domain or range, datatype properties whose range is *String*, instances of multiple classes, and instances related through datatype properties, we have encountered some problems.

When **KAON** interoperates with **WebODE**, they can interchange correctly most of the common components that these tools can model, but when they interchange datatype properties with domain and whose range is *String*, the results are not the same.

When **Protégé** interoperates with **WebODE**, they can interchange correctly all the common components that these tools can model.

Interoperability between All the Tools

Interoperability between **KAON**, **Protégé** and **WebODE** can be achieved by most of the common components that these tools can model. The only components that these tools cannot use are datatype properties with domain and whose range is *String*, and instances related through datatype properties.

Therefore, interoperability was achieved among the tools that participated in the benchmarking by using classes, class hierarchies without cycles, object properties with domain and with range, instances of a single class, and instances related through object properties.

Interoperability Regarding URI Character Restrictions

Interoperability is low when tools interchange ontologies containing URI character restrictions in class and property names. This is so because tools usually encode some or all the characters that do not comply with these restrictions, which provokes changes in class and property names.

Recommendations

Recommendations for Ontology Engineers

This section offers recommendations for ontology engineers which use more than one ontology tool to build ontologies. Depending on the tools used, the level of interoperability may be higher or lower, as can be seen in Section 5.4.

If the ontology is being developed bearing in mind interoperability, ontology engineers should be aware of the components that can be represented in the ontology development tools and in RDF(S). And they should try to use the common knowledge

components that these tools have so as to avoid the knowledge losses commented above.

Ontology engineers should also be aware of the semantic equivalences and differences between the knowledge models of the tools and the interchange language. For example, in Protégé, multiple domains in template slots are considered the union of all the domains, whereas in RDF(S) multiple domains in properties are considered the intersection of all the domains; in WebODE, on the other hand, instance attributes are local to a single concept, whereas in RDF(S) properties are global and can be used in any class.

It is not recommended to name resources using spaces or any character that is restricted in the RDF(S), URI, or XML specifications.

When the RDF repositories interoperate, even though these repositories export and import correctly to RDF(S), ontology engineers should consider the limitations that other tools have when they export their ontologies to RDF(S).

Recommendations for Tool Developers

This section includes general recommendations for improving the interoperability of the tools while developing them. In (García-Castro et al., 2006), we offer full detailed recommendations regarding the results and practices gathered to improve each of the participant tools. Although it is not compulsory to follow these recommendations, they help correct interoperability problems as we could observe when we analysed the results.

The interoperability between ontology tools (using RDF(S) as the interchange language) depends on how the importers and exporters of these tools work; on the other hand, how these importers and exporters work depends on the development decisions made by the tool developers, and these are different people with different needs. Therefore, to provide general recommendations for developers is not straightforward, though some comments can be extracted from the analysis of the benchmarking results.

In some occasions, a development decision will produce interoperability improvement with some tools and interoperability loss with others. For example, when exporting classes that are instances of a metaclass, some tools require that the class be defined as instance of *rdfs:Class*, whereas other tools require the opposite.

Tool developers, therefore, should analyze the collateral consequences of the development decisions. Thus, if a datatype is imported as a class in the ontology, then the literal values of this datatype should be imported as instances in the ontology, which would complicate the management of these values.

They also should be aware of the semantic equivalences and differences between the knowledge models of their tool and the interchange language; on the other hand, the tools should notify the user when the semantics is changed.

The first requirement for achieving interoperability is that the importers and exporters of the tools be robust and work correctly when dealing with unexpected inputs. Although this is an obvious comment, the results show that this requirement is not achieved by the tools and that some tools even crash when importing some combinations of components.

Above all, tools should deal correctly with the combinations of components that are present in the interchange language but cannot be modelled in them. For example, although cycles in class and property hierarchies cannot be modelled in some ontology development tools, these tools should be able to import these hierarchies by eliminating the cycles.

If developers want to export components that are commonly used by ontology development tools, the components should be completely defined in the file. This means that metaclasses and classes in class hierarchies should be defined as instances of *rdfs:Class*, properties should be defined as instances of *rdf:Property*, etc.

Exporting complete definitions of other components can cause problems if these are imported

by other tools. And not every tool deals with datatypes defined as instances of *rdfs:Datatype* in the file, or with *rdf:datatype* attributes in properties.

If the document does not define a default namespace, every exported resource should have a namespace.

CONCLUSION

This chapter states the need to evaluate and benchmark the Semantic Web technology and provides some references that can be helpful in these activities. It also presents the authors' approach to software benchmarking and compares it with other existing evaluation and benchmarking approaches.

We have tried to explain how the benchmarking methodology can help assess and improve software, whereas the use of benchmark suites is advisable when performing evaluations in benchmarking.

One of the strong points we make on benchmarking is its community-driven approach. Benchmarking should be performed by the experts of the community since the benefits obtained after performing it affect the whole community.

Benchmarking does not imply comparing the results of the tools but comparing the practices that lead to these results. Therefore, experimentation should be designed to obtain these practices as well as the results.

However, as we have seen, benchmarking is not the solution to every case. In a preliminary step, developers would have to asses whether benchmarking is the correct approach; as benchmarking is useful when the goals are to improve the software and to extract the practices performed by others.

Benchmarking is an activity that takes long time to perform because it requires tasks that are not immediate: announcements, agreements, etc. Therefore, benchmarking activities should start early in time, and the benchmarking planning should consider a realistic duration of the benchmarking and the resources needed for carrying them out.

We have also shown how we have applied the benchmarking methodology to a concrete case in the Semantic Web area: interoperability of ontology development tools using RDF(S) as interchange language.

Providing benchmark suites in the benchmarking allows evaluating other tools with RDF(S) import and export capabilities without their having to participate in the benchmarking; this can be useful both while the tools are being developed and afterwards, when their development has finished. In addition, the benchmarking results can be used by ontology development tool users that may find problems when interchanging ontologies or may want to foresee the results of a future interchange.

Although it is not required that the tool developers participate in the benchmarking and perform the experiments over their tool, their involvement facilitates the execution and analysis of the experimentation results to a large extent. In all the cases where tool developers carried out the experimentation over their own tools, a great improvement occurred before the *Improve* phase of the methodology because developers were able to detect problems and correct their tools while executing the benchmark suites.

We have observed that the manual execution of the experiments and the analysis of the results cause the benchmark suite to be costly. Consequently, tool developers often automate the execution of the benchmark suites, but not always. Another drawback of the manual execution of experiments is that the results obtained depend on the people performing these experiments, on their expertise with the tools, and on their ability to extract the practices performed.

FUTURE RESEARCH DIRECTIONS

As shown in Section 3, current evaluation and benchmarking activities over the Semantic Web technology are scarce and a hindrance to the full development and maturity of this technology. The Semantic Web needs to produce methods and tools for evaluating the technology at great scale and in an easy and economical way. This requires defining technology evaluations focusing on their reusability.

In the last few years, evaluation and benchmarking efforts have mainly focused on some types of technologies and on some of their metrics, namely, the interoperability of ontology development tools, the precision and recall of ontology alignment tools, and the efficiency and scalability of ontology repositories. But now we think that new efforts are required, first, to involve other Semantic Web technologies (ontology learning tools, ontology annotation tools, ontology population tools, etc.) and, second, to broaden the scope of these evaluations by considering a wider range of evaluation metrics for the technology (latency, robustness, security, usability, etc.).

The role of the research community when defining and performing benchmarking activities is crucial. Community-driven benchmarking connects experts and allows obtaining high quality results and increasing the credibility of the benchmarking and its results.

However, future research must focus on performing evaluations centred on the user of the Semantic Web technology. And it would be advisable to consider audiences from beyond the research community itself as recipients of the evaluation results.

REFERENCES

Abran, A., Moore, J. W., Bourque, P., & Dupuis, R. (Ed.). (2004). *SWEBOK: Guide to the software engineering body of knowledge*. IEEE Press.

Basili, V. R., & Selby, R. W. (1991). Paradigms for experimentation and empirical studies in software engineering. *Reliability Engineering and System Safety, 32*, 171-191.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions on Software Engineering, 12*(7), 733-743.

Basili, V. R. (1985, September). Quantitative evaluation of software methodology. In *1st Pan-Pacific Computer Conference*, Melbourne, Australia.

Bull, J. M., Smith, L. A., Westhead, M. D., Henty, D. S., & Davey, R. A. (1999). A methodology for benchmarking Java grande applications. In the *ACM 1999 conference on Java Grande* (pp. 81-88).

Camp, R. (1989). *Benchmarking: The search for industry best practices that lead to superior performance*. Milwaukee, ASQC Quality Press.

Freimut, B., Punter, T., Biffl, S., & Ciolkowski, M. (2002). *State-of-the art in empirical studies*. Technical Report ViSEK/007/E, Visek.

García-Castro, R., & Gómez-Pérez, A. (2005, November). Guidelines for benchmarking the performance of ontology management APIs. In Y. Gil, E. Motta, R. Benjamins, & M. Musen (Ed.), *4th International Semantic Web Conference (ISWC2005), 3729* in LNCS, 277-292. Galway, Ireland: Springer-Verlag.

García-Castro, R., Maynard, D., Wache, H., Foxvog, D., & González-Cabero, R. (2004). *D2.1.4 Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools*. Technical report, Knowledge Web.

García-Castro, R., Sure, Y., Zondler, M., Corby, O., Prieto-González, J., Paslaru Bontas, E., Nixon, L., & Mochol, M. (2006). *D1.2.2.1.1 Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language*. Technical report, Knowledge Web.

Gediga, G., Hamborg, K., & Duntsch, I. (2002). Evaluation of software systems. In *Encyclopedia of Computer Science and Technology, 44,* 166-192.

IEEE. (1998) *IEEE Std 1061-1998 IEEE Standard for a software quality metrics methodology.*

ISO/IEC (1999) *ISO/IEC 14598-1: Software product evaluation - Part 1: General overview.*

Juristo, N., & Moreno, A. (2001). *Basics of software engineering experimentation*. Kluwer Academic Publishers.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones P. W., Hoaglin, D. C., El-Emam, K., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in Software Engineering. *IEEE Transactions on Software Engineering 28*(8), 721-734.

Kitchenham, B. (1996). *DESMET: A method for evaluating Software Engineering methods and tools.* Technical Report TR96-09, Department of Computer Science, University of Keele, Staffordshire, UK.

Park, R. E., Goethert, W. B., & Florac, W. A. (1996). *Goal-driven software measurement - a guidebook*. Technical Report CMU/SEI-96-HB-002, Software Engineering Institute.

Perry, D. E., Porter, A. A., & Votta L. G. (2000). Empirical studies of Software Engineering: a roadmap. In A. Finkelstein (Ed.), *The Future of Software Engineering,* 345-355. ACM Press.

Rakitin, S. R. (1997). *Software Verification and Validation, a practitioner's guide.* Artech House.

Shirazi, B., Welch, L. R., Ravindran, B., Cavanaugh, C., Yanamula, B., Brucks, R., & Huh, E. (1999). Dynbench: A dynamic benchmark suite for distributed real-time systems. In the *11th IPPS/SPDP'99 Workshops,*1335-1349. Springer-Verlag.

Sim, S., Easterbrook, S., & Holt, R. (2003). Using benchmarking to advance research: A challenge to software engineering. In the *25th International Conference on Software Engineering (ICSE'03),*74-83. Portland, OR.

Spendolini, M. J. (1992). *The benchmarking book.* New York, NY: AMACOM.

Stefani, F., Macii, D., Moschitta, A., & Petri, D. (2003, June). FFT benchmarking for digital signal processing technologies. In the *17th IMEKO World Congress.* Dubrovnik, Croatia.

Sure, Y., & Corcho, O. (Ed.) (2003). Proceedings of the *2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), 87* of CEUR-WS. Florida, USA.

Weiss, A. R. (2002). *Dhrystone benchmark: History, analysis, scores and recommendations.* White paper, EEMBC Certification Laboratories, LLC.

Wireman, T. (2003). *Benchmarking best practices in maintenance management.* Industrial Press.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., & Ciolkowski, M. (2002, June). Software inspection benchmarking - a qualitative and quantitative comparative opportunity. In the *8th International Software Metrics Symposium,* 118-130.

ADDITIONAL READING

Ahmed, P., & Rafiq, M. (1998). Integrated benchmarking: a holistic examination of select techniques for benchmarking analysis. *Benchmarking for Quality Management and Technology* 5, 225-242.

Basili, V. R., Caldiera, G., & Rombach, D. H. (1994). The Goal Question Metric approach. *Encyclopedia of Software Engineering,* 528-532. Wiley.

Basili, V. R. (1993). The experimental paradigm in Software Engineering: Critical assessment and future directions. In the *International Workshop on Experimental Software Engineering Issues*, 3-12. Springer-Verlag.

Beitz, A., & Wieczorek, I. (2000). Applying benchmarking to learn from best practices. *Product Focused Software Process Improvement, Second International Conference (PROFES 2000)*, 59-72.

Brickley, D., Guha, R. V. (Ed.) (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation 10 February 2004.

Brown, A., & Wallnau, K. (1996). A framework for evaluating software technology. *IEEE Software, 13*, 39-49.

Corcho, O. (2005). *A layered declarative approach to ontology translation with knowledge preservation.* Volume 116 of Frontiers in Artificial Intelligence and its Applications. IOS Press.

Dongarra, J., Martin, J. L., & Worlton, J. (1987). Computer benchmarking: paths and pitfalls. *IEEE Spectrum, 24*(7), 38-43.

Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B., & Benjamins V. R. (1999). Wondertools? A comparative study of ontological engineering tools. In the *12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada: Kluwer Academic Publishers.

Dujmovic, J. J., (1998). Evaluation and design of benchmark suites. Chapter 12 in *State-of-the-art in performance modeling and simulation: theory, techniques and tutorials*,287-323. Gordon and Breach Publishers.

Feitelson, D. G. (2005). *Experimental computer science:The Need for a Cultural Change*.

Fenton, N. (1991). *Software metrics - a rigorous approach*. Chapman & Hall.

Fenton, N., & Neil, M. (2000). Software metrics: Roadmap. In the *Conference on the future of Software Engineering*, 357-370. ACM Press.

Fernandez, P., McCarthy, I., & Rakotobe-Joel, T. (2001). An evolutionary approach to benchmarking. *Benchmarking: An International Journal, 8*, 281-305.

García-Castro, R. (2006). Benchmarking como herramienta de transferencia tecnológica Invited talk in the *3er Encuentro Internacional de Investigadores en Informática*. Popayán, Colombia.

García-Castro, R. (2006). Keynote: Towards the improvement of the Semantic Web technology. In the *Second International Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2006)*.

García-Castro, R., & Gómez-Pérez, A. (2006). Benchmark suites for improving the RDF(S) importers and exporters of ontology development tools. In the *3rd European Semantic Web Conference (ESWC2006)*, 155-169. LNCS-4011.

García-Castro, R., Gómez-Pérez, A. (2006). Interoperability of Protégé using RDF(S) as interchange language. In the *9th International Protégé Conference*.

García-Castro, R. (2006) Keynote: Tecnologías de la Web Semántica: cómo funcionan y cómo interoperan. In the *4th Seminario Internacional Tecnologías Internet*. Popayán, Colombia.

García-Castro, R., & Gómez-Pérez, A. (2005). A method for performing an exhaustive evaluation of RDF(S) importers. In the *Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2005)*.

García-Castro, R. (2005). *D2.1.5 prototypes of tools and benchmark suites for benchmarking ontology building tools.* Technical report, Knowledge Web.

Gee, D., Jones, K., Kreitz, D., Nevell, S., O'Connor, B., & Ness, B. V. (2001). Using performance information to drive improvement. *Performance-Based Management Special Interest Group, 6*.

Goodman, P. (1993). *Practical implementation of software metrics.* McGraw Hill.

Grady, R., & Caswell, D. (1987). *Software metrics: Establishing a company-wide program*. Prentice-Hall.

Jones, C. (1995, October). Software benchmarking. *IEEE Computer*, 102-103.

Kitchenham, B., Linkman, S., & Law, D. (1994). Critical review of quantitative assessment. *Software Engineering Journal*, *9*, 43-53.

Kitchenham, B., Pfleeger, S., & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering, 21*, 929-944.

Kraft, J. (1997). *The Department of the Navy benchmarking handbook: a systems view.* Department of the Navy.

Lankford, W. (2000). Benchmarking: understanding the basics. *Coastal Business Journal.*

Lukowicz, P., Tichy, W. F., Prechelt, L., & Heinz E.A. (1995). Experimental evaluation in computer science: A quantitative study. *The Journal of Systems and Software*, 28(1),1-18.

Manola, F., & Miller, E. (2004, February 10). *RDF Primer.* W3C Recommendation.

OntoWeb (2002). *D1.3: A survey on ontology tools.* Technical report, IST OntoWeb Thematic Network.

Pfleeger, S. L. (1999). Understanding and improving technology transfer in software engineering. *Journal of Systems and Software* 47,111-124.

Sim, S. (2003). *A theory of benchmarking with applications to software reverse engineering.* PhD thesis. University of Toronto.

Sole, T., & Bist, G. (1995). Benchmarking in technical information. *IEEE Transactions on Professional Communication* 38, 77-82.

Tichy, W. (1998). Should computer scientists experiment more? *Computer* 31, 32-40.

Wache, H., Serafini, L., & García-Castro, R. (2004). *D2.1.1 survey of scalability techniques for reasoning with ontologies.* Technical report, KnowledgeWeb.

Wireman, T. (2003). *Benchmarking best practices in maintenance management.* Industrial Press.

ENDNOTES

[a] http://knowledgeweb.semanticweb.org/

[b] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/

[c] http://kaon.semanticweb.org/

[d] http://protege.stanford.edu/

[e] http://webode.dia.fi.upm.es/

[f] http://www-sop.inria.fr/acacia/soft/corese/

[g] http://jena.sourceforge.net/

[h] http://www.openrdf.org/

[i] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/

[j] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/rdfs_import_benchmark_suite.html

[k] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/rdfs_export_benchmark_suite.html

[l] The names of the tools have been shortened in the heading of the table: KAON=K, Protégé=P and WebODE=W.

APPENDIX: QUESTIONS FOR DISCUSSION

Beginner:

1. Which are the main characteristics of benchmarking?
2. Which is the goal of the *Recalibration* task in the benchmarking methodology?
3. Which are the factors that influence the correct interchange of an ontology between two Semantic Web tools?
4. When exporting one ontology from an ontology development tool to RDF(S) having interoperability in mind, is it advisable to export the complete definition of all its components?

Intermediate:

1. Which are the differences between evaluation and benchmarking?
2. Are the users of the software involved in the benchmarking?
3. Is management support needed in the *Improve* phase of the methodology?
4. Which RDF(S) components can be represented in KAON, Protégé and WebODE?

Advanced:

1. Which resources are needed for performing a benchmarking activity?
2. Why are there three different evaluation criteria to define the results of the RDF(S) Import, Export and Interoperability benchmark suites?
3. Why is it not enough to have a single ontology representation language to achieve interoperability between the Semantic Web technologies?

Practical Exercises:

1. Select one conference paper that presents some evaluation or benchmarking approach and then evaluate its reusability according to the desirable properties of a benchmark suite and the recommendations given for software evaluation in benchmarking activities.
2. Create a mid-size ontology using one ontology development tool. Can you anticipate the consequences of exporting that ontology to RDF(S)? And of importing it into another ontology development tool?
3. Then, export the ontology to RDF(S). Was your prediction correct? Has it had information addition or loss?
4. Finally, import the exported ontology into the other ontology development tool. Was your prediction correct? Has it had information addition or loss?

ANSWERS TO THE QUESTIONS FOR DISCUSSION

Beginner:

1. The main characteristics of benchmarking are continuous improvement and the search for best practices.
2. The goal of the *Recalibration* task is to improve the benchmarking process by recalibrating it and applying the lessons learnt while performing it.
3. The factors that influence the correct interchange of an ontology between two tools are the combinations of components of the knowledge model of the ontology development tool and the naming of the components present in the ontology.
4. It is advisable to export the complete definition of all its components only for

components commonly used by ontology development tools.

Intermediate:

1. Benchmarking is a continuous process, whereas an evaluation is a punctual activity. In addition, benchmarking involves evaluating software but its goals are to obtain a continuous improvement on the software and the practices used when developing the tools.
2. Yes, the users of the software are identified in the *Participant identification* task and in the *Findings communication* task.
3. Yes, it is needed to implement the necessary changes in the software and in the organisation processes affecting the software.
4. Classes, class hierarchies without cycles, datatype properties with a class as a domain and a string range, object properties with a domain and a range, instances of a single class, instances related by object properties, and instances related by datatype properties with a string range.

Advanced:

1. The resources needed are human resources though some equipment and travel resources are also required, and these are mainly used in three tasks: benchmarking organisation, experimentation definition and execution, and result analysis.
2. Because these three evaluation criteria are necessary to represent the different situations and behaviours that can occur when two tools interchange one ontology.
3. Because different types of users need different tools; existing tools have different knowledge representation models; and tools need to translate their ontologies from their knowledge models to the common ontology representation language.

This work was previously published in Semantic Web Engineering in the Knowledge Society, edited by J. Cardoso & M. Lytras, pp. 341-370, copyright 2009 by Information Science Reference (an imprint of IGI Global).

Chapter 8.18
All-Optical Internet: Next-Generation Network Infrastructure for E-Service Applications

Abid Abdelouahab
Multimedia University, Malaysia

Fouad Mohammed Abbou
Alcatel Network Systems, Malaysia

Ewe Hong Tat
Multimedia University, Malaysia

Toufik Taibi
United Arab Emirates University, UAE

ABSTRACT

To exploit the unprecedented opportunities offered by the E-Service Applications, businesses and users alike would need a highly-available, reliable, and efficient telecommunication infrastructure. This chapter provides an insight into building the next-generation network infrastructure, that is, the All-Optical Internet. It also reveals the factors driving the convergence of the Internet Protocol (IP) and the Wavelength-Division Multiplexing (WDM) technology. The chapter discusses the dominant optical networks architectures in an attempt to show the evolution towards the ultimate all-optical packet-switching network. A special focus is given to the Optical Burst Switching (OBS) as a new emerging switching paradigm and a highly promising technology. OBS network architecture, burst assembly, signaling and reservation protocols, QoS support, and contention resolution techniques are presented. Furthermore, realistic suggestions and strategies to efficiently deploy OBS are given.

INTRODUCTION

The Internet is a technology with many properties that has the potential to influence, and even transform established ways of conducting business, especially Electronic Services, while at the same time creating entirely new industries and businesses. The Internet has a profound impact on the competitive landscape, since it is affecting the way that firms' activities are coordinated, how commerce is conducted, how business communities are created, and how communications are defined and performed.

As we are moving towards the Web-dependent era, the 99 percent or even the 99.99 percent network reliability would be inadequate for the mission-critical applications that have genuine requirements that exceed the typical application needs.

The e-service applications are typically reliant on IP data networks that construct the Internet, which has become a ubiquitous success. Furthermore, the capacity of optical fibers is doubling annually toward a terabit per second per fiber, providing strong incentives to exploit the huge bandwidth of fiber-optic networks, which has increased considerably with the introduction of Wavelength-Division Multiplexing (WDM) technology.

The rapid advancement of optical technologies and the growing effort to enhance the Internet Protocol (IP) makes it possible to move from the current network architecture to an all-optical Internet, where the network traffic is optically transmitted, buffered, amplified, and switched through high performance Internet switches and routers directly connected using WDM optical links.

ALL-OPTICAL INTERNET

The Internet is an interconnection of computer networks that are a combination of hardware and software, controlled by a set of protocols to transmit and communicate data. The Internet uses the TCP/IP protocol suite, where the Transmission Control Protocol (TCP) is a connection-oriented end-to-end protocol. TCP is used to create logical connections between various applications running on different hosts, prior to executing protocols that exchange information. TCP relies on the Internet Protocol (IP) to route the packets (data units that are routed between an origin and a destination) through the network. Therefore, the Internet is simply a massive network of networks.

Depending on the deployed physical technology, three network generations can be distinguished. The first-generation networks are based on copper wire or radio; subsequently; the copper wire (around 10 gigabits per second) was replaced by a more sophisticated transmission medium, the optical fiber, which offers an immense bandwidth (theoretically, 50 terabits per second), low error rate, high reliability, availability, and maintainability. Additionally, optical fibers feature many other advantages, for example, lightweight and small space requirements, resistance to corrosive materials, less signal attenuation, and high immunity to tapping. Having optical fiber as the transmission medium in the second-generation networks enhanced the network performance and throughput; however, this improvement was restricted by the so-called Electronic Bottleneck. Electronic bottleneck phenomenon is caused by the limited processing speed of electronic components (a few gigabits per second) deployed in switches/routers, multiplexers, and end-nodes in the network. As the electronic processing speed is at its peak, the solution is to transfer the switching and routing functions from the electronic domain to the optical domain. Therefore, the third-generation networks will be designed as all-optical networks, where the data are transmitted all-optically through the network from source to destination.

IP and WDM Convergence

The success of the Internet and its omnipresence has made the Internet Protocol (IP) a de facto standard, and the demand for IP network solutions has increased exponentially. With the Internet protocol, voice and video as well as real-time multimedia traffic could be integrated and transmitted with the data traffic over a single network infrastructure, providing unprecedented opportunity for businesses to improve their services, reduce expenses, and increase their revenues. With the technological evolution, IP diminished the boundaries between the computing, entertainment, and the telecom industries, which led to more customer centric business models. Being an intermediate layer between an immense variety of IP-based services and almost all the layer-2 technologies, IP is more than a revenue-generating integration layer. IP convergence is not merely an option; it is inevitably becoming a business necessity in the Internet-dependent era and the new customer-centric economy. Such convergence is best supported by the WDM technology. WDM, which is conceptually similar to Frequency Division Multiplexing (FDM), is a method of transmitting many light beams of different wavelengths simultaneously down the core of a single optical fiber. Therefore, with WDM the available bandwidth can be increased without deploying additional optical fibers. Furthermore, WDM relieves the effect of electronic bottleneck by dividing the optical transmission spectrum into a number of non-overlapping channels; thus, the electronic components would only work at their own speed (few gigabits per second) and not at the aggregated transmission rate. IP/WDM convergence will eventually be translated to an efficient, robust, reliable, and feasible all-optical Internet.

Optical Switching Paradigms

Several different switching paradigms have been devised to transfer data over WDM, such as Optical Circuit Switching (OCS), a wavelength-routed network (Mukherjee, 1997), Optical Packet Switching (OPS) (Xu, Perros, & Rouskas, 2001; Yao, Dixit, & Mukherjee, 2000), and Optical Burst Switching (OBS) (Qiao & Yoo, 1999). Wavelength-routed optical networks (currently deployed) represent a promising technology for optical networks. However, wavelength-routed optical networks, which employ circuit switching, may not be the optimal choice and the most appropriate switching paradigm to support the various requirements of the Internet traffic. Alternatively, optical packet switching paradigm appears to be the optimum option. Unfortunately, OPS is not mature enough to provide a viable solution. Therefore, a switching technique that provides granularity in between wavelengths and packets was shaped, thus occupying the middle of the spectrum between circuit switching and packet switching paradigms. Optical burst switching is a switching technique where the benefits of both packet-switching networks and circuit-switching networks are combined; OBS borrows ideas from both to deliver a completely new functionality.

Optical Circuit Switching

In circuit switching, each sender and receiver are connected with a dedicated communication path (circuit), established through one or more intermediate switching nodes. The communication path must be set up between the two end-nodes before they could communicate. Although circuit switching is the most familiar technique used to build communication networks, and even though wavelength-routed optical networks have already been deployed, circuit switching may not be the most appropriate switching technique for the emerging

optical Internet. Circuit-switched networks lack flexibility and convergence capabilities, which lead to poor wavelength utilization, especially if the connection holding time (i.e., the time of data exchange) is very short; the situation is worsened by the bursty nature of the data traffic. Therefore, in order to fully utilize the WDM wavelengths, a sophisticated traffic grooming mechanism is needed to statistically multiplex the data coming from different sources.

Optical Packet Switching

Optical packet switching is an alternative optical switching paradigm that appears to be the optimum choice. However, with the immature current optical technology, it is impossible (at least in short-medium term) to deploy OPS networks viably. Packet switches usually work synchronously, where the arriving packets at different input ports must be aligned before being switched using the switching fabrics. This process is difficult and expensive to be implemented fully in the optical domain. Another operation concern is the relatively long time in which the optical-switching fabrics are configured compared to the data transmission speed. This is clearly demonstrated using the following example based on the current developments in the optical technology, considering an optical packet switch, with a switching fabric that takes 1 *ms* to be configured (i.e., set up a connection from an input port to an output port). At a data rate of 2.5 Gbps, it takes about 5 μs to transmit a 12,500-bit packet. Therefore, if a switch operates at the packet level, then less than 0.5% of the time is used for switching data, while the rest is wasted in setting up the switching fabric.

A more prevailing OPS's deployment issue, is the realization of the "Store and Forward" concept that is the requisite idea in packet switching networks. To store packets in switches, a buffering strategy is needed to resolve contentions in the output ports. Optical switches currently use optical fibers as Fiber Delay Lines (FDL) to emulate buffers by delaying packets for a fixed time (Xu et al., 2001). The FDLs are far away behind the random access electronic buffers, in term of cost, performance, and deployment simplicity.

Optical Burst Switching

The concept of burst switching first emerged in the context of voice communications. OBS is an adaptation of a stander known as ATM Block Transfer (ABT) developed by the telecommunication standardization sector of the International Telecommunication Union (ITU-T) for burst switching in Asynchronous Transfer Mode (ATM) networks. OBS consists of core nodes, built from optical and electronic components, and edge (Ingress/ Egress) nodes connected by WDM links. OBS differs from optical packet switching and the original burst-switching concept introduced in the 1980's (Amstutz, 1989; Kulzer & Montgomery, 1984) in a main aspect, that is, the separation of the control and the data payloads, both in time (i.e., the control information is transmitted at an offset time prior to its corresponding data payload) and physical space (i.e., the control packets and the data propagate on different designated channels). In conventional OBS, there is no need for buffering and electronic processing of data. Furthermore, OBS insures efficient bandwidth utilization, since the bandwidth is reserved only when the data is to be transferred through the link.

Optical Components

In order to build an all-optical Internet, different functional optical elements are required. These elements could be classified either as "switching" or "non-switching" components; "switching" components enable networking, while "non-switching" components are primarily used on optical links. However, a simple classification could be determined by the placement of the optical elements in the system. They can be placed

either on the transmitting-end, on the link, or on the receiving-end. In this section, the key optical communications elements are briefly described.

- **Optical fibers:** Optical fibers are made of glass or plastic, and they transmit signals in the form of light at about two-thirds the speed of light in a vacuum. In the optical fibers, the lightwaves (signals) are guided with a minimum attenuation.
- **Light sources and light detectors:** At both ends of an optical transmission system, light sources and light detectors are found. At the transmitting-end, the light source (light emitter) modulates electrical signals into light pulses. The opposite process, that is, to modulate light pulses into electrical signals, is performed by the light detectors at the receiving-end.
- **Optical Amplifiers:** To amplify an attenuated optical signal, unlike the conventional repeaters that perform respectively optical-electrical conversion, electrical amplification, and then electrical-optical conversion, the optical amplifiers operate completely in the optical domain. The optical amplifiers can boost the power levels of the optical signals. Amplifiers can be classified to two fundamental amplifier types: Semiconductor Optical Amplifiers (SOAs) and Doped-Fiber Amplifiers (DFAs).
- **Multiplexers and De-multiplexers:** Multiplexers and De-multiplexers are essential components in the optical transmission systems. Multiplexers converge and combine multiple incoming signals into one beam. The de-multiplexers have the ability to separate out the combined components of the beam and discreetly detect the different signals.
- **Optical Add/Drop Multiplexers:** Optical Add/Drop Multiplexers (OADMs) are key elements on the progress toward the ultimate goal of all-optical networks. Unlike the multiplexers and de-multiplexers, the OADMs are capable of adding or removing one or more wavelengths without the need for combining or separating all wavelengths.
- **Wavelength converters:** An optical wavelength converter is a device capable of directly translating (converting) data contained on an incoming wavelength to another wavelength without optical-electrical-optical conversion. Therefore, wavelength converters are very important elements in the deployment of all-optical networks. Two classes of wavelength converters can be identified: Optical-Gating Wavelength Converters and Wave-Mixing Wavelength Converters.

OPTICAL BURST SWITCHING

In this chapter, a comprehensive discussion of Optical Burst Switching (OBS) is presented. The OBS networks variants, reservation protocols, node architecture and switching technology, Quality of Service provisioning, and contention resolution techniques are discussed.

OBS Network Functionality

Optical Burst Switching is a relatively new switching technique and is still at the definition phase, which is clearly indicated by the number of research groups and their publications, specifically, on new OBS architectures (Callegate, Cankaya, Xiong, & Vandenhoute, 1999; Dolzer, 2002; Verma, Chaskar, & Ravikanth, 2000; Xu et al., 2001; Yoo, Qiao, & Dixit, 2001), prototypes (Baldine, Rouskas, Perros, & Stevenson, 2002), reservation mechanisms (Detti & Listanti, 2001; Dolzer, Gauger, Spath, & Bodamer, 2001; Qiao & Yoo, 2000; Tancevski, Yegnanarayanam, Castanon, Tamil, Masetti, & McDermott, 2000; Turner, 1999; Wei, Pastor, Ramamurthy, & Tsal, 1999; Xiong, Vanderhoute, & Cankaya, 2000;

Yoo, Jeong, & Qiao, 1997) and assembly mechanisms (Ge, Callegati, & Tamil, 2000; Hu, Dolzer, & Gauger, 2003; Vokkarane, Haridoss, & Jue, 2002). However, despite the fact that there is no standard architecture or a universal definition of optical burst switching, OBS networks assume the following general characteristics:

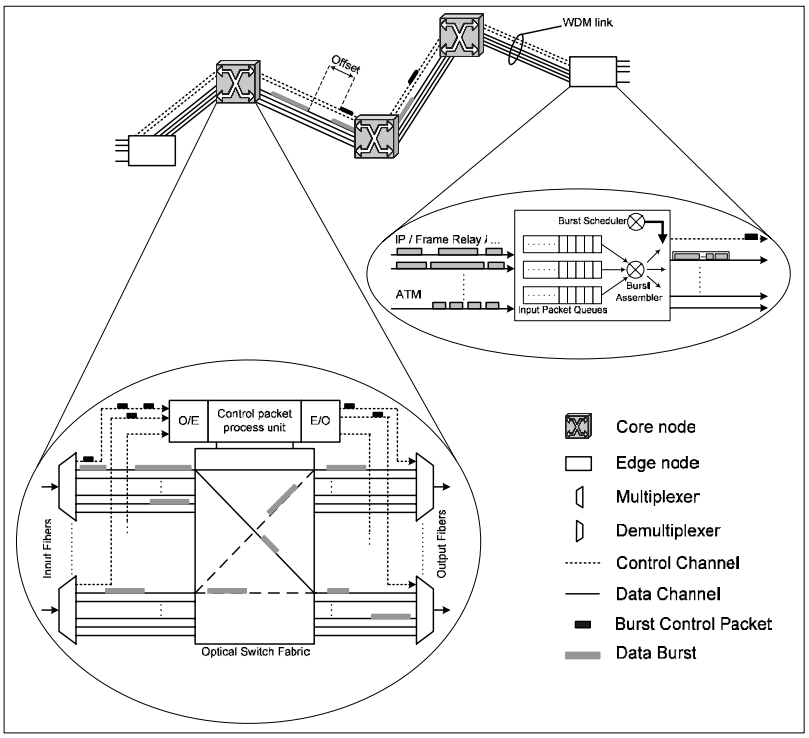
- **Granularity:** The transmission units (called bursts) are macro-packets, each of which should have a duration that is as short as possible, as it does not require a dedicated channel, but long enough to be efficiently switched optically; that is, between the optical circuit switching and optical packet switching granularity.
- **Separation of control and data:** Control information and data payload propagate on separate channels (wavelengths).
- **One-way reservation:** Network resources are generally allocated in one-way reservation fashion. That is, bursts (data) are transmitted at an offset time after their control information without receiving or waiting for any acknowledgements, to confirm the switching resources reservation, back from the destination nodes.
- **Variable burst length:** The duration (size) of the data burst may be variable. However, the burst duration should be long enough to be efficiently switched in the optical domain, and short enough to be statistically multiplexed.
- **Optical Buffering/Wavelength conversion:** In the conventional implementation of OBS networks, optical buffers/wavelength converters are not mandatory in the intermediate nodes. Thus, the data bursts would be switched optically from source to destination, without any delay (buffering), and using the same wavelength.

Not all of these characteristics need be satisfied at once and without variation in all OBS implementations, thus, allowing for design flexibility and a possible smooth evolution towards the packet switching technique, when the development on the optical switching technology matures.

As mentioned before, the model of burst switching is not new as a concept. Burst-switching has been already known since the 1980's. However, the concept has never been implemented widely and successfully in conventional electrical networks. Due to its complexity and high cost, burst switching could not compete with established, flexible, and relatively cheap electronic packet switching techniques (e.g., Asynchronous Transfer Mode (ATM)). However, with the introduction of very high-speed optical transmission techniques, this has changed. It has been agreed that it is more advantageous, cost effective, and efficient to keep data in the optical domain and to avoid optical-electrical-optical conversions. In view of the fact that all-optical packet switching techniques are still too complex to be widely deployed, a hybrid approach is desirable. In this approach, the data payload (that does not need to be processed at the core nodes) is kept in the optical domain, whereas the control information can go through optical-electrical-optical conversions to be efficiently processed in the electrical domain. OBS offers exactly that, by separating the control and the data in the physical space, that is, the control information and the data propagate discretely in different designated channels, and maybe with different data rates. OBS also separates the control and the data in time, that is, the control packet is transmitted at an offset time prior to its corresponding data. In the buffer-less core nodes, the offset time is to compensate for both processing and configuration time, needed respectively by the control unit to process the control information and the switching-fabric to be configured.

Figure 1 shows some of the main components of an OBS network. There are two types of nodes, edge (ingress/egress) nodes and core nodes. In the edge nodes, network traffic is collected from

Figure 1. Illustration of optical burst-switched network components



access networks and assembled into macro data units called data bursts (DBs). Core nodes serve as transit nodes at the core network. Their main task is to switch bursts (data) all-optically without any conversion or processing. The switching fabrics are configured according to the control information contained in the Burst Control Packets (BCPs), which are transmitted as reservation requests at an offset time ahead of their corresponding data bursts.

In the literature, various OBS switching protocols with different tradeoffs can be found. The OBS switching protocols differ in the choice of the offset time that can be zero or more, and the bandwidth reservation and release techniques (bandwidth holding time).

Three main OBS implementations can be identified, based on the offset time which is defined as the interval of time between the transmission of the first bit of the BCP and the transmission of the first bit of the data burst. The offset time can be nil, round-trip dependent, or source-routing dependent.

- **Nil offset time:** This OBS implementation is the closest to optical packet switching, and it is similar to ATM Block Transfer and Immediate Transmission (ABT-IT). The DBs and their corresponding BCPs are transmitted on different wavelengths and separated by a zero or a negligible offset time. The BCPs reserve the network resources

(wavelength and buffer) upon arrival to the core nodes. The DBs are buffered until the BCPs are processed and the switching fabrics are configured. Currently the buffers are hard-to-implement expensive FDLs that can buffer the data optically only for a short fixed time. Therefore, this implementation is feasible only when the switch configuration time (including the BCPs processing time) is very short.

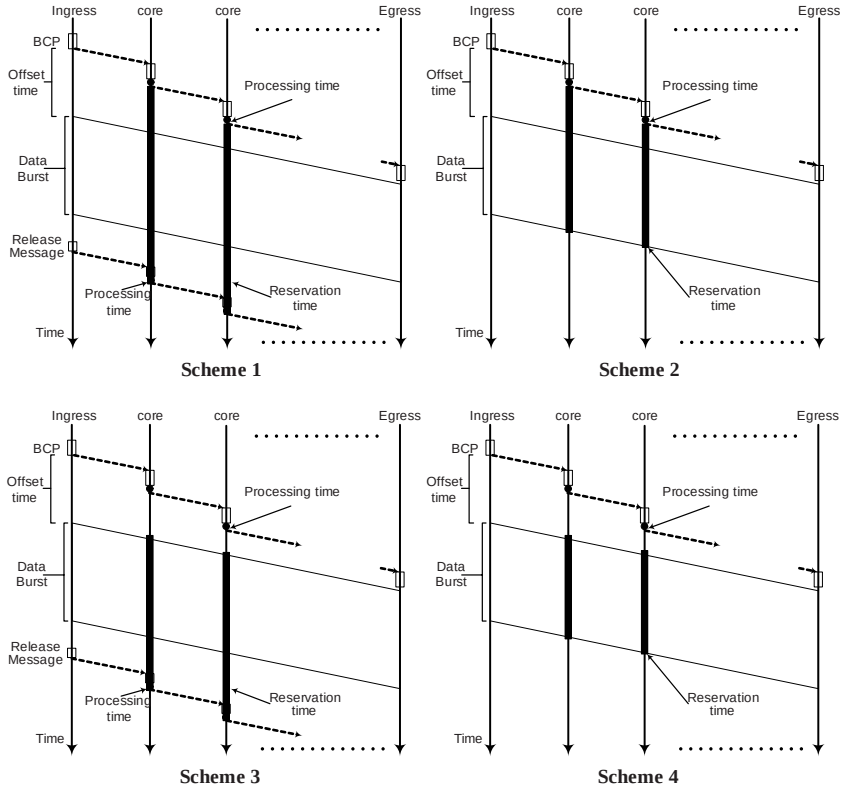
- **Round-trip-dependent offset time:** The offset time is equivalent to the round-trip time, that is, the time between the sending of a BCP and the receiving of its corresponding acknowledgment. In this implementation, a DB is transmitted only after receiving a positive response (acknowledgment) to its resource reservation request sent via the BCP, therefore, guaranteeing an uninterrupted all-optical transmission path for the DB from source to destination. Suffering from lengthy offset time due to the two-way propagation delay needed to set up an end-to-end lightpath before transmitting the user data, this scheme is the closest to optical circuit switching.
- **Source-routing-dependent offset time:** This implementation is similar to the ATM Block Transfer and Delayed Transmission (ABT-DT). The ingress node sends each control packet, followed by its corresponding data burst after an offset time. Thus, data bursts are sent without waiting for any acknowledgment or feedback on the reservation requests. This one-way reservation scheme avoids the two-way-reservation propagation delay in order to reduce the data waiting time at the ingress nodes. In this scheme, to efficiently calculate the offset time, some sort of source routing is used to determine the number of intermediate nodes from source to destination.

Channel Reservation Schemes

Under the s*ource-routing-dependent offset time* implementation, four channel reservation schemes can be identified. Depending on when the channel reservation starts and ends, the reservation schemes differ in their burst-dropping performance and their complexity. Accompanied by the illustration in Figure 2, the schemes are briefly explained as follows:

- **Scheme 1:** Each BCP contains the offset time that separates it from its corresponding DB, but not the duration of the DB. Thus, the core nodes reserve a channel as soon as they receive the BCP (reservation request). The channel remains reserved until the core nodes receive a release message that follows the transmitted DB. Therefore, for each channel, only an on/off ($1_2/0_2$) flag should be kept in each core node to indicate whether the channel is busy (reserved) or free (available). This scheme is usually referred to as Explicit Setup and Explicit Release scheme.
- **Scheme 2:** Each BCP contains the offset time that separates it from its corresponding DB and the duration of that DB. Thus, the core nodes reserve the channels as soon as they receive the BCP. Since the end of each DB is known (calculated using the duration of the DB), the channels are reserved only until the end of the DB. This scheme is more complex than the previous scheme, as each channel should be associated with a timer, which indicates when the channel will become available. This scheme is referred to as Explicit Setup and Estimated Release scheme.
- **Scheme 3:** Similarly to “scheme 1”, each BCP contains the offset time that separates it from its corresponding DB, but not the

Figure 2. One-way-based channel reservation schemes



duration of the DB, and the reserved channels are released only when the core nodes receive the corresponding release messages. However, the reservation of the channels starts approximately at the time of the DBs arrivals. This scheme is referred to as Estimated Setup and Explicit Release scheme.

- **Scheme 4:** Similarly to "scheme 2", each BCP contains both the offset time that separates it from its corresponding DB and the duration of that DB as well. Since both the beginning and the end of each DB is known (calculated respectively using the offset and the duration of the data burst), the reservation starts at the beginning of the burst and ends at its end. The reservation process is controlled using a two-element array that its elements correspondingly refer to the arrival and departure time of the burst to/from the core node. This scheme is referred to as Estimated Setup and Estimated Release scheme.

Several signaling protocols were developed for OBS networks, with different tradeoffs. For example, Just-In-Time (JIT) was developed by Wei and McFarland (2000) under Scheme 1. The Horizon protocol was proposed by Turner (1999)

under Scheme 2; under Scheme 4, Just-Enough-Time (JET) was developed by Qiao and Yoo (1999). JET was further extended by Gauger, Dolzer, and Scharf (2002) to be deployed in switches with optical buffers. For the complete operation of these signaling protocols, a variety of Channel Scheduling algorithms were proposed in the literature. Channel scheduling algorithms are used by the schedulers in the core nodes to determine the status of each channel (or possibly the optical buffer), on which the channel reservation decision is based. A channel is said to be scheduled, if it is occupied/reserved by a data burst. The time that a channel is unscheduled is known as "void", that is, the time that the channel is not occupied between two successive bursts. Schedulers use the information associated with the BCPs, for example, offset time (burst arrival time), and burst length (burst exit time). In certain Scheduling algorithms, the schedulers need to keep track of the voids between scheduled burst. Three main channel scheduling algorithms can be found in the literature: (1) First Fit Unscheduled Channel (FFUC) algorithm, (2) Latest Available Unscheduled Channel (LAUC) algorithm, and (3) Latest Available Void Filling (LAVF) algorithm. The implementation and the working of these algorithms are beyond the scope of this chapter.

Node Architecture and Switching Technology

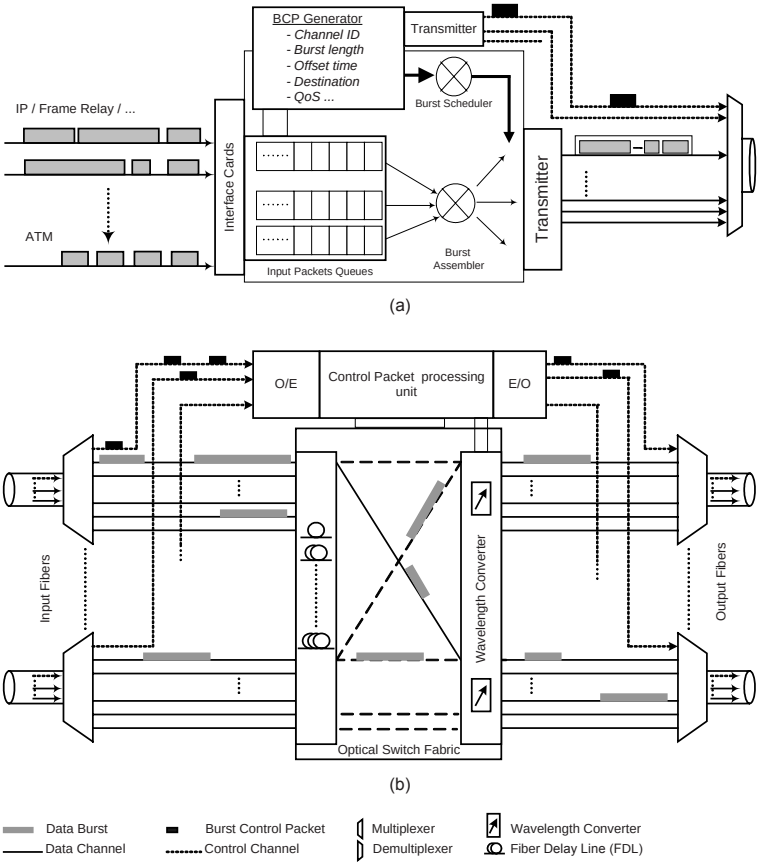
The switching technology is a very critical architectural aspect in the design of OBS networks. In order to effectively operate an OBS network with efficient bandwidth utilization, the switching time should be negligible compared to the mean transmission time of the data bursts. Therefore, both the deployed switching technology and the mean burst length have to be chosen appropriately. For example, an OBS network using fiber links at the rate of 10 Gbps to transmit a burst of about 1 Mbyte cannot adopt Micro Electro-Mechanical Systems (MEMS) (Suzuki, Otani, & Hayashi, 2003), since they require switching times between 1-10 ms, which is greater than the transmission time of an entire burst. However, if the mean burst duration is selected properly, both the Acousto-Optic Tunable Filters (AOTFs) (Sadot & Boimovich, 1998) and Semiconductor Optical Amplifiers (SOAs) (Renaud, Bachmann, & Erman, 1998) could be used as switching technology for OBS networks, as their switching time is respectively below 10 μs and 10 ns.

The OBS nodes are classified as edge nodes or core nodes. Edge nodes (Ingress/Egress) provide an interface between the OBS network (optical domain) and legacy access networks. Besides serving as an interface point between access networks and OBS, an ingress node should be capable of performing bursts assembly, BCPs generation, burst/channel scheduling, and routing. The ingress architecture is illustrated in Figure 3(a). Generally an ingress node will contain input interface cards to legacy networks, BCPs generator, bursts assembler, scheduler, and optical transmitters. In addition to the output interface cards, the Egress node contains bursts disassembler, BCPs processor/terminator, and optical receivers. Note that a generic edge node should have the functionalities of both the ingress and the egress nodes. As intermediate nodes between the edge nodes, the core nodes are located inside the OBS network. A general architecture of OBS core node is shown in Figure 3(b). The OBS core node (switch) should be equipped with the capabilities of routing, bursts switching (optically), and BCPs processing (electronically). The capabilities of a core node may be farther extended to provide optical data buffering and wavelengths conversion.

A generic core node contains the following:

- **Optical receivers/transmitters:** Reception/Transmission of the control information and data bursts; *Optical multiplexer and de-multiplexers:* Responsible for optical channels multiplexing and de-multiplexing;

Figure 3. (a) Architecture of OBS egress node, (b) Architecture of OBS core node



- **Input/Output interface for BCP:** Control reception/transmission and O/E/O conversions;
- **Control packet processing unit:** For BCPs interpretation, channels scheduling, collision resolution, routing, and optical switching-fabric control;
- **Optical switching fabric:** To optically switch data bursts; And possibly *Wavelength converters* and *fiber delay lines (FDLs).*

Data Burst Assembly

At the edge node, the burst assembly/disassembly process is implemented. A very important design parameter in OBS networks, the assembly process takes place at the ingress nodes. Burst assembly (burstification) is the process of aggregating a number of smaller transport units, for example, IP packets or ATM cells, into larger transport units called data bursts. Motivated by the need

of optimizing the switching performance in the optical domain, larger transport units allow the switches to switch the same amount of data by handling less requests, since the ratio between the control information and transmitted data gets smaller. Furthermore, the efficiency of optical switching is improved due to the reduction in the mean inter-arrival time between the switched data units (bursts).

Generally, the assembly algorithms are based on a threshold that could be time, burst-length, or hybrid time-length threshold. In a single data burst, the assembled data units may belong to various upper layer traffic flows, and ultimately destined to different final destinations; however, the destination OBS egress node of all the data units must be the same. The arriving upper layer packets are stored in appropriate queues (according to their egress destination, QoS requirement, Class of Service (CoS)) in the burst assembler. A new burst is constructed and its control packet is generated when the threshold is reached. Obviously, the generated traffic will have variable burst length and constant inter-arrival time, if the algorithm operates only based on time as threshold; however, this may produce excessively large data bursts if the upper layer traffic streams are experiencing high loads. Conversely, if the algorithm operates based on the burst-length threshold, the inter-arrival time will vary, while the burst length is maintained constant. Optimally, a hybrid approach is used, that is, a combination of the time and length thresholds. In all cases, a minimum burst-length should be imposed to avoid too small bursts, which may cause the generation of too many control packets that may lead to traffic congestion on the control channels. A data burst is constructed when its length gets bigger than a pre-defined value, or when the maximum assembly time is reached; in the later case, data padding may be applied to bring the data burst to an appropriate length. Furthermore, a minimum burst-length should be imposed to avoid too small bursts, which may cause the generation of too many control packets that may lead to traffic congestion in the control channels.

More sophisticated assembly algorithms that trade-in the simplicity with the flexibility were proposed in the literature. For example The Adaptive-Assembly-Period (AAP) Algorithm, proposed by Cao, Li, Chen, and Qiao (2002). AAP analyzes current IP traffic statistics and change the values of thresholds accordingly, which can reduce the queuing delay in the edge nodes.

The data bursts disassembly (de-burstification) process is performed by disassemblers at the egress nodes upon the bursts' arrival. In the egress-nodes, the BCPs are processed and terminated, and the data bursts are disassembled to the initial data units. The data units are then forwarded to their final destinations.

QoS Support and Contention Resolution

The number of users (with different needs) and electronic applications (with different requirements) is growing exponentially on the Internet, causing the Internet to start suffering from its own success, in terms of bandwidth and service differentiation. Using optical infrastructure, the available capacity limits could be solved by bandwidth over-provisioning. However, because IP is a best-effort protocol, there is a need to incorporate QoS mechanisms and flow control capabilities.

OBS-QoS Mechanisms

A great effort has been directed to the QoS provisioning in the Internet, where many QoS mechanisms were introduced. However, the mechanisms that worked for electronic packet switched networks did not find the same success in optical networks, because of the lack of efficient optical buffers, which reduces the switches'

scheduling capabilities. In this section, we will present an overview of the main QoS provisioning strategies proposed for OBS networks.

- **Offset-Based QoS Scheme:** Offset-based QoS (Fan, Feng, Wang, & Ge, 2002) scheme adds an extra offset time to the basic offset between the BCP and its corresponding DB. The additional offset time, called QoS offset, is to compensate for the processing time of the BCP. The duration of such a QoS offset is varied, depending on the priority of the service class. This scheme is proposed for JET, whereby higher priority classes have larger offsets. With this scheme, if a low priority DB with no additional QoS offset time and a higher priority DB with a QoS offset try to make network resources reservation, the DB with the larger offset will be able to reserve resources in advance, before the low priority DB. In general, this will result in a lower burst loss probability of high priority classes compared to the lower priority classes.

Although the offset-based QoS scheme does provide an acceptable service differentiation, it is faced with some challenges that cannot be ignored. For example, DBs of high priority classes suffer longer waiting time (delay) than the data bursts of low priority classes. Furthermore, the scheme is non-preemptive that is, as long as low priority DBs can block optical paths, no complete isolation is achieved. Yet, starvation of low priority classes is possible if the offered traffic load of high priority bursts is high and not controlled.

- **Active dropping-based QoS scheme:** In this scheme, a burst dropper (hardware) is implemented in front of every core node (Chen, Hamdi, & Tsang, 2001). Dependent on a dropping policy, some BCPs and their corresponding DBs are dropped before reaching the reservation unit. Therefore, the admission to the outgoing wavelengths is controlled, enabling the core nodes to locally control the offered load of certain service classes to maintain network resources for other service classes. Active Dropping-based QoS scheme intervene before congestion occurs, as the selective dropping of DBs is initiated according to the data traffic profile to guarantee that the higher priority classes have higher chances to make successful reservations. However, this scheme suffers from a major disadvantage which is the absence of feedback from the core nodes to the edge nodes, and thus traffic volume of different classes cannot be controlled. Furthermore, isolation between different traffic classes is not guaranteed. If the offered traffic load of a low priority class is significantly augmented, which increases the overall burst loss probability; burst loss probabilities of all classes are increased. Therefore, an additional traffic control mechanism is required.
- **Segmentation-based QoS scheme:** In segmentation-based QoS scheme (Vokkarane & Jue, 2002), each data burst is subdivided into several independent segments. If DBs contend for the same network resources, the contention is resolved by discarding or deflecting some segments of one of the contending data bursts. The remaining part of the burst (truncated DB) will be then forwarded to the downstream nodes where it will experience either more shortening, be dropped, or be delivered to the egress node. Unfortunately, this scheme is implemented at the cost of increasing the size of the control packets, since the BCP should at least contain the segments number, the burst length, and the routing information.

Furthermore, the implementation of burst segmentation strategies is faced by some challenges and practical issues summarized as follows:

- **Switching time (ST):** ST is the time needed to reconfigure the switching fabric. ST depends on the design and implementation of the core node and on the used switching technology. ST may differ from one core node to another.
- **Data burst size:** Since the transmission of DBs depends on the transmission of their BCPs, the DB length should agree with minimum and maximum length requirements, to avoid congestion in the control channels. The same is true for the truncated burst (i.e., DB that lost some of its packets).
- **Segment Delineation:** Since the data bursts are transmitted all-optically, the segments' boundaries are transparent to the core nodes, and their sizes are not reflected in the BCP.
- **Fiber Delay Lines (FDLs):** As in Optical Composite Burst Switching (OCBS) (Detti, Eramo, & Listanti, 2002), FDLs are needed to delay the data bursts while their control packets are being electronically updated with the new burst size, which increases the electronic processing time needed before forwarding the control packet to the next node.
- **Trail-control messages:** Generated by the node where the DB has been truncated, the trailing control message is needed to indicate the data burst's new size to the downstream nodes, to avoid unnecessary resource reservation, and needless contention resolution actions.

Contention Resolution

In OCS, each traffic flow is supplied with the appropriate network resources, precisely with the necessary bandwidth. Therefore, flows would not contend for the same network resources in the core nodes of an OCS network. However, contention is a major problem in both OPS and OBS, which are based on multiplexing gain between the traffic flows. Typically, in electronic packet switching, this problem is solved using random access electronic buffers; however, this solution is far from being feasible in all-optical networks where optical buffering is yet to be a viable technology. In OBS, contention occurs when two or more bursts are destined to use the same channel from the same fiber (i.e., output port) at the same time. The contention can be resolved in one of three techniques. The contending burst can be optically delayed/buffered (time domain), converted to another channel in the same fiber (optical domain), or deflected to another output port (space domain).

- **Buffering:** One of the most prevailing factors that have motivated the development of OBS network is the immaturity of the optical buffering technology. Therefore, the use of optical buffers as a contention resolution technique in the OBS networks is less than feasible. Nevertheless, several proposals, and many research activities, were dedicated to the study of buffering techniques in OBS networks. Buffering is to delay or to queue contending bursts instead of dropping them. If optical-electrical-optical conversion is allowed (i.e., not an all-optical network) then random access electronic buffers can be used to buffer the burst for a long time. However, to implement an all-optical network (i.e., the data is maintained in the optical domain end-to-end), an optical buffering technique is needed. Optical buffering currently can only be implemented using Fiber Delay Lines (FDLs). The time that the bursts can be delayed is directly proportional to the length of the FDL. More efficient optical buffers can be realized through multiple delay lines of the same or different lengths, deployed in stages (Chlamtac, Fumagalli, Kazovsky, Melman, et al., 1996) or in parallel (Haas, 1993). Such buffers can hold DBs for variable amounts of time. Recently, designs

of large optical buffers (Hunter, Cornwell, Gilfedder, et al., 1998; Tancevski, Castanon, Callegati, & Tamil, 1999) were proposed without large delay lines. However, in any optical buffer architecture, the size of the buffers and their flexibility is severely constrained by physical space and the effects of fiber dispersion. Therefore, fiber delay lines could be suitable for prototype switches in experimental environments, but not for real and commercial deployments.

- **Wavelength conversion:** WDM systems are multiple channel systems. Optical switches are connected with fiber links that are expected to carry hundreds of channels in the near future. This can be exploited to resolve traffic flow contentions. If two traffic flows contend for the same channel, one of the flows can be wavelength-converted to a free channel, and transmitted on the same link. It is clear that with this technique the contentions between data bursts in an OBS network can be minimized. However, using wavelength converters will increase the overall hardware cost of the WDM network, in addition to the complexity and technical issues associated with the wavelength conversion technology itself. Wavelength-conversion-enabled optical networks can be categorized to either: (1) networks with full wavelength conversion (any channel can be converted/shifted to any other channel); (2) networks with limited/fixed wavelength conversion (only limited/fixed channels can be converted); or (3) networks with sparse wavelength conversion (not all the network's nodes have wavelength conversion capabilities)
- **Deflection:** To delay (hold) data bursts, an alternative to the use of FDLs is the use of fiber links connecting the core nodes. When two bursts contend for the same output port, one will be switched to the correct output port, and the other will be switched (deflected) to any other available output port. Therefore, both bursts are given a chance to be switched instead of dropping one of them to resolve the contention. Because the correct output port leads usually to the shortest path (networking context) toward the destination, the bursts that are not switched to the correct output ports may end up following longer paths to their destinations. This technique is referred to as "hot-potato" routing. A drawback of this technique is the complexity associated with the calculation of the offset time of the deflected burst since it will travel on a different path with different hop count. Furthermore, the end-to-end high delays experienced by the deflected DBs may cause the DBs to be out-of-order at the destination, or trigger upper layers retransmission mechanisms. Additionally, deflecting data bursts because of local congestion may instigate global congestion. In the literature, deflection routing is studied on various network topologies, with and without FDLs.

In all the three aforementioned techniques, the data bursts are treated as single fused data units. However, there is another interesting OBS approach that suggests that the bursts should be dealt with as a combination of basic transport units called segments. Each of these segments may consist of a single packet or multiple packets. The segments indicate the possible partitioning points of a burst while the burst is traveling in the optical domain. With this approach, when a contention occurs, only the overlapping (time aspect) segments of the contending bursts are dropped, buffered, deflected, or converted to a different channel. It has been demonstrated in literature (Detti et al., 2002; Vokkarane, Jue, & Sitaraman, 2002) that the contention resolution techniques based on burst segmentation are efficient, and outperformed the traditional OBS with the "entire-burst-dropping" policy. Unfortunately the

implementation of burst segmentation strategies is faced by some technical challenges, besides the complexity added to the algorithms and protocols on both edge and core nodes.

Flavors of OBS Networks

Besides the traditional definition presented at the beginning of this chapter, the following implementations of OBS can be found in literature; each implementation has its advantages and disadvantages.

- **OBS with fixed data burst length:** In this implementation of OBS, all data bursts have the same length. Its major advantage is the simplification of the switching technique. However, its disadvantage is the long queuing time of the upper layer packets if there is no adequate traffic load. An ineffectual solution to this disadvantage is data padding. This concept is closely related to the cell-switching paradigm where the transport unit is fixed in size.
- **OBS with two-way setup:** In this implementation of OBS, a burst will be transmitted only after having received a positive acknowledgement of its resource reservation request. Its advantage is the burst-blocking avoidance in the network's core nodes. However, the additional waiting time may cause buffers to overflow at the network ingress.
- **Mandatory FDL/Wavelength Converters in all nodes:** With the disadvantage of increasing dramatically the cost of the network, OBS could be implemented with core nodes that are capable to buffer the data bursts and/or convert them to any other wavelength. In this implementation some sort of store-and-forward routing for optical bursts is deployed, where each burst is delayed for a certain time until its corresponding control information is processed and an appropriate switching decision is determined, that is, the burst could be forwarded to an output port, converted to another wavelength, or simply dropped.
- **OBS networks for Ethernet:** Extending Ethernet services over OBS was proposed in Sheeshia, Qiao, and Liu (2002) as a more scalable and bandwidth-efficient implementation compared to the Ethernet over Synchronous Optical Network (SONET). The possible role that OBS will play in the development of 10-Gbit Ethernet (10GbE) metropolitan networks was investigated. OBS was demonstrated as an improved network resource-sharing platform and efficient transport for Ethernet traffic, particularly if it is coupled with the Generalized Multiprotocol Label Switching (GMPLS).

It is clear that OBS is a technology-driven paradigm; consequently, the architecture and the operational policies of the future OBS networks will be shaped by the upcoming developments in the multiplexing and switching technologies. The technology needed to generate and transmit more than 1,000 high-quality WDM channels with fixed wavelength spacing is already available. Furthermore, the Waveband Switching (i.e., switching multiple wavelengths as a single entity) is being researched. Therefore, the future OBS would be based on waveband-selective switching or other more advanced switching technologies. In the rest of the chapter, an attempt to streamline the OBS networks that are using burst segmentation techniques is presented.

STAGED RESERVATION SCHEME (SRS)

The Staged Reservation Scheme (SRS) (Abid, Abbou, & Ewe, 2005) is introduced to streamline previous reservation protocols that are based on burst segmentation strategy, to increase the throughput of the core nodes, and to overcome

some of the limitations associated with the burst segmentation concept (shown earlier). In SRS the data burst is divided into equal Data Segments (DSs). This is an indication to where the DB might be partitioned, while traveling in the optical-domain. The DSs have the same attributes and characteristics as any DB (i.e., each segment may range from one to several packets, and its length is reflected in the BCP). The control packets format was designed to be scalable to high transmission speeds, and to reflect the DSs' length in addition to the DBs' length.

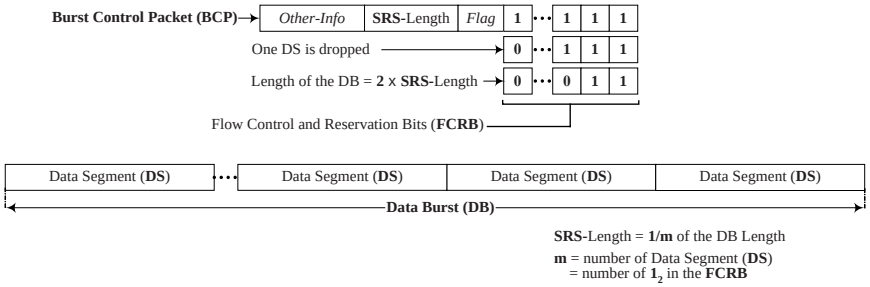
BCPs Format in SRS

The burst control packet format in SRS is redesigned and changed from the traditional format to provide a constant transmission overhead and to make the BCP scalable to higher speeds, as it uses the *Flow Control and Reservation Bits* (FCRB) as the segments' length indicator instead of flags (Detti et al., 2002). The illustration shown in Figure 4 of the format of the BCP is briefly presented here.

- **FCRB field:** *Flow Control and Reservation Bits* field is created by the ingress-node to reflect the permitted segmentations. In the core-nodes, the SRS-length is multiplied by the number of 1_2 in FCRB to obtain the actual size of the corresponding DB. For example 0111_2 is an indication that the length of the DB (or truncated DB) is (3 * SRS-length), and it might be segmented into three segments. The size of FCRB is dynamic in that it may vary from one DB to another, and the burst assembly algorithm controls it.
- **Flag field:** The field is a sequence of bits with a recognizable pattern that identifies the end of the FCRB field (as its size is not fixed), and identifies the beginning of the SRE-length field.
- **SRS-length field:** The field contains the length of one DS. However, SRS-length combined with FCRB provides sufficient information about the DB's length and segmentation. To avoid congestion in OBS control-channel, SRS-length should comply with a minimum length (Detti et al., 2002), which is the minimum permitted data burst length transmitted over the optical links. The SRS-length may vary from one DB to another.
- **Other-Info fields:** The rest of the fields may contain routing information (e.g., burst destination address), offset time, and so forth.

By adopting SRS dropping policy, which will be described in the following subsection, if a contention is anticipated in the core-nodes, the

Figure 4. Data burst structure and burst control packet format in SRS scheme



resources allocation process will not be aborted (i.e., BCP is dropped and subsequently the corresponding DB is entirely discarded). Conversely, the FCRB field in the corresponding BCP is updated according to the resources that the core-node can provide (or free up). Hence, only the overlapping segments are dropped at the arrival time, allowing part of the DB to be transmitted (i.e., the data length that can be handled by the node at the arrival time) as shown in Figure 5. Since the BCPs are updated before forwarding them to the downstream nodes, to reflect the new DBs' length, the need for trailing messages is eliminated, and the contention is resolved at the BCPs level rather than at the DBs level.

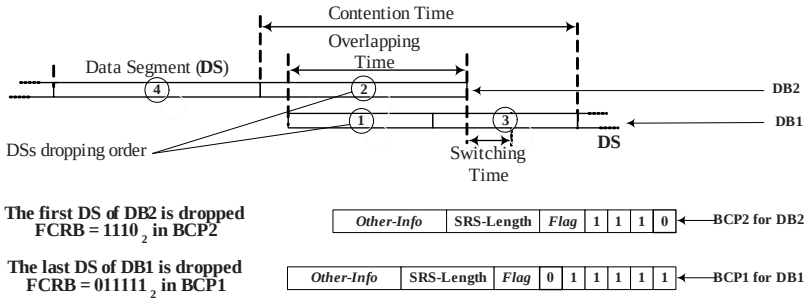
SRS Dropping Policy

Beside the *Switching-Time* (ST), the dropping policy is based on the *Overlapping-Time* (OT), and the *Contention-Time* (CT) (i.e., the aggregated length (duration) of all the DSs that are entirely or partly overlapping minus the OT), as illustrated in Figure 5. The OT may span many DSs from the contending DBs. Therefore; the overlapping DSs are dropped alternately one-by-one from the contending DSs until the CT is reached. This guarantees a reasonable level of fairness between the data bursts (i.e., the number of discarded DSs is distributed among the contending DBs). The dropping policy rules are formulated in an algorithm that will be presented when the fairness in network resources allocation in the core nodes is discussed.

The contention, therefore, is resolved by performing the following three steps:

- Firstly, if ***(OT + ST) > CT***, then a DS is added to the CT to efficiently contain the ST in the dropped DSs. The added DS is selected from the DB with the smallest SRS-length, or randomly if the contending DSs are equal in length. The ST is assumed to be much smaller than the duration of one DS.
- Secondly, if the "***SRS-length***" is the same for both contending DBs (i.e., the DSs are equal in length), then the dropping process starts from the DS at the tail of the first DB, not at the head of the contending DB. This gives fair chance to the packets to arrive to the destination in the correct sequence (assuming the retransmission of the lost packets).
- Thirdly, if the "***SRS-length***" is not the same for the contending DBs, then the dropping process starts at the DS of the shorter DB. The DS may be at the tail of the first DB or at the head of the contending DB (as it is

Figure 5. SRS dropping policy with relation to CT, OT, and ST



deemed that the amount of transmitted data has higher priority than its order).

Fairness in Resources Allocation in SRS Core Node

A key parameter in the design of an OBS network is the maximum and minimum burst size, which is managed by the edge nodes using the assembly algorithms. This key parameter is entirely overlooked by the resource allocation schemes based on the burst segmentation concept, since no policies related to the size of the truncated burst (i.e., shortened data burst) are implemented, during the burst segmentation process in the core nodes. Furthermore, there is no fairness in allocating the network resources to the contending data bursts (usually form different sources), as all the segments are simply discarded from only one burst to resolve the contention.

A better solution would be selecting evenly (fairly) the segments to be dropped from both contending data bursts. Likewise, the truncated burst size should be monitored at the core nodes, and guaranteed to be larger than the Minimum Burst Length (MBL), which is the minimum length allowed into the network to avoid congestion in the control channels.

Besides fairness, data burst size, and implementation simplicity, any proposed technique should be designed to deal with the switching time, which is the time needed to configure the switching fabric (i.e., to switch an output port from one DB to another). To understand what follows, the following definitions are provided and illustrated in Figure 6.

- DB_O: Original Data Burst with Arrival time T_{OA} and Leaving time T_{OL},
- DB_C : Contending Data Burst with Arrival time T_{CA} and Leaving time T_{CL},
- *TDB*: Truncated Data Burst (i.e., a DB with dropped segments),
- *N*, *M*: Respectively, the number of segments in DB_O, DB_C,
- *DS*: Data Segment with Length *DSL*,
- *R*: the expected number of segments to be dropped from each data burst\

$$R = \left\lceil \frac{|T_{CA} - T_{OL}|}{2DSL} \right\rceil^{*}$$

When data bursts arrive to a core node, the technique performs three functions arranged in three main events. After an initialization (Table 1), the *Contention_Detection* (Table 2) event is

Figure 6. Illustrations: (a) Segments dropping process, (b) DB structure in SRS-based QoS scheme

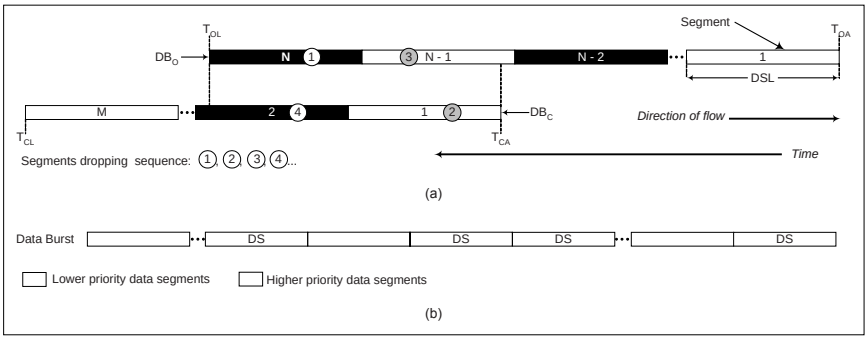


Table 1. Initialization

```
Initialization
 /* Original data burst length. */
 DBL_o = T_OL – T_OA = N * DSL
 /* Contending data burst length.  */
 DBL_c = T_CL – T_CA = M * DSL
 /* Length to be dropped from each data burst. */
 LTD = R * DSL
```

Table 2. Contention detection

```
Event :: Contention_Detection
/* There is a contention if the condition is true */
   IF ((T_CA – T_OL) < ST) THEN
      R = ⌈ |T_CA – T_OL| / 2DSL ⌉  /* calculate R. */
      Execute: Length _of _Truncated_Burst
   END IF
End of Event
```

Table 3. Length of truncated burst

```
Event :: Length _of _Truncated_Burst
/* The truncated burst will be less than the allowed
burst length if the condition is true */
   IF ((DBL_o – LTD < MBL) || (DBL_c – LTD < MBL)) THEN
      IF (DBL_c < DBL_o) THEN
        /* DB_C is the smallest, therefore, dropped */
        Drop DB_c
      ELSE
        /* DB_O is the smallest, therefore, dropped */
        Drop DB_o
      END IF
   ELSE
       Execute: Even_Resource_Allocation
   END IF
End of Event
```

performed; if a contention is detected, then *R* (number of segments to be dropped from the contending data bursts) is calculated, and the second event is executed. The second event named *Length_of _Truncated_Burst* (Table 3) is executed to guarantee that whatever is left from the data bursts after dropping some of their segments is good for transmission over the OBS network. In this event, if one of the truncated data bursts does not meet the MBL requirements, then the conten-

Table 4. Even resource allocation

```
Event :: Even_Resource_Allocation
/* Initialize counter */
i = 1
   DO
   {
   IF (i % 2 = 0) THEN
   /* DB_C is reduced by one DS from the burst head*/
      T_CA = T_CA + DSL
   ELSE
   /* DB_O is reduced by one DS from the burst tail*/
      T_OL = T_OL – DSL
   END IF
   /* Increase the counter i by 1 */
i++
   }
   WHILE ((T_CA – T_OL) < ST)
End of Event
```

tion is simply resolved by dropping the shortest data burst in its entirety. However, the third event is executed if the truncated data bursts are larger than MBL. The *Even_Resource_Allocation* (Table 4) is used to resolve the burst contention by discarding the overlapping segments *alternatively*, starting from the tail of the original burst, and then the head of the contending burst as shown in Figure 6(a).

SRS-Based QoS Scheme

Motivated by the SRS design, its dropping policy, and the use of the FCRB field, the SRS-based QoS mechanism is introduced. In this mechanism, the QoS requirements of the upper layer packets are defined based on their traffic class. Packets of the same class and destination are assembled into the same data segment, which will be labeled with a priority number accordingly. A data burst may contain data segments of the same or different priorities. Using an appropriate assembly algorithm, the data segments are assembled into data bursts in such a way that the lower priority data segments envelop the higher priority data segments as shown in Figure 6(b). To realize the SRS-based QoS scheme, both the edge nodes and core nodes must cooperate, using an assembly algorithm in the edge nodes and the SRS dropping policy in core nodes.

Assembly Algorithm

Although either timer-based or burstlength-based assembly algorithm could be used, it is preferred to use hybrid assembly algorithms. In the hybrid approach, first a burst length threshold is used in assembling the data segments; this allows the control of the data segments size, which must be fixed and restricted to a maximum length. (In some cases, data padding may be needed.) Second, in constructing the data bursts (from data segments) and to control the packets delay at the edge nodes, a timer threshold is used; that is, after a fixed time, all the data segments (could be of different priorities) destined to the same egress are assembled into a data burst. The data bursts are then transmitted to their destinations

through the core nodes to undergo the SRS dropping policy.

FCRB for Congestion Control

In the course of discussing the QoS, congestion in networks appears to cause real problems by reducing the availability and the throughput of the network. Congestion is a complex phenomenon, and it occurs when the traffic load (number of bursts) on the network begins to approach the network capacity. Therefore, a congestion control mechanism is needed to maintain the number of the bursts being transmitted through the network within the limits at which the network performance is acceptable.

By using an explicit congestion avoidance technique, the edge nodes can use as much of the network capacity as possible, while reacting to the congestion in a controlled manner. In the proposed SRS-based QoS, an explicit signaling technique is used. In this signaling technique, the bits of FCRB are used to indicate explicitly the amount of data (i.e., the number of data segments in the DB) sent and the arrived amount. This signaling approach can work in one of two directions: forward (to notify the Egress), or backward (to notify the Ingress).

- **Forward signaling:** notifies the egress node that congestion procedures should be initiated where applicable for traffic in the opposite direction of the received bursts. It indicates the number of the dropped data segments, and that the received burst has encountered congested resources. This information could be sent back to the source node, and the end system will exercise flow control on the traffic sources at the higher layers (e.g., TCP).
- **Backward signaling:** notifies the ingress node that congestion procedures should be initiated where applicable for traffic in the same direction as the sent bursts. It indicates the number of data segments dropped, and that the sent burst has encountered congested resources. The ingress node will then lower the number of data segments sent in each DB to be equal to the number of data segments that could get through the network to the destination. Then the number of data segments is augmented progressively until the maximum size of the data burst is reached, or until the FCRB field reports congestion.

SUMMARY

The network infrastructure has a decisive role in the success of any e-services provider. The next-generation network infrastructure is expected to be reliable, efficient, and supported by robust QoS schemes. The success of IP, with its flexibility, and the advent of WDM technology, with its advantages, are behind the widespread interest in selecting the IP/WDM internetworking model to build an All-optical Internet. Eventually, the next-generation network will be an all-optical Internet where the traffic flows are optically transmitted, switched/routed, tuned/amplified, and buffered.

The OBS is a major candidate to be deployed as the switching paradigm for the Next-Generation Optical Internet. OBS is flexible, manageable, and offers a good balance between OPS and OCS. Furthermore, OBS is amenable for improvement and expansion whenever the developments in optical technology permit.

In this chapter, a broad discussion on optical switching is presented. Many architectural and design aspects of OBS were reviewed, including burst assembly, signaling and reservation protocols, and QoS provisioning. At the end of the chapter, the recent work in OBS schemes is covered; however, the OBS paradigm remains an open field for more research and improvement.

REFERENCES

Abid, A., Abbou, F. M., & Ewe, H. T. (2005). Staged reservation scheme for optical burst switching networks. *IEICE Electronics Express, 2*(11), 327-332.

Amstutz, S. (1989). Burst switching—An update. *IEEE Communication Magazine,* 50-57.

Baldine, I., Rouskas, G., Perros, H., & Stevenson, D. (2002). JumpStart: A just-in-time signaling architecture for WDM burst switched networks. *IEEE Communications Magazine, 40*(2), 82-89.

Callegati, F., Cankaya, H. C., Xiong, Y., & Vandenhoute, M. (1999). Design issues of optical IP routers for Internet backbone applications. *IEEE Communications Magazine, 37*(12), 124-128.

Cao, X., Li, J., Chen, Y., & Qiao, C. (2002). Assembling TCP/IP packets in optical burst switched networks. In *Proceeding of IEEE Globecom*.

Chen, Y., Hamdi, M., & Tsang, D. H. K. (2001). Proportional QoS over OBS networks. In *Proceedings of IEEE GLOBECOM 2001, San Antonio.*

Chlamtac, I. , Fumagalli, A., Kazovsky, L. G. , Melman, P. et al. (1996). CORD: Contention resolution by delay lines. *IEEE Journal on Selected Areas in Communications, 14*(5), 1014-1029.

Detti, A., & Listanti, M. (2001). Application of tell and go and tell and wait reservation strategies in an optical burst switching network: A performance comparison. In *Proceedings of the 8th IEEE International Conference on Telecommunications (ICT 2001), Bucharest.*

Detti, A., Eramo, V., & Listanti, M. (2002). Performance evaluation of new technique for IP support in a WDM optical network: Optical composite burst switching (OCBS). *IEEE Journal of Lightwave Technology, 20*(2), 154-165.

Dolzer, K. (2002). Assured horizon—A new combined framework for burst assembly and reservation in optical burst switched networks. In *Proceedings of the European Conference on Networks and Optical Communications (NOC 2002), Darmstad.*

Dolzer, K., Gauger, C., Spath, J., & Bodamer, S. (2001). Evaluation of reservation mechanisms for optical burst switching. *AEÜ International Journal of Electronics and Communications, 55*(1).

Fan, P. , Feng, C., Wang, Y., & Ge, N. (2002). Investigation of the time-offset-based QoS support with optical burst switching in WDM networks. In *Proceeding of IEEE ICC: Vol. 5* (pp 2682-2686).

Gauger, C., Dolzer, K., & Scharf, M. (2002). Reservation strategies for FDL buffers in OBS networks. In *Proceedings of the IEEE International Conference on Communications.*

Ge, A., Callegati, F., & Tamil, L. S. (2000). On optical burst switching and self-similar traffic. *IEEE Communications Letters, 4*(3), 98-100.

Haas, Z. (1993). The "staggering switch": An electronically-controlled optical packet switch. *IEEE Journal of Lightwave Technology, 11*(5/6), 925-936.

Hu, G., Dolzer, K., & Gauger, C. M.. (2003). Does burst assembly really reduce the self-similarity. In *Proceedings of the Optical Fiber Communication Conference (OFC 2003), Atlanta.*

Hunter, D. K., Cornwell, W. D., Gilfedder, T. H. et al. (1998). SLOB: A switch with large optical buffers for packet switching. *IEEE Journal of Lightwave Technology, 16*(10), 1725-1736.

Kulzer, J., & Montgomery, W. (1984, May). *Statistical switching architectures for future services.* Paper presented at ISS '84, Florence.

Mukherjee, B. (1997). *Optical communication networking.* McGraw-Hill

Qiao, C., & Yoo, M. (1999). Optical burst switching (OBS)-A new paradigm for an optical Internet. *Journal of High Speed Networks, 8*(1), 69-84.

Qiao, C., & Yoo, M. (2000). Choices, features, and issues in optical burst switching. *Optical Networks Magazine, 1*(2), 37-44.

Renaud, M., Bachmann, M., & Erman, M. (1996). Semiconductor optical space switches. *IEEE Journal on Selected Topics in Quantum Electronics, 2*(2), 277-288.

Sadot, D., & Boimovich, E. (1998). Tunable optical filters for dense WDM networks. *IEEE Communications Magazine, 36*(12), 50-55.

Sheeshia, S., Qiao, C., & Liu, U.J. (2002). Supporting Ethernet in optical-burst-switched networks. *SPIE the Journal of Optical Networking, 1*(8/9), 299-312.

Suzuki, M., Otani, T., & Hayashi, M. (2003). Transparent optical networks. In *Proceedings of 2003 5th International Conference: Vol. 1* (pp. 26-31).

Tancevski, L., Castanon, G., Callegati, F., & Tamil, L. (1999). Performance of an optical IP router using non-degenerate buffers.In *Proceeding of IEEE Globecom '99, Rio de* Janeiro, Brazil (pp. 1454-1459).

Tancevski, L., Yegnanarayanam, S., Castanon, G., Tamil, L., Masetti, F., & McDermott, T. (2000). Optical routing of asynchronous, variable length packets. *IEEE Journal on Selected Areas in Communications, 18*(10), 2084-2093.

Turner, J. S. (1999). Terabit burst switching. *Journal of High Speed Networks, 8*(1), 3-16.

Verma, S., Chaskar, H., & Ravikanth, R. (2000). Optical burst switching: A viable solution for terabit IP backbone. *IEEE Network, 14*(6), 48-53

Vokkarane, V., Haridoss, K., & Jue, J. (2002). Threshold-based burst assembly policies for QoS support in optical burst-switched networks. In *Proceedings of the SPIE Optical Networking and Communications Conference (OptiComm 2002),* Boston (pp. 125-136).

Vokkarane, V., & Jue, J. (2002). Prioritized routing and burst segmentation for QoS in optical burst- switched networks. In *Proceeding of OFC* (pp. 221-222).

Vokkarane, V., Jue, J., & Sitaraman, S. (2002). Burst segmentation: An approach for reducing packet loss in optical burst switched networks. *IEEE International Conference on Communications,* 5, 2673-2677.

Wei, J. Y., & McFarland, R. I. (2000). Just-in-time signaling for WDM optical burst switching networks. *Journal of Lightwave Technology, 18*(12), 2019-2037.

Wei, J. Y., Pastor, J. L., Ramamurthy, R. S., & Tsal, Y. (1999). Just-in-time optical burst switching for multiwavelength networks. In *Proceedings of the 5th IFIP TC6 International Conference on Broadband Communications (BC '99),* Hong Kong (pp. 339-352).

Xiong, Y., Vanderhoute, M., & Cankaya, C. C. (2000). Control architecture in optical burst switched WDM networks. *IEEE Journal on Selected Areas in Communications, 18*(10), 1838-1851.

Xu, L., Perros, H. G., & Rouskas, G. N. (2001). Techniques for optical packet switching and optical burst switching. *IEEE Communications Magazine, 39*(1), 136-142.

Yao, S., Dixit, S., & Mukherjee, B. (2000). Advances in photonic packet switching: An overview. *IEEE Communications, 38*(2), 84-94.

Yoo, M., Jeong, M., & Qiao, C. (1997). A high speed protocol for bursty traffic in optical net-

works. In *Proceedings of the 3rd SPIE Conference on All-Optical Communication Systems, Dallas* (pp. 79-90).

Yoo, M., Qiao, C., & Dixit, S. (2001). Optical burst switching for service differentiation in the next-generation optical Internet. *IEEE Communications Magazine, 39*(2), 98-104.

This work was previously published in Architecture of Reliable Web Applications Software, edited by M. Radaideh & H. Al-Ameed, pp. 323-349, copyright 2007 by IGI Publishing (an imprint of IGI Global).

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z